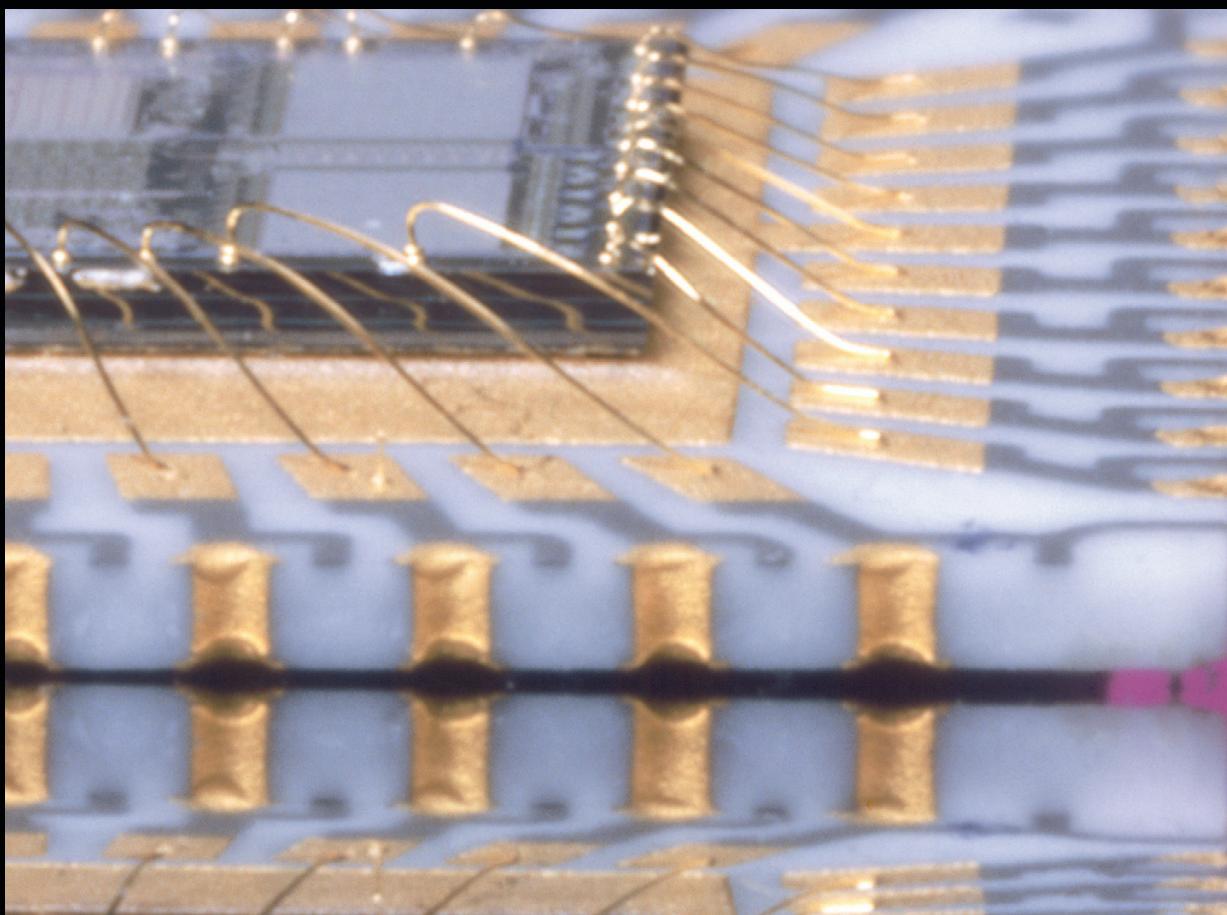


MICROPROCESADORES INTEL

**8086/8088, 80186/80188, 80286, 80386, 80486, PENTIUM,
PROCESADOR PENTIUM PRO, PENTIUM II, PENTIUM III, PENTIUM 4**

ARQUITECTURA, PROGRAMACIÓN E INTERFAZ

SÉPTIMA EDICIÓN



Barry B. Brey

CAPÍTULO 1

Introducción al microprocesador y la computadora

INTRODUCCIÓN

Este capítulo presenta las generalidades sobre la familia de microprocesadores Intel. Se incluye una discusión sobre la historia de las computadoras y la función del microprocesador en el sistema de computadora basado en microprocesador. También introduciremos los términos y la jerga propios de la computación, para que pueda entender los modismos utilizados al hablar sobre microprocesadores y computadoras.

El diagrama de bloques y una descripción de la función de cada bloque detallan la operación de un sistema computacional. Los bloques en el diagrama muestran cómo se interconectan la memoria y el sistema de entrada/salida (E/S) de la computadora personal. En este capítulo explicaremos con detalle cómo se almacenan los datos en la memoria, de manera que se pueda utilizar cada tipo de datos a medida que se vaya desarrollando el software. Los datos numéricos se almacenan como enteros, números de punto flotante y decimal codificado en binario (BCD); los datos alfanuméricos se almacenan utilizando el código ASCII (Código estándar estadounidense para el intercambio de información).

OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

1. Conversar utilizando la terminología computacional apropiada, como bit, byte, datos, sistema de memoria real, sistema de memoria en modo protegido, Windows, DOS, E/S, etcétera.
2. Detallar brevemente la historia de la computadora y listar las aplicaciones realizadas por los sistemas computacionales.
3. Proporcionar las generalidades de los diversos miembros de la familia 80X86 y Pentium.
4. Dibujar el diagrama de bloques de un sistema computacional y explicar el propósito de cada bloque.
5. Describir la función del microprocesador y detallar su operación básica.
6. Definir el contenido del sistema de memoria en la computadora personal.
7. Realizar conversiones de números binarios, decimales y hexadecimales.
8. Diferenciar y representar la información numérica y alfanumérica como datos enteros, de punto flotante, BCD y ASCII.

1-1**ANTECEDENTES HISTÓRICOS**

En esta primera sección describiremos los eventos históricos que produjeron el desarrollo del microprocesador y, específicamente, los extremadamente poderosos y actuales microprocesadores 80X86,¹ Pentium, Pentium Pro, Pentium III y Pentium 4.² Aunque no es esencial un estudio de la historia para comprender el funcionamiento del microprocesador, sirve como lectura de interés y proporciona una perspectiva histórica de la rápida evolución de la computadora.

La era mecánica

La idea de un sistema computacional no es nueva; ya existía mucho antes de que se desarrollaran los dispositivos eléctricos y electrónicos modernos. La idea de realizar cálculos con una máquina se remonta hasta el año 500 a.C., cuando los babilonios inventaron el **ábaco**, la primera calculadora mecánica. El ábaco tiene hileras de cuentas con las que se realizan los cálculos. Los sacerdotes de Babilonia lo utilizaban para llevar la cuenta de sus vastos almacenes de grano. El ábaco, que aún se utiliza en la actualidad, se mejoró hasta 1642 cuando el matemático Blaise Pascal inventó una calculadora construida con base en engranajes y ruedas. Cada engranaje contenía 10 dientes que, cuando se movían una vuelta completa, hacían que un segundo engranaje avanzara una posición. Éste es el mismo principio que se utiliza en el mecanismo del odómetro de un automóvil y es la base de todas las calculadoras mecánicas. De hecho, el lenguaje de programación PASCAL se nombra así en honor de Blaise Pascal, por su trabajo pionero en las matemáticas y con la calculadora mecánica.

A principios de la década de 1800 llegaron las primeras máquinas mecánicas con engranajes prácticos, utilizadas para calcular información de manera automática. Esto fue antes de que los humanos inventaran la bombilla eléctrica o antes de que se conociera mucho sobre la electricidad. En este amanecer de la era computacional, se soñaba con equipos mecánicos que pudieran calcular hechos numéricos mediante un programa; y no simplemente calcular hechos, como lo hace una calculadora.

En 1937 unos planos y diarios mostraron que uno de los primeros pioneros de la maquinaria computacional mecánica fue Charles Babbage, ayudado por Augusta Ada Byron, la Condesa de Lovelace. Babbage fue comisionado en 1823 por la Real Sociedad Astronómica de la Gran Bretaña para producir una máquina calculadora programable. Esta máquina debía generar tablas de navegación para la Armada Real Británica. Él aceptó el reto y empezó a crear lo que llamó su **Máquina analítica**. Este motor era una computadora mecánica operada por vapor que almacenaba 1000 números decimales de veinte dígitos y un programa variable que podía modificar la función de la máquina para realizar varias tareas de cálculo. La entrada a su máquina era a través de tarjetas perforadas, muy parecidas a las tarjetas perforadas que usaban las computadoras en las décadas de 1950 y 1960. Se supone que él obtuvo la idea de usar tarjetas perforadas gracias a Joseph Jaquard, un francés que utilizaba tarjetas perforadas como base para un telar que inventó en 1801, al que todavía se le conoce como *telar de Jacquard*. Este telar utilizaba tarjetas perforadas para crear complicados patrones de tejido.

Después de muchos años de trabajo, el sueño de Babbage empezó a desvanecerse cuando se dio cuenta que los maquinistas de su época no podían crear las piezas mecánicas que necesitaban para completar su trabajo. La máquina analítica requería más de 50,000 piezas maquinadas, las cuales no podían fabricarse con la precisión suficiente como para que su motor funcionara de manera confiable.

La era eléctrica

La década de 1800 fue testigo de la llegada del motor eléctrico (concebido por Michael Faraday) con lo cual surgió una multitud de máquinas de sumar controladas por motores, todas basadas en la calculadora mecánica desarrollada por Blaise Pascal. Estas calculadoras mecánicas controladas en forma eléctrica no fueron piezas comunes del equipo de oficina sino hasta principios de la década de 1970, cuando apareció la pequeña calculadora electrónica portátil, introducida por primera vez por Bomar Corporation y

¹80X86 es un acrónimo aceptado para los microprocesadores 8086, 8088, 80186, 80188, 80286, 80386 y 80486; lo cual también incluye la serie Pentium.

²Pentium, Pentium Pro, Pentium II, Pentium III y Pentium IV son marcas registradas de Intel Corporation.

llamada el **Cerebro Bomar**. Monroe fue también un pionero líder de las calculadoras electrónicas, pero sus máquinas eran modelos de escritorio de 4 funciones, del tamaño de cajas registradoras.

En 1889 Herman Hollerith desarrolló la tarjeta perforada para almacenar datos. Al igual que Babbage, aparentemente él también tomó prestada la idea de una tarjeta perforada de Jacquard. También desarrolló un equipo mecánico (controlado por uno de los nuevos motores eléctricos) que contaba, ordenaba y cotejaba información almacenada en tarjetas perforadas. La idea de calcular mediante maquinaria intrigó tanto al gobierno de los EE.UU. que Hollerith fue comisionado para utilizar su sistema de tarjetas perforadas para almacenar y tabular información para el censo de 1890.

En 1896 Hollerith formó la empresa Tabulating Machine Company, la cual desarrolló una línea de equipos que utilizaban tarjetas perforadas para la tabulación. Después de varias fusiones esta empresa se convirtió en International Business Machines Corporation, la cual se conoce actualmente como IBM, Inc, y a las tarjetas perforadas utilizadas en los sistemas computacionales se les llamó **tarjetas de Hollerith**, en honor de Herman Hollerith. El código de 12 bits que utilizan esas tarjetas perforadas se conoce como **código de Hollerith**.

Los equipos mecánicos controlados por motores eléctricos continuaron dominando el mundo del procesamiento de información hasta la construcción de la primera máquina calculadora electrónica en 1941. El inventor alemán Konrad Zuse, quien trabajaba como ingeniero para la empresa Henschel Aircraft Company en Berlín, fue quien inventó la primera computadora moderna. En 1936 Zuse construyó una versión mecánica de su sistema y posteriormente en 1939 construyó su primer sistema computacional electromecánico, llamado Z2. Su computadora calculadora Z3, como se muestra en la figura 1-1, probablemente se utilizó en el diseño de aeronaves y misiles durante la Segunda Guerra Mundial para el ejército alemán. El Z3 era un equipo de lógica de relevadores que tenía un reloj de 5.33 Hz (mucho más lento que los recientes microprocesadores de varios GHz). Si el gobierno alemán le hubiera dado a Zuse un patrocinio adecuado, es muy probable que hubiera desarrollado un sistema computacional mucho más poderoso. Zuse está por fin siendo honrado por su trabajo pionero en el área de la electrónica digital y por su sistema computacional Z3.

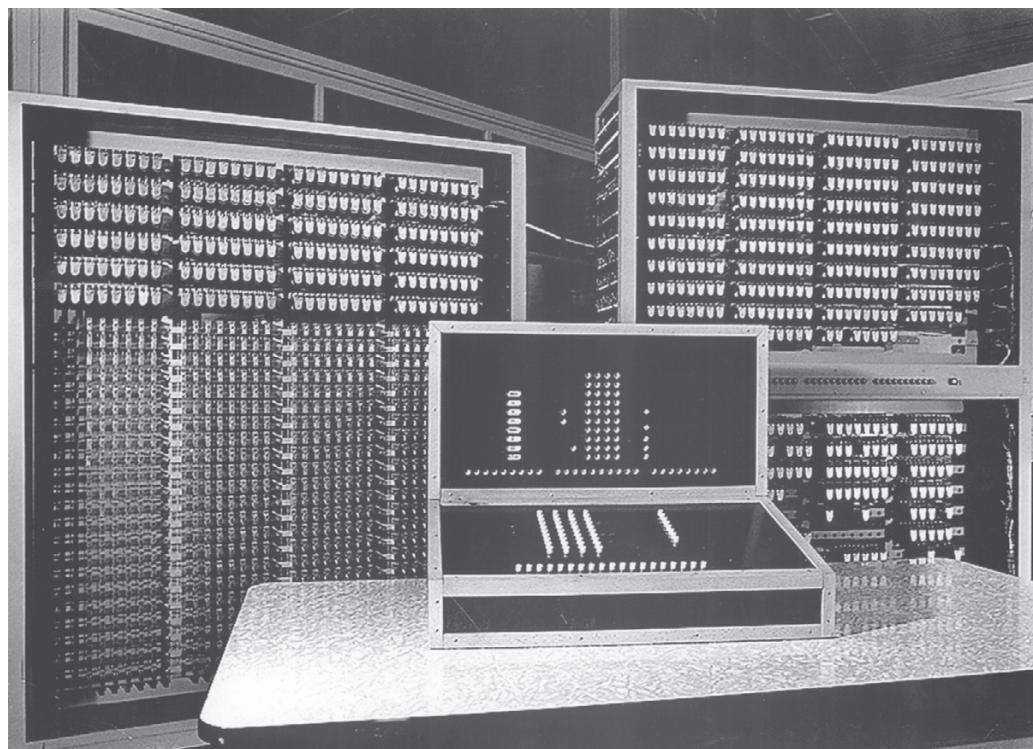


FIGURA 1-1 La computadora Z3 desarrollada por Konrad Zuse utiliza una frecuencia de reloj de 5.33 Hertz. (Fotografía cortesía de Horst Zuse, el hijo de Konrad.)

Recientemente se descubrió (por medio de la desclasificación de documentos militares de los ingleses) que la primera computadora electrónica se puso en operación en 1943 para quebrantar los códigos militares secretos de los alemanes. Este primer sistema computacional electrónico, que utilizaba tubos al vacío, fue inventado por Alan Turing, quien llamó a su máquina **Colossus**, probablemente por su tamaño. Un problema con Colossus fue que, aunque su diseño le permitía quebrantar códigos militares secretos de los alemanes, generados por la **máquina Enigma** mecánica, no podía resolver otros problemas. Colossus no era programable; era un sistema computacional de programa fijo, lo que se conoce actualmente como una **computadora de propósito especial**.

El primer sistema computacional electrónico programable de propósito general se desarrolló en 1946, en la Universidad de Pensilvania. Esta computadora moderna se llamó **ENIAC (Calculadora e Integradora Numérica Electrónica)**. La ENIAC era una máquina enorme que contenía más de 17,000 tubos al vacío y más de 500 millas de cables. Esta máquina pesaba más de 30 toneladas y realizaba sólo aproximadamente 100,000 operaciones por segundo. La ENIAC llevó al mundo a la era de las computadoras electrónicas. Se programaba cambiando el cableado de sus circuitos; un proceso que tomaba a muchos trabajadores varios días en completarlo. Los trabajadores cambiaban las conexiones eléctricas en tableros de conexiones que tenían una apariencia como la de los comutadores telefónicos. Otro problema con la ENIAC era la vida de los componentes de tubos al vacío, que requerían de un mantenimiento frecuente.

Los descubrimientos posteriores fueron el desarrollo del transistor, en diciembre 23 de 1947, en los laboratorios Bell por John Bardeen, William Shockley y Walter Brattain. Después le siguió, en 1958, la invención del circuito integrado por Jack Kilby de Texas Instruments. El circuito integrado condujo al desarrollo de los circuitos integrados digitales (RTL, o lógica de resistencia a transistor) en la década de 1960 y al primer microprocesador en Intel Corporation en 1971. En ese tiempo, los ingenieros de Intel Federico Faggin, Ted Hoff y Stan Mazor desarrollaron el microprocesador 4004 (Patente de los E.U. 3,821,715); el dispositivo que empezó la revolución de los microprocesadores, el cual continúa actualmente a un ritmo cada vez más acelerado.

Avances en la programación

Una vez desarrolladas las máquinas programables, empezaron a aparecer los programas y los lenguajes de programación. Como se mencionó anteriormente, el primer sistema computacional electrónico programable se programaba volviendo a cablear sus circuitos. Como esto era demasiado laborioso para una aplicación práctica, en la primera etapa de la evolución de los sistemas computacionales comenzaron a aparecer los lenguajes computacionales para controlar la computadora. El primer lenguaje de ese tipo, el **lenguaje máquina**, estaba compuesto de unos y ceros y utilizaba códigos binarios almacenados en el sistema de memoria computacional como grupos de instrucciones, a lo cual se le llamaba programa. Esto era más eficiente que volver a cablear una máquina para programarla, pero aún requería de mucho tiempo el desarrollar un programa debido al número total de códigos requeridos. El matemático John von Neumann fue la primera persona en desarrollar un sistema que aceptaba instrucciones y las almacenaba en memoria. A menudo a las computadoras se les llama **máquinas de von Neumann** en honor de John von Neumann. (Recuerde que Babbage también había desarrollado el concepto mucho antes que von Neumann.)

Una vez que estuvieron disponibles los sistemas computacionales tales como el UNIVAC, a principios de la década de 1950, se utilizó el **lenguaje ensamblador** para simplificar la tarea de introducir código binario en una computadora como sus instrucciones. El ensamblador permite al programador utilizar códigos nemáticos en inglés, como ADD para la suma, en lugar de un número binario tal como 0100 0111. Aunque el lenguaje ensamblador era una ayuda para la programación, no fue sino hasta 1957 cuando Grace Hopper desarrolló el primer lenguaje de programación de alto nivel llamado **FLOW-MATIC**, que las computadoras se hicieron más fáciles de programar. En el mismo año, IBM desarrolló el lenguaje FORTRAN (**FORmula TRANslator –Traductor de fórmulas**) para sus sistemas computacionales. Este lenguaje permitía a los programadores desarrollar programas que utilizaran fórmulas para resolver problemas matemáticos. FORTRAN es utilizado aún por algunos científicos para la programación de computadoras. Otro lenguaje similar, introducido un año después de FORTRAN, fue ALGOL (**ALGOrithmic Language –Lenguaje algorítmico**).

El primer lenguaje de programación verdaderamente exitoso y de amplio uso para las aplicaciones comerciales fue COBOL (**C**OMputer **B**usiness **O**riented **L**anguage –**L**enguaje computacional orientado a los negocios). Aunque el uso del COBOL ha diminuido de manera considerable en la actualidad, aún juega un papel importante en muchos sistemas comerciales extensos. Otro lenguaje de negocios que una vez fue popular es RPG (**R**eport **P**rogram **G**enerator –**G**enerador de programas de reportes), el cual permite la programación especificando la forma de la entrada, la salida y los cálculos.

Desde los primeros días de programación han aparecido varios lenguajes adicionales. Algunos de los más comunes son BASIC, Java, C#, C/C++, PASCAL y ADA. Los lenguajes BASIC y PASCAL se diseñaron como lenguajes de enseñanza, pero han escapado del salón de clases y se utilizan en muchos sistemas computacionales. El lenguaje BASIC es probablemente el más fácil de aprender. Algunas estimaciones indican que el lenguaje BASIC se utiliza en la computadora personal en el 80% de los programas escritos por los usuarios. En la década pasada una nueva versión de BASIC, llamada Visual BASIC, ha facilitado la programación en el entorno Windows. El lenguaje Visual BASIC puede eventualmente suplantar a C/C++ y PASCAL como un lenguaje científico, pero eso es dudoso.

En la comunidad científica, C/C++ y en ocasiones PASCAL y FORTRAN aparecen como programas de control. Estos lenguajes, en especial C/C++, permiten al programador un control casi completo sobre el entorno de programación y el sistema computacional. En muchos casos, C/C++ está sustituyendo a varios de los controladores o del software de control de máquina de bajo nivel, reservados normalmente para el lenguaje ensamblador. Aún así, el lenguaje ensamblador sigue jugando un papel importante en la programación. La mayoría de los videojuegos para la computadora personal están escritos casi de manera exclusiva en lenguaje ensamblador. Este lenguaje también se entremezcla con C/C++ y PASCAL para realizar las funciones de control de la máquina eficientemente. Algunas de las instrucciones en paralelo más recientes (SIMD) que se encuentran en los microprocesadores Pentium son sólo programables en lenguaje ensamblador.

El lenguaje ADA es muy utilizado por el Departamento de defensa de los Estados Unidos. Este lenguaje fue nombrado en honor de Augusta Ada Byron, Condesa de Lovelace. La condesa trabajaba con Charles Babbage a principios de la década de 1800 en el desarrollo de software para su Máquina Analítica.

La era del microprocesador

El primer microprocesador en el mundo, el Intel 4004, era de 4 bits; un controlador programable en un chip. Podía direccionar tan solo 4096 posiciones de memoria de 4 bits. (Un **bit** es un dígito binario con un valor de uno o cero. A una posición de memoria de 4 bits se le llama comúnmente **cuarteto**.) El conjunto de instrucciones del 4004 contenía solamente 45 instrucciones. Estaba fabricado con la tecnología MOSFET de canal P (que en ese entonces era la más avanzada) que sólo le permitía ejecutar instrucciones a una lenta velocidad de 50 KIPs (**kilo instrucciones por segundo**). Esto era lento si se le comparaba con las 100,000 instrucciones ejecutadas por segundo por la computadora ENIAC de 30 toneladas en 1946. La principal diferencia era que el 4004 pesaba menos de una onza.

Al principio, las aplicaciones abundaban para este dispositivo. El microprocesador de 4 bits fue usado en los primeros sistemas de videojuegos y en los pequeños sistemas de control basados en microprocesador. Uno de los primeros videojuegos, el videojuego shuffleboard, fue producido por Bailey. Los principales problemas con este microprocesador eran su velocidad, la anchura de las palabras y el tamaño de la memoria. La mejora del microprocesador de 4 bits terminó cuando Intel sacó al mercado el microprocesador 4040, una versión actualizada del 4004. El 4040 operaba a una mayor velocidad, aunque carecía de mejoras en la anchura de las palabras y en el tamaño de la memoria. Otras compañías, en especial Texas Instruments (TMS-1000), también produjeron microprocesadores de 4 bits. Este tipo de microprocesador aún sobrevive en aplicaciones de bajo nivel tales como hornos de microondas y pequeños sistemas de control, y aún está disponible a través de algunos fabricantes de microprocesadores. La mayoría de las calculadoras se basan todavía en microprocesadores de 4 bits que procesan códigos BCD (**D**ecimal **C**odificado **en** **B**inario) de 4 bits.

A finales de 1971 y teniendo en cuenta que el microprocesador era un producto comercialmente viable, Intel Corporation sacó al mercado el 8008; una versión extendida de 8 bits del microprocesador 4004. El 8008 direccionaba un tamaño de memoria expandida (16 Kbytes) y contenía instrucciones

TABLA 1-1 Los primeros microprocesadores de 8 bits.

Fabricante	Número de pieza
Fairchild	F-8
Intel	8080
MOS Technology	6502
Motorola	MC6800
National Semiconductor	IMP-8
Rockwell International	PPS-8
Zilog	Z-8

adicionales (un total de 48) que brindaban la oportunidad para su aplicación en sistemas más avanzados. (Un **byte** es por lo general un número binario de 8 bits, y un **K** equivale a 1024. A menudo, el tamaño de la memoria se especifica en **Kbytes**.)

A medida que los ingenieros fueron desarrollando usos más demandantes para el microprocesador 8008, descubrieron que su tamaño de memoria relativamente pequeño, su baja velocidad y su conjunto de instrucciones limitaban su utilidad. Intel reconoció estas limitaciones y presentó el microprocesador 8080 en 1973; el primero de los microprocesadores modernos de 8 bits. Aproximadamente seis meses después de que Intel lanzó el microprocesador 8080, Motorola Corporation introdujo su microprocesador MC6800. Las compuertas de la presa se abrieron y el 8080 (junto con el MC6800, aunque en menor grado) pasó a la era del microprocesador. En poco tiempo, otras compañías empezaron a introducir sus propias versiones del microprocesador de 8 bits. La tabla 1-1 muestra varios de los primeros microprocesadores y sus fabricantes. De todos esos productores, sólo Intel y Motorola (IBM también produce microprocesadores estilo Motorola) continúan creando exitosamente versiones más nuevas y mejoradas del microprocesador. Zilog aún fabrica microprocesadores, pero permanece en segundo plano, concentrado en microcontroladores y controladores integrados en vez de microprocesadores de propósito general. Rockwell casi ha abandonado el desarrollo de microprocesadores a favor de los circuitos modernos. Motorola ha disminuido su participación en el mercado, de casi un 50% a un porcentaje más bajo.

¿Qué tenía de especial el 8080? Este microprocesador no sólo podía direccionar más memoria y ejecutar instrucciones adicionales, sino que las ejecutaba 10 veces más rápido que el 8008. Resumiendo, un sistema basado en el 8008 tardaba 20 μ s (50,000 instrucciones por segundo) y un sistema basado en el 8080 requería de sólo 2.0 μ s (500,000 instrucciones por segundo). Además, el 8080 era compatible con tecnología TTL (lógica transistor-transistor), mientras que el 8008 no era directamente compatible. Esto hizo que la interfaz fuera más sencilla y menos costosa. El 8080 también direccionaba cuatro veces más memoria (64 Kbytes) que el 8008 (16 Kbytes). Estas mejoras anunciaron la era del 8080 y la saga del microprocesador continua. De hecho, la primera computadora personal, la MITS Altair 8800, salió al mercado en 1974. (Tome en cuenta que probablemente se haya elegido el número 8800 para evitar violaciones de derechos de autor de Intel.) Bill Gates y Paul Allen, los fundadores de Microsoft Corporation, desarrollaron el intérprete de lenguaje BASIC escrito para la computadora Altair 8800 en 1975. Digital Research escribió el programa ensamblador para la Altair 8800, empresa que una vez produjo el DR-DOS para la computadora personal.

El microprocesador 8085. En 1977 Intel Corporation presentó una versión actualizada del 8080: el 8085. Éste fue el último microprocesador de propósito general de 8 bits desarrollado por Intel. Aunque era sólo un poco más avanzado que un microprocesador 8080, el 8085 ejecutaba software a una velocidad aún mayor. Resaltando, el 8080 tardaba 2.0 μ s (500,000 instrucciones por segundo en el 8080) y el 8085 requería solamente de 1.3 μ s (769,230 instrucciones por segundo). Las principales ventajas del 8085 eran su generador de reloj interno, su controlador de sistema interno y una frecuencia de reloj más alta. Este mayor nivel de integración de los componentes redujo el costo del 8085 y aumentó su utilidad. Intel ha logrado vender más de 100 millones de copias del microprocesador 8085, su microprocesador de propósito general de 8 bits más exitoso. Como el 8085 también lo fabrican (de segunda fuente) muchas otras compañías, hay más de 200 millones de estos microprocesadores en existencia. Es muy probable que las aplicaciones que contienen el 8085 sigan siendo populares. Otra compañía que vendió 500 millones de microprocesadores de 8 bits es Zilog Corporation, la cual produjo el microprocesador

Z-80. El lenguaje máquina de este procesador es compatible con el del 8085, lo cual significa que hay más de 700 millones de microprocesadores que ejecutan código compatible con el 8085/Z-80.

El microprocesador moderno

En 1978, Intel sacó al mercado el microprocesador 8086; aproximadamente un año después sacó al mercado el 8088. Ambos dispositivos eran microprocesadores de 16 bits, los cuales ejecutaban instrucciones en tiempos de hasta 400 ns (2.5 MIPs, o **2.5 millones de instrucciones por segundo**). Esto representó una considerable mejora en cuanto a la velocidad de ejecución del 8085. Además, el 8086 y el 8088direccionaban 1 Mbyte de memoria, que era 16 veces más memoria que la direccionada por el 8085. (Una **memoria de 1 Mbyte** contiene posiciones de memoria de un tamaño de 1024 Kbytes, o 1,048,576 bytes.) La mayor velocidad de ejecución y el tamaño más grande de memoria permitieron que el 8086 y el 8088 sustituyeran a las minicomputadoras más pequeñas en muchas aplicaciones. Otra de las características incluidas en el 8086/8088 era una caché o cola de instrucciones de 4 o 6 bytes que obtenían previamente unas cuantas instrucciones antes de ejecutarlas. La cola agilizó la operación de muchas secuencias de instrucciones y se consolidó como base para las cachés de instrucciones mucho mayores que se encuentran en los microprocesadores modernos.

El incremento de memoria y las instrucciones adicionales en los microprocesadores 8086 y 8088 impulsaron el desarrollo de muchas aplicaciones sofisticadas para los microprocesadores. Entre las mejoras al conjunto de instrucciones se incluían las instrucciones de multiplicar y dividir, que no existían en los primeros microprocesadores. Además, el número de instrucciones aumentó de 45 en el 4004, a 246 en el 8085, a más de 20,000 variaciones en los microprocesadores 8086 y 8088. Hay que tener en cuenta que a estos microprocesadores se les llama **CISC (computadoras de conjunto de instrucciones complejas)** debido al número y la complejidad de las instrucciones. Las instrucciones adicionales facilitaron la tarea de desarrollar aplicaciones eficientes y sofisticadas, aún y cuando el número de instrucciones es al principio abrumador y se requería una cantidad considerable de tiempo para aprenderlas. El microprocesador de 16 bits también proporcionaba un mayor espacio de almacenamiento en los registros internos que el microprocesador de 8 bits. Los registros adicionales permitían la escritura más eficiente de software.

El microprocesador de 16 bits mejoró principalmente debido a la necesidad de sistemas de memoria más extensas. La popularidad de la familia Intel se aseguró en 1981, cuando IBM Corporation decidió utilizar el microprocesador 8088 en su computadora personal. Las aplicaciones tales como las hojas de cálculo, los procesadores de palabras, los correctores ortográficos y los tesauros basados en computadora requerían mucha memoria, de hecho más de los 64 Kbytes que se encuentran en los microprocesadores de 8 bits, para ejecutarse eficientemente. Los microprocesadores 8086 y 8088 de 16 bits contaban con 1 Mbyte de memoria para estas aplicaciones. En poco tiempo, hasta el sistema de memoria de 1 Mbyte resultó ser limitante para las bases de datos extensas y otras aplicaciones. Esto llevó a Intel a presentar el microprocesador 80286 (un 8086 actualizado) en 1983.

El microprocesador 80286. Este microprocesador (que también tenía una arquitectura de 16 bits) era casi idéntico al 8086 y al 8088, con la excepción de que direccionaba un sistema de memoria de 16 Mbytes en vez de 1 Mbyte, y contaba con unas cuantas instrucciones adicionales que administraban los 15 Mbytes de memoria adicional. Se incrementó la velocidad del reloj del 80286 para que ejecutara algunas instrucciones en tan sólo 250 ns (4.0 MIPs) con la versión original de 8.0 MHz. También hubo algunos cambios en cuanto a la ejecución interna de las instrucciones, lo cual produjo un aumento de ocho veces más la velocidad para muchas instrucciones, en comparación con las instrucciones del 8086/8088.

El microprocesador de 32 bits. Las aplicaciones empezaron a demandar velocidades mayores para los microprocesadores, y rutas de datos más amplias. Esto provocó que en 1986 llegara el 80386 fabricado por Intel Corporation. Este microprocesador representó un importante adelanto en la arquitectura de 16 bits de los microprocesadores 8086-80286. El 80386 fue el primer microprocesador de 32 bits práctico de Intel que contenía un bus de datos de 32 bits y una dirección de memoria de 32 bits. (Hay que considerar que Intel produjo antes un microprocesador de 32 bits que no tuvo éxito, llamado iapx-432.) Mediante estos buses de 32 bits, el 80386 direccionaba hasta 4 Gbytes de memoria. (**1 G** de memoria contiene 1024 M, o 1,073,741,824 posiciones.) Una memoria de 4 Gbytes puede almacenar la sorprendente cantidad

de 1,000,000 páginas de datos de texto ASCII, mecanografiadas a doble espacio. El 80386 estaba disponible en unas cuantas versiones modificadas, como el 80386SX, que direccionaba 16 Mbytes de memoria a través de un bus de datos de 16 bits y un bus de direcciones de 24 bits, y el 80386SL/80386SLC, que direccionaba 32 Mbytes de memoria a través de un bus de datos de 16 bits y un bus de datos de 25 bits. Una versión de nombre 80386SLC contenía una memoria caché interna que le permitía procesar datos a velocidades aún mayores. En 1995, Intel sacó al mercado el microprocesador 80386EX. A este microprocesador se le llama **PC integrada** porque contiene todos los componentes de la computadora personal de clase AT en un solo circuito integrado. El 80386EX también contiene 24 líneas para entrada/salida de datos, un bus de direcciones de 26 bits, un bus de datos de 16 bits, un controlador de actualización de DRAM y una lógica de selección de chip programable.

Las aplicaciones que requieren de mayores velocidades de microprocesador y sistemas de memoria más grandes incluyen sistemas de software que utilizan una GUI, o **interfaz gráfica de usuario**. Las pantallas gráficas modernas a menudo contienen 256,000 o más elementos de imagen (**pixels**, o **pels**). La pantalla de vídeo VGA (**arreglo de gráficos de vídeo**) menos sofisticada tiene una resolución de 640 pixeles por línea de exploración con 480 líneas de exploración (ésta es la resolución que se utiliza cuando la computadora se inicia y muestra la pantalla de arranque). Para mostrar una pantalla de información se debe cambiar cada uno de los elementos de imagen, para lo cual se requiere un microprocesador de alta velocidad. Casi todos los nuevos paquetes de software utilizan este tipo de interfaz de vídeo. Estos paquetes basados en GUI requieren de microprocesadores de alta velocidad y de adaptadores de vídeo acelerado para una rápida y eficiente manipulación del texto de vídeo y los datos de los gráficos. El sistema más impresionante que requiere una computadora de alta velocidad para su interfaz de pantalla de gráficos es Windows de Microsoft Corporation.³ A menudo a la GUI se le llama una pantalla WYSIWYG (**lo que ve es lo que obtiene**).

El microprocesador de 32 bits es necesario debido al tamaño de su bus de datos, el cual transfiere números reales (de punto flotante con precisión simple) que requieren una memoria de 32 bits de ancho. Para poder procesar con eficiencia los números de 32 bits, el microprocesador debe pasarlos eficientemente entre él y la memoria. Si los números pasan a través de un bus de datos de 8 bits, se requieren cuatro ciclos de lectura o escritura; no obstante, si se pasan a través de un bus de datos de 32 bits sólo se requiere un ciclo de lectura o escritura. Esto aumenta considerablemente la velocidad de cualquier programa que manipule números reales. La mayoría de los lenguajes de alto nivel, las hojas de cálculo y los sistemas de administración de bases de datos utilizan números reales para el almacenamiento de los datos. Los números reales también se utilizan en paquetes de diseño gráfico que utilizan vectores para trazar imágenes en la pantalla de vídeo. Entre ellos se incluyen sistemas CAD (**dibujo/diseño asistido por computadora**) tales como AUTOCAD, ORCAD, etcétera.

El microprocesador 80386 además de proporcionar velocidades de reloj más altas, incluía una unidad de administración de memoria que permitía al sistema operativo asignar y administrar los recursos de memoria. Los primeros microprocesadores dejaban la administración de la memoria completamente al software. El 80386 incluía circuitos de hardware para administrar y asignar la memoria, lo cual mejoraba su eficiencia y reducía la sobrecarga impuesta por el software.

El conjunto de instrucciones del microprocesador 80386 era compatible hacia arriba con los primeros microprocesadores 8086, 8088 y 80286. Las instrucciones adicionales hacían referencia a los registros de 32 bits y administraban el sistema de memoria. Las instrucciones de administración de memoria y las técnicas utilizadas por el microprocesador 80286 también son compatibles con el 80386. Estas características permitían que el software antiguo de 16 bits operara con el microprocesador 80386.

El microprocesador 80486. En 1989 Intel sacó al mercado el microprocesador 80486, que incorporaba un microprocesador parecido al 80386, un coprocesador numérico similar al 80387 y un sistema de memoria de caché de 8 Kbytes en un paquete integrado. Aunque el microprocesador 80486 no era radicalmente distinto del 80386, incluía un cambio substancial. La estructura interna del 80486 se había modificado en base al 80386, de tal forma que la mitad de sus instrucciones se ejecutaran en un ciclo de reloj en vez de dos. Como el 80486 estaba disponible en una versión de 50 Mhz, casi la mitad de

³Windows es marca registrada de Microsoft Corporation y está disponible actualmente como Windows 98, Windows 2000, Windows ME y Windows XP.

las instrucciones se ejecutaban en 25ns (50 MIPS). La mejora promedio de velocidad para una mezcla común de instrucciones era de un 50% más que un 80386 que funcionara a la misma velocidad de reloj. Las versiones posteriores del 80486 ejecutaban instrucciones a velocidades aún mayores con una versión (80486DX2) cuya velocidad de reloj estaba aumentada al doble (66 MHz). Esta versión ejecutaba instrucciones a la velocidad de 66 MHz, y las transferencias de memoria se ejecutaban a la velocidad de 33 MHz. (Ésta es la razón de por qué se le llamaba microprocesador con doble velocidad de reloj [double-clocked].) El 80486DX4, una versión de Intel con triple velocidad de reloj, mejoró la velocidad de ejecución interna a 100 MHz, con transferencias de memoria a 33 MHz. Este procesador ejecutaba instrucciones aproximadamente a la misma velocidad que el Pentium de 60 MHz. También contenía una caché expandida de 16 Kbytes en lugar de la caché estándar de 8 Kbytes que tenían los primeros microprocesadores 80486. Advanced Micro Devices (AMD) ha producido una versión con triple velocidad de reloj, que opera con una velocidad de bus de 40 MHz y una velocidad de reloj de 120 MHz. El futuro promete brindarnos microprocesadores que ejecuten internamente instrucciones a velocidades de hasta 10 Ghz o mayores.

Hubo otras versiones del 80486, a las cuales se les llamó procesadores OverDrive⁴. Este tipo de procesador era en realidad una versión del 80486DX con doble velocidad de reloj, que sustituía a un 80486SX o a un 80486DX de menor velocidad. Cuando el procesador OverDrive se conectaba en su zócalo, deshabilitaba o sustituía el 80486SX o el 80486DX y funcionaba como una versión del microprocesador con doble velocidad de reloj. Por ejemplo, si un 80486SX que operaba a 25 MHz se sustituía con un procesador OverDrive, funcionaba como un microprocesador 80486DX2 de 50 MHz, con una velocidad de transferencia de memoria de 25 MHz.

En la tabla 1-2 se muestran muchos microprocesadores producidos por Intel y Motorola, con información sobre sus tamaños de palabras y de memoria. Otras compañías produjeron microprocesadores, pero ninguna ha obtenido el éxito de Intel y, en un menor grado, de Motorola.

El microprocesador Pentium. El Pentium se introdujo en 1993 y era similar a los microprocesadores 80386 y 80486. Este microprocesador se llamaba originalmente P5 o 80586, pero Intel decidió no utilizar un número ya que parecía imposible proteger un número con derechos de autor. Las dos versiones introductorias del Pentium operaban con una frecuencia de reloj de 60 MHz y 66 MHz, y con una velocidad de 110 MIPS, además de las versiones con una frecuencia más alta de 100 MHz (una velocidad de reloj multiplicada por uno y medio veces la velocidad original) que operaban a 150 MIPS. También estaba disponible el Pentium con doble velocidad de reloj, que operaba a 120 MHz y 133 MHz, así como versiones de mayor velocidad. (La versión más veloz producida por Intel es el Pentium de 233 MHz, que es una versión con una velocidad de reloj multiplicada tres y medio veces.) Otra diferencia era que el tamaño de la caché se había aumentado a 16 Kbytes, en comparación con la caché de 8 K que tenía la versión básica del 80486. El Pentium contenía una caché de instrucciones de 8 Kbytes y una caché de datos de 8 Kbytes, lo cual permitía que un programa que transfería una gran cantidad de datos de memoria pudiera beneficiarse de una caché. El sistema de memoria contenía hasta 4 Gbytes, y la anchura del bus de datos se incrementó de los 32 bits que tenían los microprocesadores 80386 y 80486, a 64 bits. La velocidad de transferencia del bus de datos era de 60 MHz o de 66 MHz, dependiendo de la versión del Pentium. (Recuerde que la velocidad de bus del 80486 era de 33 MHz.) Este bus de datos más ancho aceptaba números de punto flotante de doble precisión, utilizados para la visualización de gráficos modernos de alta velocidad, generados por vectores. Estas velocidades de bus más altas deberían permitir que el software y el vídeo de realidad virtual operaran a velocidades más realistas en las plataformas actuales y futuras basadas en el Pentium. El bus de datos ampliado y la velocidad de ejecución mayor del Pentium permitían que las pantallas de vídeo de cuadro completo operaran a velocidades de exploración de 30 Hz o mayores: comparables con la televisión comercial. Las versiones recientes del Pentium también incluían instrucciones adicionales, llamadas extensiones multimedia, o instrucciones MMX. Aunque Intel esperaba que las instrucciones MMX se utilizaran ampliamente, parece ser que pocas compañías de software las han utilizado. La principal razón es que no hay un soporte de lenguaje de alto nivel para estas instrucciones.

Intel también había sacado al mercado el tan esperado Pentium OverDrive (P24T) para los sistemas 80486 antiguos que operaban con un reloj de 63 MHz o de 83 MHz. La versión de 63 MHz es la

⁴OverDrive es una marca registrada de Intel Corporation.

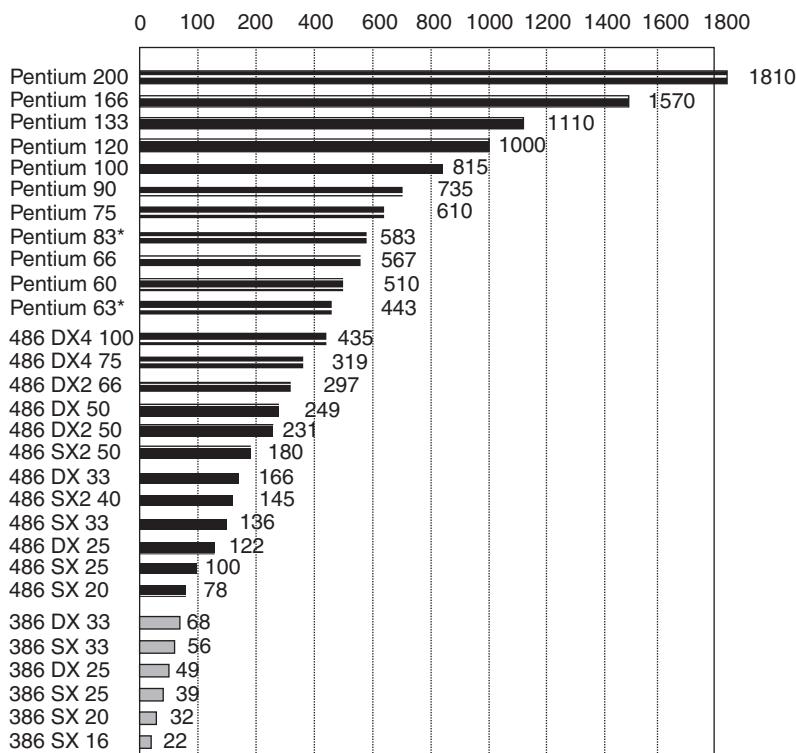
TABLA 1-2 Varios procesadores Intel y Motorola modernos.

<i>Fabricante</i>	<i>Número de pieza</i>	<i>Anchura del bus de datos</i>	<i>Tamaño de memoria</i>
Intel	8048	8	2 K interna
	8051	8	8 K interna
	8085A	8	64K
	8086	16	1 M
	8088	8	1 M
	8096	16	8 K interna
	80186	16	1 M
	80188	8	1 M
	80251	8	16 K interna
	80286	16	16 M
	80386EX	16	64 M
	80386DX	32	4 G
	80386SL	16	32 M
	80386SLC	16	32 M + caché de 8 K
	80386SX	16	16 M
	80486DX/DX2	32	4 G + caché de 8 K
	80486SX	32	4 G + caché de 8 K
	80486DX4	32	4 G + caché de 16 K
Motorola	Pentium	64	4 G + caché de 16 K
	Pentium OverDrive	32	4 G + caché de 16 K
	Pentium Pro	64	64 G + caché L1 de 16 K + caché L2 de 256K
	Pentium II	64	64 G + caché L1 de 32 K + caché L2 de 256 K
	Pentium III	64	64 G + caché L1 de 32 K + caché L2 de 256 K
	Pentium 4	64	64G + caché L1 de 8 K + caché L2 de 512 K (o mayor)
	6800	8	64 K
Motorola	6805	8	2 K
	6809	8	64 K
	68000	16	16 M
	68008D	8	4 M
	68008Q	8	1 M
	68010	16	16 M
	68020	32	4 G
	68030	32	4 G + caché de 256 K
	68040	32	4 G + caché de 8 K
	68050	32	Propuesto, pero nunca salió al mercado
	68060	64	4 G + caché de 16 K
	PowerPC	64	4 G + caché de 32 K

actualización de los sistemas 80486DX2 de 50 Mhz más antiguos; la versión de 83 Mhz es la actualización de los sistemas 80486DX2 de 66 Mhz. El sistema actualizado de 83 Mhz funciona a una velocidad similar al rango entre un Pentium de 66 Mhz y un Pentium de 75 Mhz. Si los controladores antiguos de vídeo de bus local VESA y de caché de disco parecen demasiado costosos como para desecharlos, el Pentium OverDrive representa una ruta ideal de actualización del 80486 al Pentium.

Probablemente la característica más ingeniosa del Pentium sean sus procesadores de enteros duales. El Pentium ejecuta dos instrucciones (que no dependan una de la otra) simultáneamente ya que contiene dos procesadores de enteros internos independientes, a lo cual se le llama tecnología superes-

FIGURA 1-2 El índice iCOMP de Intel.



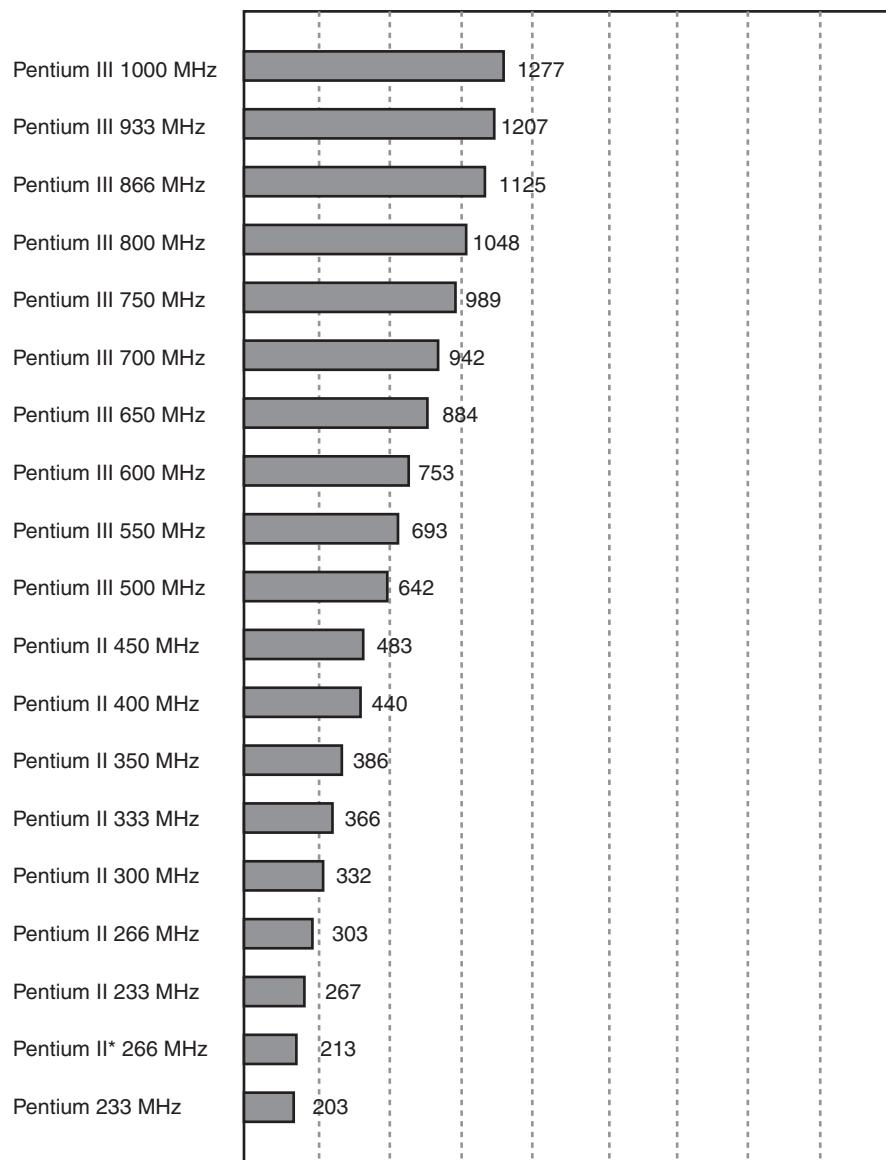
Nota: *Pentium OverDrive, la primera parte de la escala no es lineal, y los procesadores de 166 Mhz y 200 Mhz son tecnología MMX.

calar. Esto permite al Pentium ejecutar dos instrucciones por cada periodo de reloj. Otra característica que mejora el desempeño es una tecnología de predicción de saltos que agiliza la ejecución de los ciclos de un programa. Al igual que con el 80486, el Pentium también emplea un coprocesador interno de punto flotante para manejar los datos de punto flotante, aunque la mejora en velocidad es de cinco veces más. Estas características presagian un éxito continuo para la familia Intel de microprocesadores. Tal vez Intel pueda permitir que el Pentium sustituya algunos de los equipos RISC (**computadora con un conjunto reducido de instrucciones**) que en la actualidad ejecutan una instrucción por cada ciclo del reloj. Algunos procesadores RISC más recientes ejecutan más de una instrucción por ciclo del reloj mediante la introducción de la tecnología superescalar. Motorola, Apple e IBM producen el PowerPC, un microprocesador que tiene dos unidades de números enteros y una unidad de punto flotante. El PowerPC evidentemente impulsa el desempeño de la Apple Macintosh, pero en la actualidad es lento como para emular de manera eficiente la familia de microprocesadores Intel. Las pruebas indican que el software actual de emulación ejecuta aplicaciones DOS y Windows a velocidades más lentas que el microprocesador 80486DX de 25 Mhz. Debido a esto, la familia Intel sobrevivirá por muchos años en los sistemas de computadoras personales. En la actualidad hay 6 millones de sistemas Apple Macintosh⁵ y más de 260 millones de computadoras personales basadas en los microprocesadores Intel. En 1998 los informes indicaron que el 96% de todas las PCs se vendían con el sistema operativo Windows instalado.

Para poder comparar las velocidades de varios microprocesadores, Intel ideó el índice de clasificación iCOMP. Este índice es una mezcla de SPEC92, ZD Bench y Power Meter. El índice de clasificación iCOMP se utiliza para clasificar la velocidad de todos los microprocesadores Intel hasta el Pentium. La figura 1-2 muestra las velocidades relativas de la versión del 80386DX a 25 Mhz en el extremo inferior, hasta la versión del Pentium a 233 Mhz en el extremo superior del espectro.

⁵Macintosh es una marca registrada de Apple Computer Corporation.

FIGURA 1-3 El índice iCOMP2 de Intel.

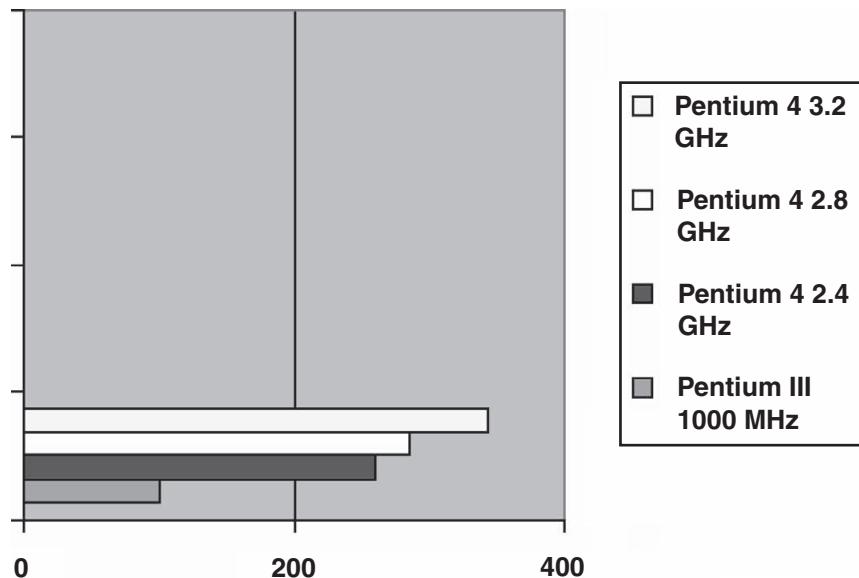


Nota: *Pentium II, Celeron, sin caché. Las cifras de iCOMP2 se muestran arriba. Para convertir a iCOMP3 multiplique por 2.568.

Desde que el Pentium Pro y el Pentium II salieron al mercado, Intel cambió al índice iCOMP2, el cual está escalado por un factor de 10, con base en el índice iCOMP1. Un microprocesador con un índice de 1000 que utiliza la clasificación iCOMP1 se clasifica como 100 si utiliza iCOMP2. Otra diferencia es el tipo de pruebas de rendimiento (benchmarks) que se utiliza para las puntuaciones. La figura 1-3 muestra el índice iCOMP2, en donde aparece el Pentium III a velocidades de hasta 1000 Mhz. La figura 1-4 muestra a SYSmark 2002 para el Pentium III y el Pentium 4.

Procesador Pentium Pro. Una reciente adición de Intel es el procesador Pentium Pro, antes conocido como el microprocesador P6. Este procesador contiene 21 millones de transistores, tres unidades de enteros y una unidad de punto flotante para aumentar el rendimiento de la mayoría del software. La frecuencia de reloj básica era de 150 Mhz y de 166 Mhz en el ofrecimiento inicial que se hizo disponible a finales de 1995. Además de la caché interna de nivel uno (L1) de 16 K (8 K para datos y 8 K para

FIGURA 1-4 El desempeño de los microprocesadores Intel, usando el SYSmark 2002.



instrucciones), el procesador Pentium Pro también contiene una caché de nivel dos (L2) de 256 K. Otro de los cambios significativos es que el procesador Pentium Pro utiliza tres motores de ejecución, por lo que puede ejecutar hasta tres instrucciones a la vez, que pueden estar en conflicto y de todas formas ejecutarse en paralelo. Esto lo diferencia del Pentium, que ejecuta dos instrucciones en forma simultánea, siempre y cuando no estén en conflicto. El microprocesador Pentium Pro está optimizado para ejecutar código de 32 bits con eficiencia; por esta razón, a menudo se vendía con equipos que utilizaban Windows NT en vez de versiones normales de Windows 95. Intel lanzó el procesador Pentium Pro para el mercado de los servidores. Otro de los cambios es que el Pentium Pro puede direccionar un sistema de memoria de 4 Gbytes o de 64 Gbytes. El Pentium Pro tiene un bus de direcciones de 36 bits, si está configurado para un sistema de memoria de 64 Gbytes.

Los microprocesadores Pentium II y Xeon. El microprocesador Pentium II (que se sacó al mercado en 1997) representa una nueva dirección para Intel. En vez de ser un circuito integrado como en las versiones anteriores del microprocesador, el Pentium II está colocado en un pequeño tablero de circuitos. La principal razón de este cambio es que la caché L2 que se encuentra en el tablero de circuitos principal del Pentium no era lo suficientemente rápido como para funcionar apropiadamente con el Pentium II. En el sistema Pentium la caché L2 opera a la velocidad del bus del sistema, que es de 60 Mhz o de 66 Mhz. La caché L2 y el microprocesador están en un tablero de circuitos llamado **módulo de Pentium II**. Esta caché L2 en el tablero opera a una velocidad de 133 Mhz y almacena 512 Kbytes de información. El microprocesador en el módulo de Pentium II es en realidad un Pentium Pro con extensiones MMX.

En 1998 Intel cambió la velocidad del bus del Pentium II. Debido a que los microprocesadores Pentium II de 266 Mhz a 333 Mhz utilizaban una velocidad de bus externo de 66 Mhz y había un cuello de botella, por lo que los microprocesadores Pentium II más recientes utilizan una velocidad del bus de 100 Mhz. Los microprocesadores Pentium II con velocidades de 350 Mhz, 400 Mhz y 450 Mhz utilizan esta velocidad de bus de memoria de 100 Mhz. El bus de memoria de mayor velocidad requiere el uso de memoria SDRAM de 8 ns, en vez de la SDRAM de 10 ns que se utiliza con el bus de 66 Mhz de velocidad.

A mediados de 1998 Intel anunció una nueva versión del Pentium II de nombre Xeon,⁶ que estaba diseñado específicamente para aplicaciones de estaciones de trabajo de alta tecnología y aplicaciones de servidor. La principal diferencia entre el Pentium II y el Pentium II Xeon es que el Xeon está disponible con un tamaño de caché L1 de 32 Kbytes y un tamaño de caché L2 de 512 K, 1 M o 2 M bytes. El Xeon funciona con el juego de chips 440GX. El Xeon también está diseñado para funcionar

⁶Xeon es una marca registrada de Intel Corporation.

con cuatro Xeons en el mismo sistema, lo cual es similar al Pentium Pro. Este producto más nuevo representa un cambio en la estrategia de Intel: ahora Intel produce una versión profesional y una versión para hogar/oficina del microprocesador Pentium II.

Microprocesador Pentium III. Este microprocesador utiliza un núcleo más rápido que el Pentium II, pero sigue siendo un procesador P6 o Pentium Pro. También está disponible en versión Slot 1 (montado en un cartucho de plástico) y en una versión Socket 370 (conocida como “flip-chip”, que tiene una apariencia similar al paquete del Pentium anterior. Intel asegura que la versión flip-chip cuesta menos. Otra diferencia es que el Pentium III está disponible en frecuencias de reloj de hasta 1 Ghz. La versión Slot 1 contiene una caché de 512 K y la versión flip-chip contiene una caché de 256 K. Las velocidades son comparables, ya que la caché en la versión Slot 1 opera a la mitad de la velocidad del reloj, mientras que la caché en la versión flip-chip opera a la velocidad del reloj. Ambas versiones utilizan una velocidad de bus de memoria de 100 Mhz, mientras que el Celeron⁷ utiliza una velocidad de reloj de 66 Mhz para el bus de memoria.

La velocidad del bus frontal (la conexión del microprocesador al controlador de memoria, controlador PCI y controlador AGP) es ahora de 100 Mhz o de 133 Mhz. Aunque la memoria aún se ejecuta a 100 Mhz, este cambio ha mejorado el rendimiento.

Microprocesador Pentium 4. El microprocesador Pentium 4, la más reciente producción de Intel, se puso a disposición del público a finales del 2000. Este procesador, al igual que los procesadores Pentium Pro hasta el Pentium III, utilizan la arquitectura P-6 de Intel. La principal diferencia es que el Pentium 4 está disponible en velocidades de hasta 3.2 Ghz y más, además de que el juego de chips que soporta el Pentium 4 utiliza las tecnologías RAMBUS o memoria DDR en vez de la tecnología SDRAM, que una vez fue el estándar. Estas velocidades superiores del microprocesador son posibles gracias a una mejora en el tamaño de la integración interna, que actualmente es una tecnología de 0.095 micrones. También es interesante tener en cuenta que Intel ha cambiado el tamaño de la caché de nivel uno de 32 K a 8 Kbytes. Las investigaciones deben haber demostrado que este tamaño es lo suficientemente grande para la versión de producción inicial del microprocesador, y las futuras versiones probablemente incluyan una caché L1 de 32 K. La caché L2 sigue teniendo 256 Kbytes como en la versión Coppermine del Pentium, y las versiones más recientes contienen una caché de 512 K. El Pentium 4 Extreme Edition contiene una caché L2 de 2 M y el Pentium 4e contiene una caché L2 de 1 M.

Otro cambio que es muy probable es el pasar de interconexiones de aluminio a interconexiones de cobre dentro del microprocesador. Como el cobre es un mejor conductor, debe permitir frecuencias de reloj mayores para el microprocesador en el futuro. Esto es evidentemente cierto ahora que ha surgido un método para utilizar cobre en IBM Corporation. Otro suceso que hay que esperar es un cambio en la velocidad del bus frontal, que es probable que aumentará más allá del máximo actual de 200 Mhz.

La tabla 1-3 muestra los diversos números P de Intel y los microprocesadores que corresponden a cada clase. Las versiones P muestran qué núcleo interno se encuentra en cada uno de los microprocesadores Intel. Observe que todos los microprocesadores desde el Pentium Pro utilizan el mismo núcleo básico de microprocesador.

TABLA 1-3 Versiones del núcleo (P) de los microprocesadores Intel.

Versión del núcleo (P)	Microprocesador
P1	8086, 8088, 80186, y 80188
P2	80286
P3	80386
P4	80486
P5	Pentium
P6	Pentium Pro, Pentium II, Pentium III, y Pentium 4

⁷Celeron es una marca registrada de Intel Corporation.

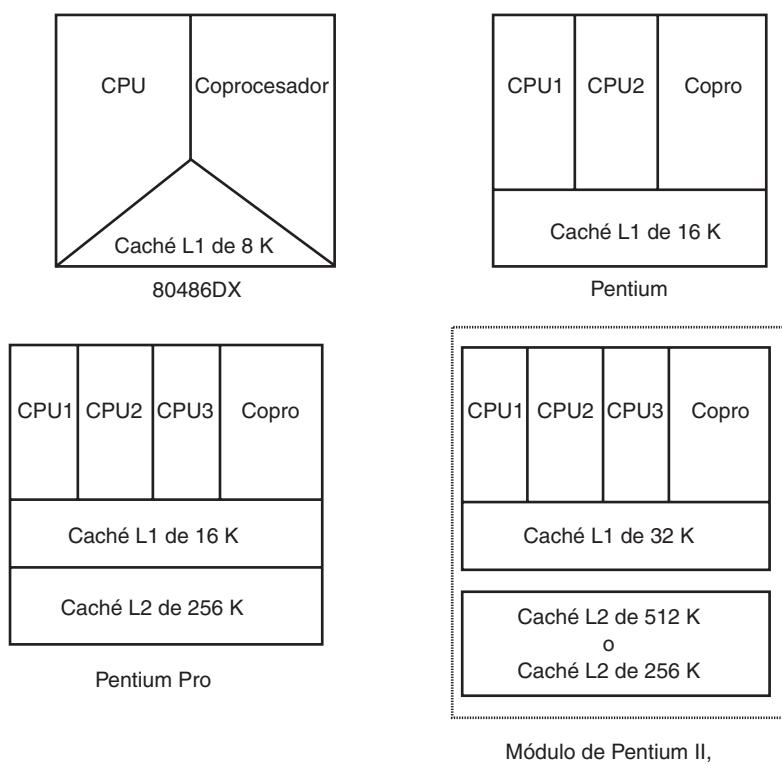
El futuro de los microprocesadores. Nadie puede hacer predicciones precisas, pero el éxito de la familia Intel debe continuar por muchos años todavía. Lo que puede ocurrir es un cambio en la tecnología RISC, pero es más probable que haya mejoras en una nueva tecnología conjunta entre Intel y Hewlett-Packard llamada tecnología Hyper-Threading. Incluso hasta esta nueva tecnología incluye el conjunto de instrucciones CISC de la familia de microprocesadores 80X86, por lo que sobrevivirá el software para el sistema. La premisa básica detrás de esta tecnología es que muchos microprocesadores se comunican directamente entre sí, lo que permite el procesamiento en paralelo sin necesidad de cambios en el conjunto de instrucciones o en el programa. En la actualidad la tecnología superescalar utiliza muchos microprocesadores, pero todos comparten el mismo conjunto de registros. Esta nueva tecnología contiene muchos microprocesadores, cada uno de los cuales contiene su propio conjunto de registros que está vinculado con los registros de los otros microprocesadores. Esta tecnología ofrece un verdadero procesamiento en paralelo sin necesidad de escribir un programa especial.

La tecnología Hyper-Threading debe continuar en el futuro, brindando aún más procesadores en paralelo (actualmente son dos procesadores). Hay información que indica que Intel puede también incorporar el juego de chips en el paquete del microprocesador.

En el 2002, Intel sacó al mercado una nueva arquitectura de microprocesador que tiene 64 bits de anchura y un bus de datos de 128 bits. Esta nueva arquitectura conocida como Itanium,⁸ es un proyecto llamado EPIC (Computación de instrucciones explícitamente en paralelo) realizado conjuntamente por Intel y Hewlett-Packard. La arquitectura Itanium permite un mayor paralelismo que las arquitecturas tradicionales, como el Pentium III o el Pentium 4. Estos cambios incluyen 128 registros enteros de propósito general, 128 registros de punto flotante, 64 registros de predicción y muchas unidades de ejecución para asegurar suficientes recursos de hardware para el software. El Itanium está diseñado para el mercado de servidores y podría o no llegar al mercado doméstico/comercial en el futuro.

La figura 1-5 es una vista conceptual en la que se comparan los microprocesadores desde el 80486 hasta el Pentium 4. Cada vista muestra la estructura interna de estos microprocesadores: el CPU, el coprocesador y la memoria caché. Esta ilustración muestra la complejidad y el nivel de integración en cada versión del microprocesador.

FIGURA 1-5 Vistas conceptuales de los microprocesadores 80486, Pentium Pro, Pentium II, Pentium III y Pentium 4.



⁸Itanium es una marca registrada de Intel Corporation.

1-2

EL SISTEMA DE COMPUTADORA PERSONAL BASADO EN MICROPROCESADOR

Los sistemas computacionales han pasado por muchos cambios últimamente. Los equipos que una vez llenaron grandes áreas se han reducido a pequeños sistemas computacionales de escritorio debido al microprocesador. Aunque estas computadoras de escritorio son compactas, poseen un poder de cómputo que hace unos cuantos años tan sólo era un sueño. Los sistemas computacionales tipo mainframe de un millón de dólares, desarrollados a principios de 1980, no son tan poderosos como las computadoras basadas en el Pentium 4 de la actualidad. De hecho, muchas compañías pequeñas han sustituido sus computadoras mainframe con sistemas basados en microprocesador. Compañías como DEC (Digital Equipment Corporation, ahora propiedad de Hewlett-Packard Company) han dejado de producir sistemas computacionales tipo mainframe para concentrar sus recursos en los sistemas basados en microprocesador.

En esta sección veremos la estructura del sistema de computadora personal basado en microprocesador. Esta estructura incluye información sobre la memoria y el sistema operativo utilizados en muchos sistemas computacionales basados en microprocesador.

En la figura 1-6 podrá ver el diagrama de bloques de la computadora personal. Este diagrama también se aplica a cualquier sistema computacional, desde las primeras computadoras mainframe hasta los más recientes sistemas basados en microprocesador. El diagrama de bloques está compuesto de tres bloques interconectados por buses. (Un **bus** es un conjunto de conexiones comunes que llevan el mismo tipo de información. Por ejemplo, el bus de direcciones, que contiene 20 o más conexiones, transporta la dirección de la memoria a la computadora.) En esta sección describiremos estos bloques y su función en una computadora personal.

La memoria y el sistema de E/S

Las estructuras de memoria de todas las computadoras personales basadas en Intel son similares. Aquí se incluyen desde las primeras computadoras personales basadas en el 8088, introducidas en 1981 por IBM, hasta las versiones más poderosas de alta velocidad de la actualidad, basadas en el Pentium 4. La figura 1-7 ilustra el mapa de memoria de un sistema de computadora personal. Este mapa se aplica a cualquier computadora personal IBM, o a cualquiera de los muchos clones compatibles con IBM que hay en existencia.

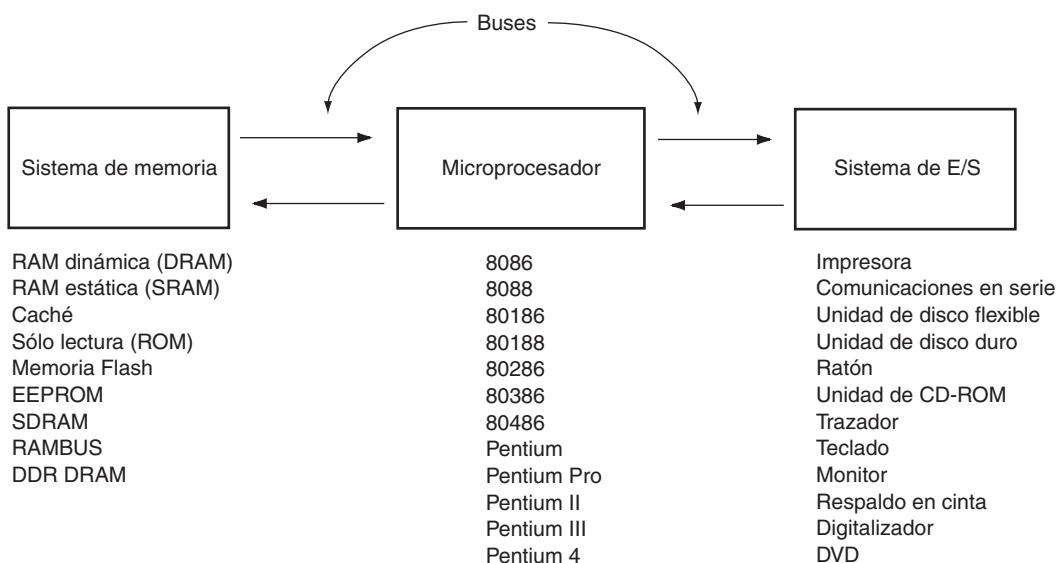
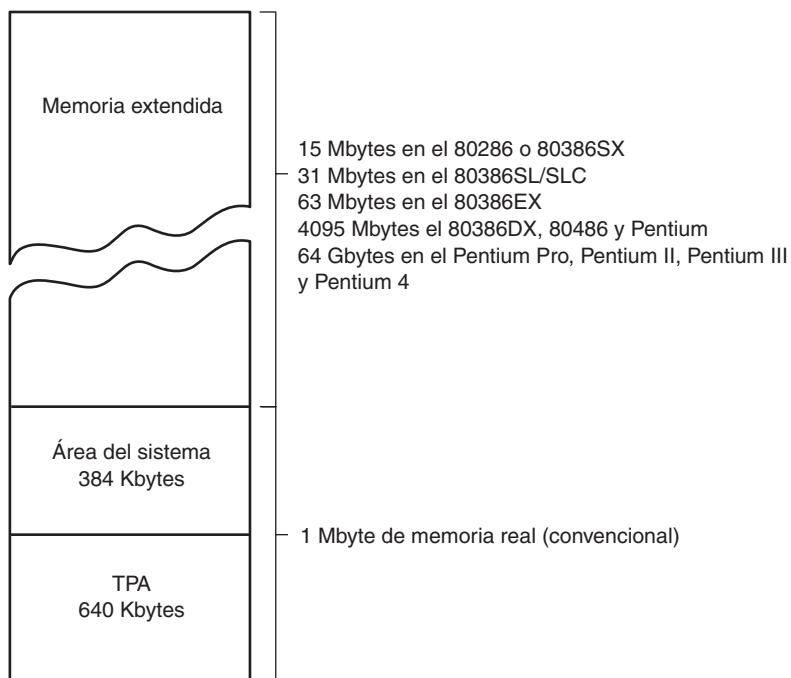


FIGURA 1-6 El diagrama de bloques de un sistema computacional basado en microprocesador.

FIGURA 1-7 El mapa de memoria de la computadora personal.



El sistema de memoria está dividido en tres partes principales: TPA (**área de programas transitorios**), área del sistema y XMS (**sistema de memoria extendida**). El tipo de microprocesador en su computadora determina si existe o no un sistema de memoria extendida. Si la computadora está basada en un procesador 8086 u 8088 es realmente vieja (una PC o XT), la TPA y el área de sistema existen, pero no hay área de memoria extendida. Las computadoras PC y XT contienen 640 Kbytes de TPA y 384 Kbytes de memoria de sistema, para un tamaño total de memoria de 1 Mbyte. A este primer 1 Mbyte de memoria se le llama por lo general el sistema de memoria real o convencional, ya que cada microprocesador Intel está diseñado para funcionar en esta área utilizando su modo real de operación.

Los sistemas computacionales basados en el 80286 y hasta el Pentium 4 no sólo contienen la TPA (640 Kbytes) y el **área de sistema** (384 Kbytes), sino que también contienen memoria extendida. A menudo a estos equipos se les llama equipos clase AT. Las computadoras PS/1 y PS/2, producidas por IBM, son otras versiones del mismo diseño básico de memoria. A estos equipos se les conoce también como equipos ISA (**arquitectura estándar de la industria**) o EISA (**EISA extendida**). A la PS/2 se le conoce como sistema de arquitectura de microcanal, o sistema ISA, dependiendo del número de modelo.

Un cambio que comenzó con la introducción del microprocesador Pentium y el equipo clase ATX es la adición de un bus llamado PCI (**interconexión de componentes periféricos**), el cual se utiliza ahora en todos los sistemas Pentium-Pentium 4. La memoria extendida contiene hasta 15 Mbytes en las computadoras basadas en el 80286 y 80386SX, y hasta 4095 Mbytes en los microprocesadores 80386DX, 80486 y Pentium, además del primer 1 Mbyte de memoria real o convencional. Los sistemas computacionales que usan desde Pentium Pro hasta Pentium 4 tienen 1 M menos que 4 G o 1 M menos que 64 G de memoria extendida. Los servidores tienden a usar el mapa de memoria de 64 G, mientras que las computadoras domésticas/comerciales utilizan el mapa de memoria de 4 Gbytes. El equipo ISA contiene un bus periférico de 8 bits que se utiliza para conectar los dispositivos de 8 bits a la computadora en el sistema computacional PC o XT basado en el 8086/8088. El equipo clase AT, también llamado equipo ISA, utiliza un bus periférico de 16 bits como interfaz y puede contener el microprocesador 80286 o los antes mencionados. El bus EISA es un bus de interfaz periférica de 32 bits que se encuentra en sistemas antiguos basados en el 80386DX y el 80486. Observe que cada uno de estos buses es compatible con las versiones anteriores. Esto es, una tarjeta de interfaz de 8 bits funciona en el sistema de bus ISA de 8 bits, de 16 bits, o en el EISA de 32 bits. De igual forma, una tarjeta de interfaz de 16 bits funciona en el sistema ISA de 16 bits o en el EISA de 32 bits.

Otro tipo de bus que existe en muchas computadoras personales basadas en el 80486 es el bus local **VESA**, o bus VL. El **bus local** conecta el disco y el vídeo con el microprocesador a nivel del bus local, el cual permite que las interfaces de 32 bits funcionen a la misma velocidad de reloj que el microprocesador. Una modificación reciente al bus local VESA es que soporta el bus de datos de 64 bits del microprocesador Pentium y compite directamente con el bus PCI, aunque ha generado muy poco (si acaso) interés. Los estándares ISA y EISA funcionan a sólo 8 Mhz, lo cual reduce el rendimiento de las interfaces de disco y de vídeo que utilizan estos estándares. El bus PCI es un bus de 32 o de 64 bits que está diseñado específicamente para funcionar con los microprocesadores Pentium a Pentium 4, a una velocidad de bus de 33 Mhz.

Han aparecido tres buses más recientes en los sistemas clase ATX. El primero fue el **USB (bus serial universal)**. Este bus está diseñado para conectar dispositivos periféricos tales como teclados, un ratón, módems y tarjetas de sonido al microprocesador, a través de una ruta de datos en serie y un par trenzado de cables. La idea principal es reducir el costo del sistema reduciendo el número de cables. Otra ventaja es que el sistema de sonido puede tener una fuente de energía separada de la PC, lo cual significa mucho menos ruido. Las velocidades de transferencia de datos a través del USB son de 10 Mbps hasta este momento para el USB1; aumenta hasta 480 Mbps para el USB2.

El segundo bus más reciente es el AGP (**puerto de gráficos avanzados o acelerados**) para las tarjetas de vídeo. Este puerto de gráficos avanzados transfiere datos entre la tarjeta de vídeo y el microprocesador a mayores velocidades (66 Mhz con una ruta de datos de 64 bits, o 533 Mbytes por segundo) de las que era posible obtener con cualquier otro bus o conexión. La velocidad del AGP más reciente es de 8X, o 2 Gbytes por segundo. Este cambio en el subsistema de vídeo se hizo para ajustarse a los nuevos reproductores de DVD para la PC.

El tercer bus más reciente es la interfaz ATA serial (**SATA**) para las unidades de disco duro, que transfiere datos de la PC al disco duro con velocidades de 150 Mbytes por segundo. El estándar ATA serial llegará eventualmente a velocidades de 450 Mbytes por segundo.

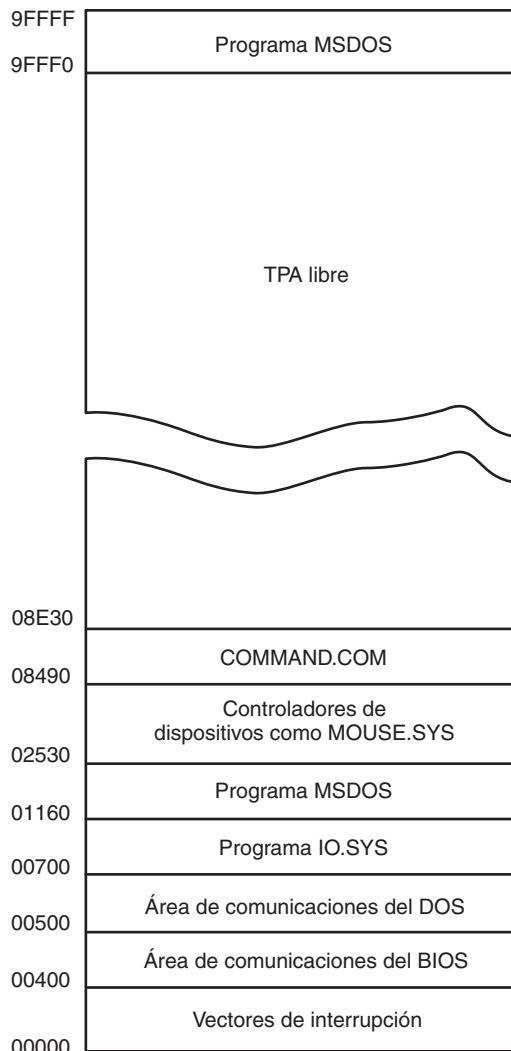
La TPA. El área de programas transitorios (TPA) guarda el sistema operativo DOS (**sistema operativo en disco**) y otros programas que controlan el sistema computacional. La TPA es un concepto de DOS y no se aplica realmente en Windows. La TPA también almacena cualquier programa de aplicación de DOS activo o inactivo. La longitud de la TPA es de 640 Kbytes. Como se mencionó anteriormente, esta área de memoria guarda el sistema operativo DOS, el cual requiere que funcione una porción de la TPA. En la práctica, la cantidad de memoria remanente para el software de aplicación es de aproximadamente 628 Kbytes si se utiliza el MSDOS⁹ versión 7.x como sistema operativo. Las versiones anteriores de DOS requerían más del área de la TPA y a menudo dejaban sólo 530 Kbytes o menos para los programas de aplicación. La figura 1-8 muestra la organización de la TPA en un sistema computacional que ejecuta DOS.

El mapa de memoria del DOS muestra cómo se utilizan las distintas áreas de la TPA para los programas, datos y controladores del sistema. También muestra una gran área de memoria disponible para los programas de aplicaciones. A la izquierda de cada área hay un número hexadecimal que representa las direcciones de memoria que empiezan y terminan cada área de datos. Se utilizan **direcciones de memoria o posiciones de memoria** en hexadecimal para numerar cada byte del sistema de memoria. (Un **número hexadecimal** es un número representado en raíz 16 o base 16, en donde cada dígito representa un valor de 0 a 9 y de A a F. Por lo general, un número hexadecimal termina con una H para indicar que es un valor hexadecimal. Por ejemplo, 1234H es 1234 hexadecimal. También se representan los datos hexadecimales como 0x1234 para un 1234 hexadecimal.)

Los vectores de Interrupción acceden a varias características del DOS, del BIOS (**sistema básico de E/S**) y de las aplicaciones. El BIOS del sistema es una colección de programas almacenados en una memoria flash o de sólo lectura (ROM), que opera muchas de los dispositivos de E/S conectados a su sistema computacional. El BIOS del sistema y las áreas de comunicaciones del DOS contienen datos transitorios utilizados por los programas para acceder a los dispositivos de E/S y a las características internas del sistema computacional. Éstos se almacenan en la TPA, por lo que pueden cambiarse a medida que el DOS opere.

⁹MSDOS es una marca registrada de Microsoft Corporation y la versión 7.x se incluye con Windows XP.

FIGURA 1-8 El mapa de memoria de la TPA en una computadora personal.
(Este mapa varía entre los sistemas.)



El IO.SYS es un programa que se carga en la TPA desde el disco, siempre que se inicia un sistema MSDOS. El IO.SYS contiene programas que permiten al DOS utilizar el teclado, la pantalla de vídeo, la impresora y otros dispositivos de E/S que se encuentran con frecuencia en el sistema computacional. El programa IO.SYS vincula el DOS a los programas almacenados en la ROM del BIOS del sistema.

El tamaño del área de los controladores y el número de ellos varían de una computadora a otra. Los **controladores** son programas que controlan los dispositivos de E/S instalables, como un ratón, la caché de disco, el digitalizador manual, la memoria de CD-ROM (**Memoria de disco compacto de sólo lectura**), el DVD (**Disco versátil digital**) o los dispositivos instalables, así como los programas. Los controladores instalables son programas que controlan dispositivos o programas que se agregan al sistema computacional. Los controladores de DOS son por lo común archivos que tienen la extensión .SYS, como MOUSE.SYS; en el DOS versión 3.2 o posterior los archivos tienen la extensión .EXE, como EMM386.EXE. Aunque estos archivos no los utiliza Windows, aún se utilizan para ejecutar aplicaciones DOS, incluso con Windows XP. Windows utiliza un archivo llamado SYSTEM.INI para cargar los controladores utilizados por Windows. En versiones más recientes de Windows tales como Windows XP, se agrega un registro para contener información sobre el sistema y los controladores usados por el mismo. Usted puede ver el registro con el programa REGEDIT.

El programa COMMAND.COM (**procesador de comandos**) controla la operación de la computadora desde el teclado cuando se opera en modo DOS. El programa COMMAND.COM procesa los comandos de DOS a medida que se escriben desde el teclado. Por ejemplo, si se escribe DIR el programa

COMMAND.COM muestra un directorio de los archivos en el directorio del disco actual. Si se borra el programa COMMAND.COM, la computadora no puede usarse desde el teclado en modo DOS. Nunca borre los archivos COMMAND.COM, IO.SYS o MSDOS.SYS para hacer espacio para otro software, o su computadora no funcionará.

El área del sistema. Aunque es más pequeña que la TPA, el área del sistema DOS es igual de importante. El área del sistema contiene programas en memoria de sólo lectura (ROM) o en memoria flash, y áreas de memoria de lectura/escritura (RAM) para almacenar datos. La figura 1-9 muestra el área del sistema de un sistema computacional típico. Al igual que con el mapa de la TPA, este mapa también incluye las direcciones de memoria hexadecimales de las diversas áreas.

La primer área del espacio del sistema contiene la RAM de visualización de vídeo y los programas de control de vídeo en memoria ROM o flash. Esta área empieza en la posición A0000H y se extiende hasta la posición C7FFFH. El tamaño y la cantidad de memoria utilizada depende del tipo de adaptador de visualización de gráficos conectado al sistema. Los adaptadores de pantalla por lo general tienen su RAM de vídeo ubicada en A0000H-AFFFFH, en donde se almacenan datos gráficos o mapas de bits, y la memoria en B0000H-BFFFFH almacena datos de texto. El BIOS de vídeo, que se encuentra en una memoria ROM o flash, está en las posiciones C0000H-C7FFFH y contiene programas que controlan la pantalla de vídeo del DOS.

El área en las posiciones C8000H-DFFFFH por lo general está abierta o libre. Esta área se utiliza para el sistema de memoria expandida (EMS) en un sistema PC o XT, o para el sistema de memoria superior en un sistema AT. Su uso depende del sistema y de su configuración. El sistema de memoria expandida permite que los programas de aplicaciones utilicen una trama de página de memoria de 64 Kbytes. Esta trama de página de 64 Kbytes (generalmente en las posiciones D0000H hasta

FIGURA 1-9 El área del sistema de una computadora personal típica.

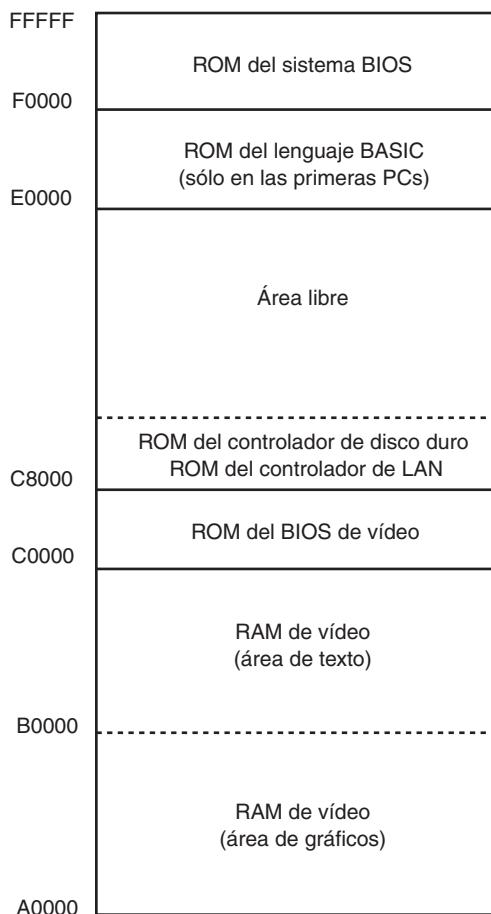
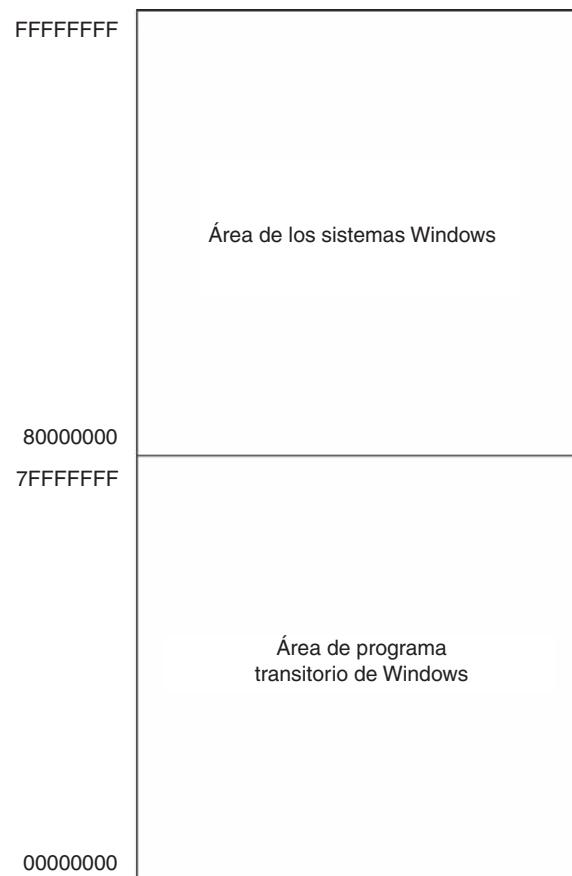


FIGURA 1-10 El mapa de memoria utilizado por Windows XP.



la DFFFFH) se utiliza para expandir el sistema de memoria, cambiando páginas de memoria del EMS hacia este rango de direcciones de memoria.

Las posiciones de memoria E0000H-EFFFFH contienen el casete de lenguaje BASIC en ROM que tenían los primeros sistemas de computadora personal de IBM. A menudo esta área está abierta o libre en los sistemas computacionales más recientes.

Por último, la ROM del BIOS del sistema se encuentra en los primeros 64 Kbytes del área del sistema (F0000H-FFFFFH). Esta ROM controla la operación de los dispositivos básicos de E/S conectados al sistema computacional. No controla la operación del sistema de vídeo, el cual tiene su propia ROM de BIOS en la posición C0000H. La primera parte del BIOS del sistema (F0000H-F7FFFH) a menudo contiene programas para configurar la computadora; la segunda parte contiene procedimientos que controlan el sistema básico de E/S.

Sistemas Windows. Las computadoras modernas utilizan un mapa de memoria distinto a los mapas de memoria del DOS de las figuras 1-8 y 1-9. El mapa de memoria de Windows aparece en la figura 1-10 y tiene dos áreas principales, un área para la TPA y un área del sistema. La diferencia entre este mapa de memoria y el de DOS la representan los tamaños y las ubicaciones de estas áreas.

La TPA de Windows está compuesta por los primeros 2 Gbytes del sistema de memoria, desde la posición 00000000H hasta la posición 7FFFFFFFH. El área del sistema de Windows está compuesta por los últimos 2 Gbytes de memoria, desde la posición 80000000H hasta la posición FFFFFFFFH. Parece ser que la misma idea utilizada para construir el mapa de memoria del DOS se utilizó también en un sistema moderno basado en Windows. El área del sistema es en donde se encuentran el BIOS del sistema y la memoria de vídeo. En esta área también se encuentra el programa Windows y los controladores. Todos los programas escritos para Windows pueden usar hasta 2 Gbytes de memoria ubicada en las direcciones lineales 00000000H hasta la 7FFFFFFFH.

¿Significa esto que mi programa empezará en la dirección física 00000000H? No, el mapa físico del sistema de memoria es muy distinto del modelo de programación lineal que se muestra en la figura 1-10. Cada proceso en un sistema Windows XP o 2000 tiene su propio conjunto de tablas de página, los cuales definen en qué parte de la memoria física se encuentra cada página de 4 Kbytes del proceso. Esto significa que el proceso puede ubicarse en cualquier parte de la memoria, incluso en páginas que no estén contiguas. Más adelante en este capítulo hablaremos sobre las tablas de páginas y la estructura de paginación del microprocesador, ya que están más allá del alcance del texto hasta este punto. En cuanto a lo que a una aplicación respecta, siempre tendrá 2 Gbytes de memoria, aún si la computadora tiene menos. El sistema operativo (Windows) se encarga de asignar la memoria física a la aplicación y, si no existe suficiente memoria física, utiliza la unidad de disco duro para cualquier cantidad que no esté disponible.

Espacio de E/S. El espacio de E/S (entrada/salida) en un sistema computacional se extiende desde el puerto de E/S 0000H hasta el puerto FFFFH. (Una **dirección de puerto de E/S** es similar a una dirección de memoria, sólo que en vez de direccionar memoria direcciona un dispositivo de E/S.) Los dispositivos de E/S permiten que el microprocesador se comunique con el mundo exterior. El espacio de E/S permite que la computadora utilice hasta 64 K para distintos dispositivos de E/S de 8 bits, 32 K para distintos dispositivos de 16 bits o 16 K para distintos dispositivos de 32 bits. Hay muchas de estas posiciones disponibles para la expansión de la mayoría de los sistemas computacionales. Para ver el mapa en su computadora en Windows XP, vaya al Panel de control | Rendimiento y mantenimiento | Sistema, seleccione la ficha Hardware | Administrador de dispositivos | Ver | Recursos por tipo y haga clic en el signo positivo al lado de Entrada/Salida (E/S).

El área de E/S contiene dos secciones principales. El área debajo de la posición de E/S 0400H se considera reservada para los dispositivos del sistema; muchos se muestran en la figura 1-11. El área restante es el espacio de E/S disponible para expansión, que se extiende desde el puerto de E/S 0400H hasta el puerto FFFFH. Por lo general, las direcciones de E/S entre 0000H y 00FFH direccionan componentes en el tablero principal de la computadora (motherboard), mientras que las direcciones entre 0100H y 03FFH direccionan los dispositivos ubicados en tarjetas insertables (o en el tablero principal). Hay que tener en cuenta que la limitación de las direcciones de E/S entre 0000 y 03FFH se debe al estándar original de la PC, especificado por IBM. Al utilizar el bus ISA, sólo se pueden utilizar direcciones entre 0000H y 03FFH. El bus PCI utiliza las direcciones de E/S entre 0400H y FFFFH.

Varios dispositivos de E/S que controlan la operación del sistema, por lo general, no se dirigen directamente, sino que el ROM del BIOS del sistema direcciona estos dispositivos básicos, que pueden variar ligeramente en cuanto a su ubicación y su función, de una computadora a otra. Siempre se debe acceder a la mayoría de los dispositivos de E/S a través de llamadas a funciones de Windows, DOS o BIOS para mantener la compatibilidad de un sistema computacional a otro. El mapa que se muestra en la figura 1-11 es una guía para mostrar el espacio de E/S en el sistema.

El microprocesador

En el corazón del sistema computacional basado en microprocesador está el circuito integrado del microprocesador. Este elemento, que algunas veces se le denomina como el CPU (**unidad central de proceso**), es el elemento de control en un sistema computacional. El microprocesador controla la memoria y la E/S a través de una serie de conexiones llamadas buses. Los buses seleccionan un dispositivo de E/S o de memoria, transfieren datos entre un dispositivo de E/S o la memoria y el microprocesador, y controlan el sistema de E/S y de memoria. La memoria y la E/S se controlan mediante instrucciones que se almacenan en la memoria para que el microprocesador las ejecute.

El microprocesador realiza tres tareas principales para el sistema computacional: (1) transfiere datos entre sí mismo y los sistemas de memoria o de E/S, (2) operaciones simples aritméticas y lógicas, y (3) el flujo del programa mediante decisiones simples. Aunque éstas son tareas sencillas, es por medio de ellas que el microprocesador realiza casi cualquier serie de operaciones o tareas.

El poder del microprocesador está en su capacidad de ejecutar billones de instrucciones por segundo, provenientes de un programa o software (**grupo de instrucciones**) almacenado en el sistema de memoria. Este concepto de programa almacenado ha convertido al microprocesador y al sistema compu-

FIGURA 1-11

Algunas posiciones de E/S en una computadora personal típica.

	Input/output (IO)
	[00000000 - 0000000F] Direct memory access controller
	[00000000 - 00000CF7] PCI bus
	[00000010 - 0000001F] Motherboard resources
	[00000020 - 00000021] Programmable interrupt controller
	[00000022 - 0000002D] Motherboard resources
	[0000002E - 0000002F] Motherboard resources
	[00000030 - 0000003F] Motherboard resources
	[00000040 - 00000043] System timer
	[00000044 - 0000005F] Motherboard resources
	[00000060 - 00000060] Easy Internet Keyboard
	[00000061 - 00000061] System speaker
	[00000062 - 00000063] Motherboard resources
	[00000064 - 00000064] Easy Internet Keyboard
	[00000065 - 0000006F] Motherboard resources
	[00000070 - 00000073] System CMOS/real time clock
	[00000074 - 0000007F] Motherboard resources
	[00000080 - 00000090] Direct memory access controller
	[00000091 - 00000093] Motherboard resources
	[00000094 - 0000009F] Direct memory access controller
	[000000A0 - 000000A1] Programmable interrupt controller
	[000000A2 - 000000BF] Motherboard resources
	[000000C0 - 000000DF] Direct memory access controller
	[000000E0 - 000000EF] Motherboard resources
	[000000F0 - 000000FF] Numeric data processor
	[00000170 - 00000177] Secondary IDE Channel
	[000001F0 - 000001F7] Primary IDE Channel
	[00000200 - 00000207] Standard Game Port
	[00000274 - 00000277] ISAPNP Read Data Port
	[00000279 - 00000279] ISAPNP Read Data Port
	[000002F8 - 000002FF] Communications Port (COM2)
	[00000376 - 00000376] Secondary IDE Channel
	[00000378 - 0000037F] Printer Port (LPT1)
	[000003B0 - 000003BB] ALL-IN-WONDER 9700 SERIES
	[000003B0 - 000003BB] Intel(R) 82845G/GL/GE/PE/GV Processor to AGP Controller - 256I
	[000003C0 - 000003DF] ALL-IN-WONDER 9700 SERIES
	[000003C0 - 000003DF] Intel(R) 82845G/GL/GE/PE/GV Processor to AGP Controller - 256I
	[000003F0 - 000003F1] Motherboard resources
	[000003F2 - 000003F5] Standard floppy disk controller
	[000003F6 - 000003F6] Primary IDE Channel
	[000003F7 - 000003F7] Standard floppy disk controller
	[000003F8 - 000003FF] Communications Port (COM1)
	[000004D0 - 000004D1] Motherboard resources
	[000004D6 - 000004D6] Motherboard resources
	[00000A79 - 00000A79] ISAPNP Read Data Port
	[00000D00 - 0000FFFF] PCI bus
	[00008400 - 0000843F] SoundMAX Integrated Digital Audio
	[00008800 - 000088FF] SoundMAX Integrated Digital Audio

tacional en dispositivos muy poderosos. (Recuerde que Babbage también quería utilizar el concepto de programa almacenado en su Máquina Analítica.)

La tabla 1-4 muestra las operaciones aritméticas y lógicas ejecutadas por la familia de microprocesadores Intel. Estas operaciones son muy básicas, pero a través de ellas pueden resolverse problemas muy complejos. Los datos se operan desde el sistema de memoria o desde los registros internos. Las anchuras de los datos son variables y pueden ser: **byte** (8 bits), **word** (16 bits) y **doubleword** (32 bits). Sólo los microprocesadores del 80386 hasta el Pentium 4 manipulan directamente números de 8, 16 y 32 bits. Los microprocesadores 8086 hasta el 80286 manipulaban directamente números de 8 y de 16 bits, pero no de 32. Comenzando con el 80486, el microprocesador contiene un coprocesador numérico que le permite realizar operaciones aritméticas complejas utilizando aritmética de punto flotante. El coprocesador numérico, que es similar a un chip de calculadora, era un componente adicional en las computadoras personales basadas en los microprocesadores del 8086 al 80386. El coprocesador numérico también es capaz de realizar operaciones enteras en **quadwords** (64 bits). Las unidades MMX y

TABLA 1-4
Operaciones aritméticas
y lógicas simples.

	<i>Operación</i>	<i>Comentario</i>
Suma		
Resta		
Multiplicación		
División		
AND		Multiplicación lógica
OR		Suma lógica
NOT		Inversión lógica
NEG		Inversión aritmética
Desplazamiento		
Rotación		

TABLA 1-5 Decisiones incluidas en los microprocesadores 8086-Pentium 4.

	<i>Decisión</i>	<i>Comentario</i>
Zero		Evalúa si un número es cero o distinto de cero.
Sign		Evalúa el signo positivo o negativo de un número.
Carry		Evalúa un acarreo después de la suma o una sustracción después de la resta.
Parity		Evalúa si un número tiene una cantidad par o impar de unos.
Overflow		Evalúa un desbordamiento que indica un resultado inválido después de una suma con signo o una resta con signo.

SIMD dentro del Pentium-Pentium 4 funcionan con números enteros y de punto flotante en paralelo. La unidad SIMD requiere números almacenados como **octalwords** (128 bits).

Otra característica que hace poderoso al microprocesador es su habilidad de realizar decisiones simples con base en hechos numéricos. Por ejemplo, un microprocesador puede decidir si un número es cero, si es positivo, etcétera. Estas decisiones simples permiten al microprocesador modificar el flujo del programa, de manera que los programas parezcan “pensar” en estas decisiones simples. La tabla 1-5 muestra las capacidades de toma de decisiones de la familia de microprocesadores Intel.

Buses. Un **bus** es un grupo común de cables que interconectan componentes en un sistema computacional. Los buses que interconectan las secciones de un sistema computacional transfieren información sobre direcciones, datos y control entre el microprocesador y sus sistemas de memoria y de E/S. En el sistema computacional basado en microprocesador existen tres buses para esta transferencia de información: de dirección, de datos y de control. La figura 1-12 muestra cómo estos buses interconectan varios componentes del sistema, como el microprocesador, la memoria de lectura/escritura (RAM), la memoria de sólo lectura (ROM o flash) y unos cuantos dispositivos de E/S.

El bus de direcciones solicita a la memoria una posición de memoria, o solicita a los dispositivos de E/S una posición de E/S. Si se direcciona la E/S, el bus de direcciones contiene una dirección de E/S de 16 bits, que puede estar entre 0000H y FFFFH. La dirección de E/S de 16 bits (o número de puerto) selecciona uno de 64 K dispositivos de E/S distintos. Si se direcciona la memoria, el bus de direcciones contiene una dirección de memoria, la cual puede variar en anchura dependiendo de las distintas versiones del microprocesador. El 8086 y el 8088 direccionan 1 Mbyte de memoria por medio de una dirección de 20 bits que selecciona las posiciones de la 00000H a la FFFFFH. El 80286 y el 80386SX direccionan 16 Mbytes de memoria por medio de una dirección de 24 bits que selecciona las posiciones de la 000000H a la FFFFFFFH. El 80386SL, el 80386SLC y el 80386EX direccionan 32 Mbytes de memoria mediante una dirección de 25 bits que selecciona las posiciones de la 0000000H a la 1FFFFFFH. El 80386DX, el 80486SX y el 80486DX direccionan 4 Gbytes de memoria mediante una dirección de

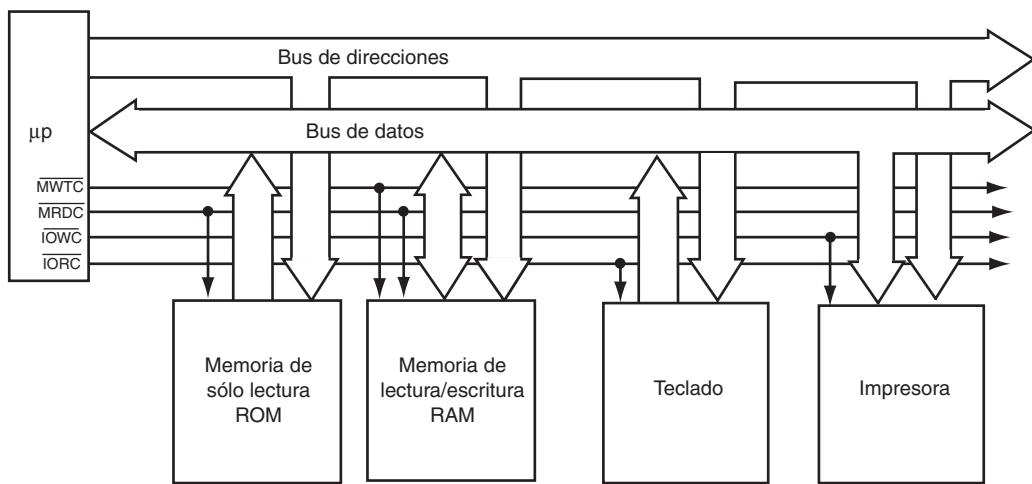


FIGURA 1-12 El diagrama de bloques de un sistema computacional, en el que se muestra la estructura del bus de direcciones, el bus de datos y el bus de control.

32 bits que selecciona las posiciones de la 00000000H a la FFFFFFFFH. El Pentium también direcciona 4 Gbytes de memoria, pero utiliza un bus de datos de 64 bits para utilizar hasta 8 bytes de memoria a la vez. Los microprocesadores del Pentium Pro al Pentium 4 tienen un bus de datos de 64 bits y un bus de direcciones de 32 bits para direccionar 4 G de memoria de las posiciones 00000000H a la FFFFFFFFH, o un bus de direcciones de 36 bits que direcciona 64 G de memoria de las posiciones 00000000H a la FFFFFFFFFFH, dependiendo de su configuración. En la tabla 1-6 encontrará un listado completo de los tamaños de bus y de memoria de la familia de microprocesadores Intel.

El bus de datos transfiere información entre el microprocesador y su espacio de direcciones de memoria y de E/S. Las transferencias de datos varían en tamaño, desde 8 bits hasta 64 bits de ancho en varios miembros de la familia de microprocesadores Intel. Por ejemplo, el 8088 tiene un bus de datos de 8 bits que transfiere 8 bits de datos a la vez. Los microprocesadores 8086, 80286, 80386SL, 80386SX y 80386EX transfieren 16 bits de datos a través de sus buses de datos; los microprocesadores 80386DX, 80486SX y 80486DX transfieren 32 bits de datos; y los microprocesadores del Pentium al Pentium 4 transfieren 64 bits de datos. La ventaja de un bus de datos más ancho es la velocidad en aplicaciones que utilizan datos anchos. Por ejemplo, si se almacena un número de 32 bits en memoria, el microprocesador

TABLA 1-6 Tamaños de bus y de memoria de la familia de microprocesadores Intel.

Micropocesador	Anchura del bus de datos	Anchura del bus de direcciones	Tamaño de la memoria
8086	16	20	1 M
8088	8	20	1 M
80186	16	20	1 M
80188	8	20	1 M
80286	16	24	16 M
80386SX	16	24	16 M
80386DX	32	32	4 G
80386EX	16	26	64 M
80486	32	32	4 G
Pentium	64	32	4 G
Pentium Pro–Pentium 4	64	32	4 G
Pentium Pro–Pentium 4 (si se habilita el direccionamiento extendido)	64	36	64 G

8088 necesita cuatro operaciones de transferencia para completar la operación ya que su bus de datos sólo tiene 8 bits de ancho. El 80486 realiza la misma tarea con una transferencia, ya que su bus de datos es de 32 bits de ancho. La figura 1-13 muestra las anchuras y tamaños de memoria de los microprocesadores 8086-80486 y Pentium-Pentium 4. Observe cómo los tamaños y las organizaciones de la memoria difieren entre los distintos miembros de la familia de microprocesadores Intel. En todos los miembros de la familia, la memoria está numerada por byte. Todos los microprocesadores Pentium-Pentium 4 tienen un bus de datos de 64 bits de ancho.

El bus de control tiene líneas que seleccionan la memoria o E/S y hacen que realice una operación de lectura o de escritura. En la mayoría de los sistemas computacionales hay cuatro conexiones del bus de control: $MRDC$ (control de lectura de memoria), $MWTC$ (control de escritura de memoria), $IORD$ (control de lectura de E/S) y $IOWC$ (control de escritura de E/S). La barra superior indica que la señal de control es activa a baja; es decir, está activa cuando aparece un cero lógico en la línea de control. Por ejemplo, cuando $\overline{IOWC} = 0$, el microprocesador escribe datos del bus de datos hacia un dispositivo de E/S cuya dirección aparece en el bus de direcciones.

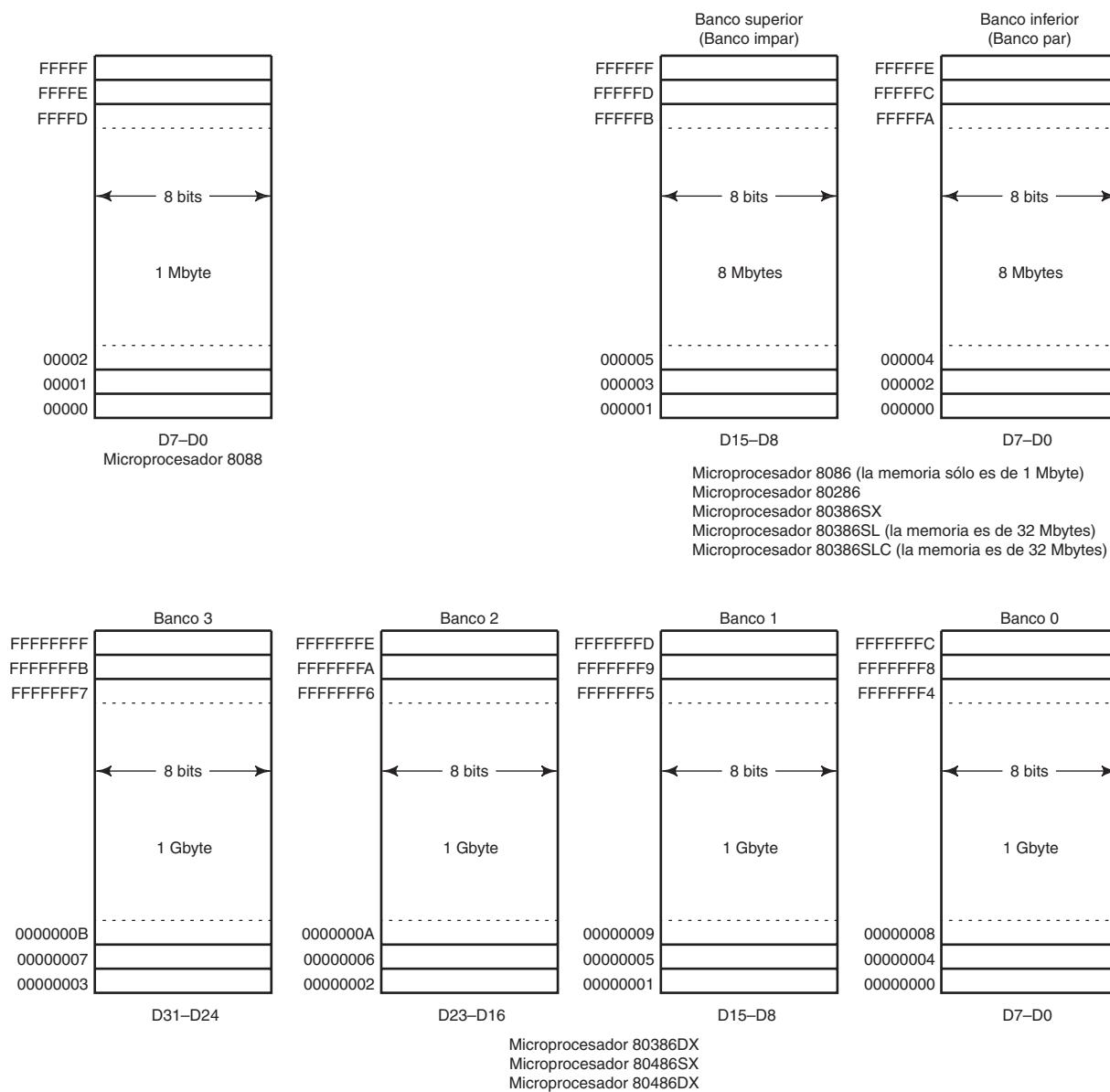
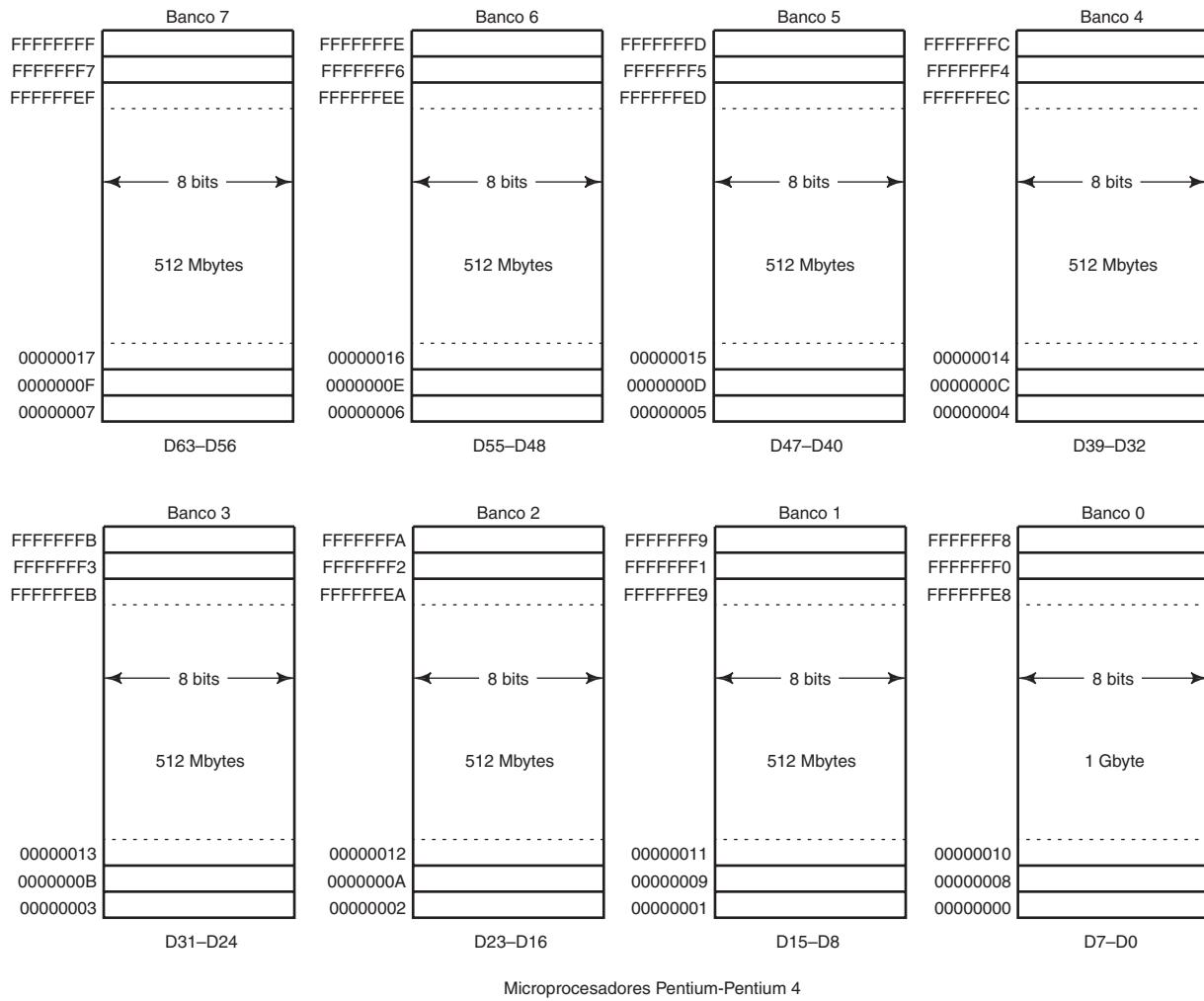


FIGURA 1-13 Los sistemas de memoria física de los microprocesadores 8086 al Pentium 4.

**FIGURA 1-13** (continuación)

El microprocesador lee el contenido de una posición de memoria enviando a la memoria una dirección a través del bus de direcciones; posteriormente envía la señal de control de lectura de memoria (*MRDC*) para hacer que la memoria lea los datos; por último, los datos que lee la memoria se pasan al microprocesador a través del bus de datos. Siempre que ocurre una escritura de memoria, una escritura de E/S o una lectura de E/S se lleva a cabo la misma secuencia, sólo que se emiten distintas señales de control y los datos fluyen del microprocesador a través de su bus de datos en una operación de escritura.

1-3**SISTEMAS NUMÉRICOS**

Para usar el microprocesador se requiere un conocimiento práctico de los sistemas numéricos binario, decimal y hexadecimal. En esta sección veremos las bases sobre estos sistemas numéricos para aquellos que no estén familiarizados. Describiremos las conversiones entre decimal y binario, entre decimal y hexadecimal, y entre binario y hexadecimal.

Dígitos

Antes de convertir los números de una base a otra, hay que comprender qué son los dígitos. Anteriormente aprendimos que un número decimal (base 10) se construye con 10 dígitos: del 0 al 9. El primer dígito en cualquier sistema numérico es siempre cero. Por ejemplo, un número de base 8 (**octal**) contiene 8 dígitos: del 0 al 7; un número de base 2 (**binario**) contiene 2 dígitos: 0 y 1. Si la base de un número se pasa de 10, utilizamos las letras del alfabeto para los dígitos adicionales, empezando con una A. Por ejemplo, un número de base 12 contiene 12 dígitos: del 0 al 9, seguidos de A para el 10 y de B para el 11. Tenga en cuenta que un número de base 10 no contiene un dígito 10, de igual forma que un número de base 8 no contiene un dígito 8. Los sistemas numéricos más comunes que se utilizan en las computadoras son decimal, binario y hexadecimal (base 16). (Anteriormente los números octales eran populares.) En esta sección describiremos y utilizaremos cada uno de estos sistemas numéricos.

Notación posicional

Una vez comprendidos los dígitos de un sistema numérico podemos construir números más grandes mediante el uso de la notación posicional. En la escuela primaria aprendimos que la posición a la izquierda de la posición de las unidades es la posición de las decenas, la posición a la izquierda de las decenas es la posición de las centenas y así sucesivamente. (Un ejemplo es el número decimal 132: este número tiene 1 centena, 3 decenas y 2 unidades.) Lo que probablemente no aprendimos es el valor exponencial de cada posición: la posición de las unidades tiene un peso de 10^0 , o 1; la posición de las decenas tiene un peso de 10^1 , o 10; y la posición de las centenas tiene un peso de 10^2 , o 100. Las potencias exponenciales de las posiciones son imprescindibles para comprender los números en otros sistemas numéricos. La posición a la izquierda del punto de la base (**base numérica**), que se conoce como punto decimal sólo en el sistema decimal, es siempre la posición de las unidades en cualquier sistema numérico. Por ejemplo, la posición a la izquierda del punto binario es siempre 2^0 , o 1; la posición a la izquierda del punto octal es 8^0 , o 1. En cualquier caso, un número elevado a su potencia cero es siempre 1, o la posición de las unidades.

La posición a la izquierda de la posición de las unidades es siempre la base numérica elevada a la primera potencia; en el sistema decimal es 10^1 , o 10. En el sistema binario es 2^1 , o 2; y en el sistema octal es 8^1 , u 8. Por lo tanto, un 11 decimal tiene un valor distinto de un 11 binario. El número decimal está compuesto de 1 diez más 1 unidad, y tiene un valor de 11 unidades; el número binario 11 está compuesto de 1 dos más 1 unidad, para un valor de 3 unidades decimales. El 11 octal tiene un valor de 9 unidades decimales.

En el sistema decimal, las posiciones a la derecha del punto decimal tienen potencias negativas. El primer dígito a la derecha del punto decimal tiene un valor de 10^{-1} , o de 0.1. En el sistema binario el primer dígito a la derecha del punto binario tiene un valor de 2^{-1} , o 0.5. En general, los principios que se aplican a los números decimales también se aplican a los números en cualquier otro sistema decimal.

El ejemplo 1-1 muestra el número 110.101 en binario (a menudo escrito como 110.101_2). También muestra la potencia y el peso o valor de la posición de cada dígito. Para convertir un número binario en decimal se suman los pesos de cada dígito para formar su equivalente en decimal. El 110.101_2 es equivalente a 6.625 en decimal ($4 + 2 + 0.5 + 0.125$). Observe que ésta es la suma de 2^2 (o 4) + 2^1 (o 2), pero 2^0 (o 1) no se suma ya que no hay dígitos bajo esta posición. La parte fraccionaria está compuesta de 2^{-1} (.5) más 2^{-3} (o .125), pero no hay dígito bajo la posición 2^{-2} (o .25), por lo que no se suma el .25.

EJEMPLO 1-1

Potencia	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
Peso	4	2	1	.5	.25	.125
Número	1	1	0	.1	0	1
Valor numérico	4	+	2	+	0	+.5

$$+ \quad + \quad + \quad + \quad . \quad + \quad .125 = 6.625$$

Suponga que la técnica de conversión se aplica a un número de base 6, tal como 25.2₆. El ejemplo 1-2 muestra a este número colocado bajo las potencias y los pesos de cada posición. En este ejemplo,

hay un 2 bajo 6^1 , el cual tiene un valor de 12_{10} (2×6) y un 5 bajo 6^0 , que tiene un valor de 5 (5×1). La porción del número entero tiene un valor decimal de $12 + 5$, o 17. El número a la derecha del punto hex es un 2 bajo 6^{-1} , que tiene un valor de .333 ($2 \times .167$). Por lo tanto, el número 25.2_6 tiene un valor de 17.333.

EJEMPLO 1-2

Potencia	6^1	6^0	6^{-1}
Peso	6	1	.167
Número	2	5	.2
Valor numérico	$12 + 5 + .333 = 17.333$		

Conversión a decimal

Los ejemplos anteriores demostraron que para convertir de cualquier base numérica a decimal, se determinan los pesos o valores de cada posición del número y luego se suman los pesos para formar el equivalente en decimal. Suponga que el número 125.7_8 octal se convierte a decimal. Para lograr esta conversión, primero anote los pesos de cada posición del número. Éstos aparecen en el ejemplo 1-3. El valor de 125.7_8 es 85.875 decimal, o 1×64 más 2×8 más 5×1 más $7 \times .125$.

EJEMPLO 1-3

Potencia	8^2	8^1	8^0	8^{-1}
Peso	64	8	1	.125
Número	1	2	5	.7
Valor numérico	$64 + 16 + 5 + .875 = 85.875$			

Observe que el peso de la posición a la izquierda de la posición de las unidades es de 8. Esto es 8 por 1. Ahora observe que el peso de la siguiente posición es 64, u 8 por 8. Si existiera otra posición, sería 64 por 8, o 512. Para encontrar el peso de la siguiente posición de mayor orden, multiplique el peso de la posición actual por la base numérica (en este ejemplo, 8). Para calcular los pesos de la posición a la derecha del punto de la base, divida por la base numérica. En el sistema octal, la posición que está inmediatamente a la derecha del punto octal es $1/8$, o 0.125. La siguiente posición es $0.125/8$, o 0.015625, que también puede escribirse como $1/64$. Además observe que el número en el ejemplo 1-3 también puede escribirse como el número decimal $85\frac{7}{8}$.

El ejemplo 1-4 muestra el número binario 11011.0111 escrito con los pesos y las potencias de cada posición. Si se suman estos pesos, el valor del número binario convertido a decimal es 27.4375.

EJEMPLO 1-4

Potencia	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
Peso	16	8	4	2	1	.5	.25	.125	.0625
Número	1	1	0	1	1	.0	1	1	1
Valor numérico	$16 + 8 + 0 + 2 + 1 + 0 + .25 + .125 + .0625 = 27.4375$								

Es interesante observar que 2^{-1} también es $\frac{1}{2}$, 2^{-2} es $\frac{1}{4}$, y así sucesivamente. También es interesante observar que 2^{-4} es $1/16$, o 0.0625. La parte fraccionaria de este número es $7/16$ o 0.4375 decimal. Observe que 0111 es un 7 en código binario para el numerador y el uno más a la derecha está en la posición $1/16$ para el denominador. Otros ejemplos: la fracción binaria de .101 es $5/8$ y la fracción binaria de .001101 es $13/64$.

Los números hexadecimales se utilizan con frecuencia en las computadoras. El número 6A.CH (H de hexadecimal) se muestra con sus pesos en el ejemplo 1-5. La suma de sus dígitos es 106.75, o 106 $\frac{3}{4}$. La parte entera del número está representada con 6×16 más $10(A) \times 1$. La parte fraccionaria es 12 (C) como numerador y $16(16^{-1})$ como denominador, o $12/16$, que se reduce a $\frac{3}{4}$.

EJEMPLO 1-5

Potencia	16^1	16^0	16^{-1}	
Peso	16	1	.0625	
Número	6	A	.	C
Valor numérico	96	+ 10	+ .75	= 106.75

Conversión desde decimal

Las conversiones desde el sistema decimal hacia otros sistemas numéricos son más difíciles de realizar que la conversión a decimal. Para convertir la porción entera de un número desde decimal, se divide entre la base. Para convertir la porción fraccionaria se multiplica por la base.

Conversión de un número entero desde decimal. Para convertir un número entero decimal a otro sistema numérico, se divide por la base y se guardan los residuos como dígitos significativos del resultado. El siguiente es un algoritmo para esta conversión:

1. Dividir el número decimal por la base (base numérica).
2. Guardar el residuo (el primer residuo es el dígito menos significativo).
3. Repetir los pasos 1 y 2 hasta que el cociente sea cero.

Por ejemplo, para convertir el 10 decimal en binario, se divide por 2; el resultado es 5, con un residuo de 0; el primer residuo es la posición de las unidades del resultado (en este ejemplo, un 0); posteriormente se divide el 5 por 2; el resultado es 2, con un residuo de 1; el 1 es el valor de la posición de los dos (2^1). Se continúa la división hasta que el cociente sea cero. El ejemplo 1-6 muestra este proceso de conversión. El resultado se escribe como 1010_2 desde abajo hacia arriba.

EJEMPLO 1-6

2) <u>10</u>	residuo = 0	
2) <u>5</u>	residuo = 1	
2) <u>2</u>	residuo = 0	
2) <u>1</u>	residuo = 1	resultado = 1010
0		

Para convertir el 10 decimal en base 8 se divide por 8, como se muestra en el ejemplo 1-7. El número 10 decimal es 12 octal.

EJEMPLO 1-7

8) <u>10</u>	residuo = 2	
8) <u>1</u>	residuo = 1	resultado = 12
0		

La conversión de decimal a hexadecimal se logra dividiendo por 16. Los residuos variarán en valor, de 0 a 15. Cualquier residuo entre 10 y 15 se convierte después en las letras A a la F para el número hexadecimal. El ejemplo 1-8 muestra el número decimal 109 convertido en 6DH.

EJEMPLO 1-8

16) <u>109</u>	residuo = 13 (D)	
16) <u>6</u>	residuo = 6	resultado = 6D
0		

Conversión desde una fracción decimal. La conversión de una fracción decimal a otra base numérica se logra multiplicando la base. Por ejemplo, para convertir una fracción decimal en un número binario se multiplica por 2. Despues de la multiplicación, la porción entera del número del resultado se guarda como un dígito significativo del resultado y el residuo fraccionario se multiplica de nuevo por la

base. Cuando el residuo de la fracción es cero, termina la multiplicación. Observe que algunos números nunca terminan (cantinela). Es decir, nunca se llega a un residuo de cero. El siguiente es un algoritmo para convertir de una fracción decimal a otro sistema numérico:

1. Multiplicar la fracción decimal por la base (base numérica).
2. Guardar la porción del número entero del resultado (aún si es cero) como un dígito. El primer resultado se escribe inmediatamente a la derecha del punto de la base.
3. Repita los pasos 1 y 2, utilizando la parte fraccionaria del paso 2 hasta que ésta sea cero.

Suponga que el número 0.125 decimal se convierte en binario. Esto se logra multiplicando por 2, como se muestra en el ejemplo 1-9. Observe que la multiplicación continúa hasta que el residuo fraccionario sea cero. Las porciones del número entero se escriben como una fracción binaria (0.001) en este ejemplo.

EJEMPLO 1-9

$$\begin{array}{r}
 & .125 \\
 \times & 2 \\
 \hline
 & 0.25 \quad \text{el dígito es } 0
 \end{array}$$

$$\begin{array}{r}
 & .25 \\
 \times & 2 \\
 \hline
 & 0.5 \quad \text{el dígito es } 0
 \end{array}$$

$$\begin{array}{r}
 & .5 \\
 \times & 2 \\
 \hline
 & 1.0 \quad \text{el dígito es } 1 \qquad \text{resultado} = 0.001_2
 \end{array}$$

Esta misma técnica se utiliza para convertir una fracción decimal en cualquier base numérica. El ejemplo 1-10 muestra la misma fracción decimal de 0.125 del ejemplo 1-9, convertida a octal mediante la multiplicación por 8.

EJEMPLO 1-10

$$\begin{array}{r}
 & .125 \\
 \times & 8 \\
 \hline
 & 1.0 \quad \text{el dígito es } 1 \qquad \text{resultado} = 0.1_8
 \end{array}$$

En el ejemplo 1-11 aparece la conversión a una fracción hexadecimal. Aquí, el número decimal 0.046875 se convierte a hexadecimal multiplicando por 16. Observe que 0.046875 es 0.0CH.

EJEMPLO 1-11

$$\begin{array}{r}
 & .046875 \\
 \times & 16 \\
 \hline
 & 0.75 \quad \text{el dígito es } 0
 \end{array}$$

$$\begin{array}{r}
 & .75 \\
 \times & 16 \\
 \hline
 & 12.0 \quad \text{el dígito es } 12(C) \qquad \text{resultado} = 0.0C_{16}
 \end{array}$$

Hexadecimal codificado en binario

El **hexadecimal codificado en binario** (BCH, por sus siglas en inglés *Binary-code hexadecimal*) se utiliza para representar datos hexadecimales en código binario. Un número hexadecimal codificado en binario es un número hexadecimal escrito de tal forma que cada dígito esté representado por un número binario de 4 bits. Los valores para los dígitos BCH aparecen en la tabla 1-7.

Los números hexadecimales se representan en código BCH mediante la conversión de cada dígito a código BCH con un espacio entre cada dígito codificado. El ejemplo 1-12 muestra el número 2AC convertido a código BCH. Observe que cada dígito BCH está separado por un espacio.

TABLA 1-7 Código hexadecimal codificado en binario (BCH).

Dígito hexadecimal	Código BCH
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

EJEMPLO 1-12

$$2AC = 0010\ 1010\ 1100$$

El propósito del código BCH es permitir que una versión binaria de un número hexadecimal se escriba en un formato que pueda convertirse fácilmente entre BCH y hexadecimal. El ejemplo 1-13 muestra un número codificado en BCH convertido de vuelta a código hexadecimal.

EJEMPLO 1-13

$$1000\ 0011\ 1101 .\ 1110 = 83D.E$$

Complementos

Algunas veces los datos se almacenan en forma de complemento para representar números negativos. Se utilizan dos sistemas para representar datos negativos: complementos a la **base** y complementos a la **base -1**. El primer sistema fue el complemento a la base -1, en el que cada dígito del número se resta de la base -1 para generar el complemento a la base -1, para representar un número negativo.

El ejemplo 1-14 muestra cómo el número binario de 8 bits 01001100 es complementado a uno (base -1) para representarlo como un valor negativo. Observe que a cada dígito del número se le resta uno para generar el complemento a la base -1 (a uno). En este ejemplo el negativo de 01001100 es 10110011. La misma técnica puede aplicarse a cualquier sistema numérico como se muestra en el ejemplo 1-15, en donde el complemento a quince (base -1) del número hexadecimal 5CD se calcula restando 15 a cada dígito.

EJEMPLO 1-14

$$\begin{array}{r} 1111\ 1111 \\ -\ 0100\ 1100 \\ \hline 1011\ 0011 \end{array}$$

EJEMPLO 1-15

$$\begin{array}{r} 15\ 15\ 15 \\ -\ 5\ C\ D \\ \hline A\ 3\ 2 \end{array}$$

En la actualidad el complemento a la base –1 no se utiliza por sí solo; se utiliza como un paso para encontrar el complemento a la base. Este complemento se utiliza para representar números negativos en los sistemas computacionales modernos. (El complemento a la base –1 se utilizaba en los primeros días de la tecnología computacional.) El principal problema con el complemento a la base –1 es que existe un cero negativo o un cero positivo; en el sistema de complemento a la base sólo puede existir un cero positivo.

Para formar el complemento a la base primero hay que encontrar el complemento a la base –1 y después se suma uno al resultado. El ejemplo 1-16 muestra cómo se convierte el número 0100 1000 en un valor negativo, complementándolo a dos (base).

EJEMPLO 1-16

$$\begin{array}{r}
 1111 \ 1111 \\
 - \underline{0100 \ 1000} \\
 1011 \ 0111 \text{ (complemento a uno)} \\
 + \underline{\quad \quad \quad 1} \\
 1011 \ 1000 \text{ (complemento a dos)}
 \end{array}$$

Para demostrar que 0100 1000 es el inverso (negativo) de 1011 1000, sume los dos números para formar un resultado de 8 dígitos. El noveno dígito se descarta y el resultado es cero, ya que 0100 1000 es un 72 positivo, mientras que 1011 1000 es un 72 negativo. La misma técnica se aplica a cualquier sistema numérico. El ejemplo 1-17 muestra cómo se encuentra el inverso del número 345 hexadecimal, primero complementando a 15 el número y después sumando uno al resultado, para formar el complemento a dieciséis. Como antes, si se suma el número original 345 de 3 dígitos al inverso de CBB, el resultado es el número 000 de 3 dígitos. También se descarta el cuarto bit (acarreo). Esto demuestra que 345 es el inverso de CBB. En la siguiente sección se presentará información adicional sobre los complementos a uno y a dos con números con signo.

EJEMPLO 1-17

$$\begin{array}{r}
 15 \ 15 \ 15 \\
 - \underline{3 \ 4 \ 5} \\
 C \ B \ A \text{ (complemento a quince)} \\
 + \underline{\quad \quad \quad 1} \\
 C \ B \ B \text{ (complemento a dieciséis)}
 \end{array}$$

1-4

FORMATOS DE DATOS DE COMPUTADORA

La programación exitosa requiere de un preciso entendimiento de los formatos de los datos. En esta sección describiremos muchos formatos de datos de computadora comunes, según su uso en la familia de microprocesadores Intel. Por lo común los datos aparecen como ASCII, Unicode, BCD, enteros con o sin signo y números de punto flotante (números reales). Hay otros formatos disponibles, pero no los presentaremos aquí porque generalmente no se encuentran.

Datos ASCII y Unicode

Los datos ASCII (**Código estándar estadounidense para el intercambio de información**) representan caracteres alfanuméricos en la memoria de un sistema computacional (vea la tabla 1-8). El código ASCII estándar es un código de 7 bits, en donde el octavo bit (el más significativo) se utiliza para guardar la paridad en algunos sistemas anticuados. Si se utilizan los datos ASCII con una impresora, los bits más significativos son 0 para la impresión alfanumérica y 1 para la impresión de gráficos. En la computadora personal se selecciona un conjunto extendido de caracteres ASCII mediante la colocación de un 1 en el bit más a la izquierda. La tabla 1-9 muestra el conjunto extendido de caracteres ASCII mediante el uso de los códigos 80H-FFH. Los caracteres ASCII extendidos almacenan algunas letras

TABLA 1–8 Código ASCII.

Segundo																		
	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	XA	XB	XC	XD	XE	XF		
Primero	0X	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI	
	1X	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EMS	SUB	ESC	FS	GS	RS	US	
	2X	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/	
	3X	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	
	4X	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
	5X	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_	
	6X	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	
	7X	p	q	r	s	t	u	v	w	x	y	z	{	}	~	⋮⋮⋮	⋮⋮⋮	

TABLA 1-9 Código ASCII extendido, como se imprimen por la impresora IBM ProPrinter.

y signos de puntuación extranjeros, caracteres griegos, caracteres matemáticos, caracteres de dibujo de cuadros y otros caracteres especiales. Tome en cuenta que los caracteres extendidos pueden variar de una impresora a otra. La lista que proporcionamos está diseñada para usarse con la impresora IBM ProPrinter, que también concuerda con el conjunto de caracteres especiales incluido en la mayoría de los procesadores de palabras.

Los caracteres de control ASCII, que también se muestran en la tabla 1-8, realizan las funciones de control en un sistema computacional tales como limpiar pantalla, retroceso, salto de línea, etcétera. Para introducir los códigos de control desde el teclado, mantenga oprimida la tecla Ctrl mientras escribe una letra. Para obtener el código de control 01H, escriba Ctrl-A; el código 02H se obtiene oprimiendo Ctrl-B, y así sucesivamente. Observe que los códigos de control aparecen en la pantalla, desde el símbolo del sistema, como ^A para Ctrl-A, ^B para Ctrl-B, etcétera. Además tenga en cuenta que el código de retorno de carro (CR) es la tecla Intro en la mayoría de los teclados modernos. El propósito de CR es regresar el cursor o la cabeza de impresión al margen izquierdo. Otro código que aparece en muchos programas es el código de salto de línea (LF), el cual desplaza el cursor una línea hacia abajo.

Para utilizar las tablas 1-8 o 1-9 para convertir caracteres alfanuméricos o de control en caracteres ASCII, primero localice el código alfanumérico para la conversión. Después encuentre el primer dígito del código ASCII hexadecimal. Luego encuentre el segundo dígito. Por ejemplo, la letra A mayúscula es el código ASCII 41H, y la letra minúscula es el código ASCII 61H. Muchas aplicaciones basadas en Windows utilizan el sistema **Unicode** para almacenar datos alfanuméricos. Este sistema almacena cada carácter como datos de 16 bits. Los códigos 0000H-0FFFH son los mismos que los del código estándar

ASCII. Los códigos restantes, 0100H-FFFFH, se utilizan para almacenar todos los caracteres especiales de muchos conjuntos de caracteres internacionales. Esto permite que el software escrito para el entorno Windows se utilice en muchos países en el mundo.

Los datos ASCII se almacenan por lo general en la memoria, utilizando una directiva especial para el programa ensamblador llamada definir byte(s), o DB. (El ensamblador es un programa que se utiliza para programar una computadora en su lenguaje máquina nativo.) Una alternativa para DB es la palabra BYTE. En el ejemplo 1-18 se listan las directivas DB y BYTE, junto con varios ejemplos de su uso con cadenas de caracteres en código ASCII. Observe cómo cada cadena de caracteres está encerrada entre comillas (''); nunca utilice la doble comilla (""). También observe que el ensamblador lista el valor en código ASCII para cada carácter a la izquierda de la cadena de caracteres. En el extremo izquierdo está la posición de memoria hexadecimal en donde se almacena primero la cadena de caracteres en el sistema de memoria. Por ejemplo, la cadena de caracteres 'Quien' se almacena empezando en la dirección de memoria 001D. La primera letra se almacena como 51 (Q), seguida de 75 (u), y así sucesivamente. El ejemplo 1-19 muestra las mismas tres cadenas definidas como cadenas de caracteres CString para usarlas con Visual C++. Observe que Visual C++ utiliza comillas dobles para encerrar las cadenas.

EJEMPLO 1-18

0000	42 61 72 72 79	NOMBRES	DB	'Barry B. Brey'
	20 42 2E 20 42			
	72 65 79			
000D	45 6E 20 64 6F	DESORDEN	DB	'En donde estara?'
	6E 64 65 20 65			
	73 74 61 72 61			
	3F			
001D	51 75 69 65 6E	QUIEN	DB	'Quien es primero.'
	20 65 73 20 70			
	72 69 6D 65 72			
	6F 2E			

EJEMPLO 1-19

```
CString NOMBRES = "Barry B. Brey"
CString DESORDEN = "En donde estara?"
CString QUIEN = "Quien es primero."
```

Datos BCD (Decimal codificado en binario)

La información decimal codificada en binario (BCD) se almacena ya sea en formato empaquetado o desempaquetado. Los datos **BCD empaquetados** se almacenan como dos dígitos por byte y los datos **BCD desempaquetados** se almacenan como un dígito por byte. El rango de un dígito BCD se extiende desde 0000_2 hasta 1001_2 , o del 0 al 9 decimal. Los datos BCD desempaquetados se devuelven desde un teclado numérico o teclado convencional. Los datos BCD empaquetados se utilizan para algunas de las instrucciones incluidas para la suma y resta BCD del conjunto de instrucciones del microprocesador.

La tabla 1-10 muestra algunos números decimales convertidos a los formatos BCD empaquetado y BCD desempaquetado. Las aplicaciones que requieren datos BCD son las terminales de punto de

TABLA 1-10 Datos BCD empaquetados y desempaquetados.

Decimal	Empaquetado	Desempaquetado				
12	0001 0010	0000	0001	0000	0010	
623	0000 0110 0010 0011	0000	0110	0000	0010	0000 0011
910	0000 1001 0001 0000	0000	1001	0000	0001	0000 0000

venta y casi cualquier dispositivo que realice una mínima cantidad de operaciones aritméticas simples. Si un sistema requiere de aritmética compleja, es raro utilizar datos BCD ya que no hay un método simple y eficiente para realizar aritmética BCD compleja.

El ejemplo 1-20 muestra cómo utilizar el ensamblador para definir datos BCD tanto empaquetados como desempaquetados. El ejemplo 1-21 muestra cómo hacer esto por medio de Visual C++ y char o bytes. En todos los casos se sigue la convención de almacenar primero los datos menos significativos. Esto significa que para almacenar el número 83 en la memoria, primero se almacena el 3, seguido por el 8. Además tenga en cuenta que con los datos BCD empaquetados la letra H (hexadecimal) va después del número para asegurar que el ensamblador almacene el valor BCD, en vez de un valor decimal para los datos BCD empaquetados. Observe cómo están almacenados los números en memoria como empaquetados, un dígito por byte; o empaquetados, como dos dígitos por byte.

EJEMPLO 1-20

```
Datos BCD desempaquetados (los datos menos significativos primero)
;
0000 03 04 05      NUMB1 DB      3,4,5      ;define el número 543
0003 07 08      NUMB2 DB      7,8      ;define el número 87
;
; Datos BCD empaquetados (los datos menos significativos)
;
0005 37 34      NUMB3 DB      37H,34H      ;define el número 3437
0007 03 45      NUMB4 DB      3,45H      ;define el número 4503
```

EJEMPLO 1-21

```
//Datos BCD desempaquetados (los datos menos significativos primero)
//
char Numb1 = 3,4,5;      ;define el número 543
char Numb2 = 7,8;      ;define el número 87
//
Datos BCD empaquetados (los datos menos significativos primero)
//
char Numb3 = 0x37,0x34      ;define el número 3437
char Numb4 = 3,0x45      ;define el número 4503
```

Datos del tamaño de un byte

Estos datos se almacenan como enteros *con signo* y *sin signo*. La figura 1-14 muestra los formatos con signo y sin signo del entero tamaño byte. La diferencia en estos formatos es el peso de la posición más a

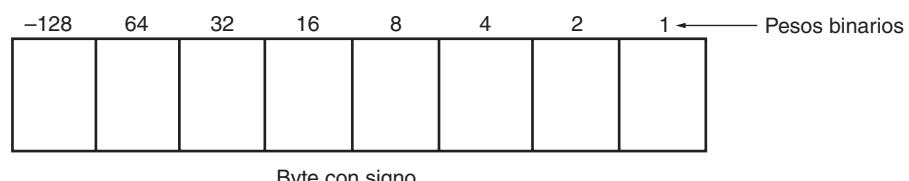
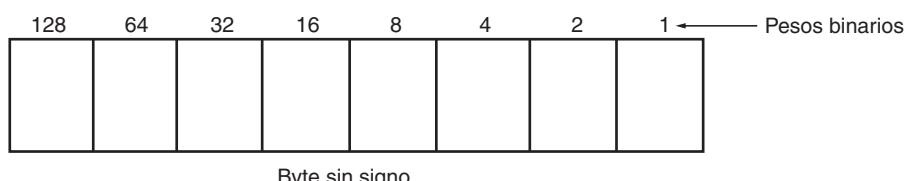


FIGURA 1-14 Los bytes con signo y sin signo muestran los pesos de cada posición de bit binario.

la izquierda. Su valor es 128 para el entero sin signo y menos 128 para el entero con signo. En el formato de entero con signo, el bit más a la izquierda representa el bit de signo del número, así como un peso de menos 128. Por ejemplo, 80H representa un valor de 128 como número sin signo; como número con signo representa un valor de menos 128. El valor de los enteros sin signo varía de 00H a FFH (0-255). El valor de los enteros con signo varía de -128 a 0 y hasta +127.

Aunque los números con signo negativo se representan de esta forma, se almacenan en formato de complemento a dos. El método para evaluar un número con signo mediante el uso de los pesos de cada posición de bit es más sencillo que el acto de complementar un número a dos para encontrar su valor. Esto es muy cierto en el mundo de las calculadoras diseñadas para los programadores.

Siempre que se complementa un número a dos, su signo cambia de negativo a positivo o de positivo a negativo. Por ejemplo, el número 00001000 es un +8. Para encontrar su valor negativo (-8) se complementa el +8 a dos. Para formar un complemento a dos, primero se complementa el número a uno. Para complementar un número a uno, se invierte cada bit del número de cero a uno o de uno a cero. Una vez formado el complemento a uno, se encuentra el complemento a dos al sumar uno al complemento a uno. El ejemplo 1-22 muestra cómo se complementan los números a dos mediante esta técnica.

EJEMPLO 1-22

$$\begin{array}{r}
 +8 = 00001000 \\
 \quad\quad\quad 11110111 \text{ (complemento a uno)} \\
 + \quad\quad\quad \underline{1} \\
 -8 = 11111000 \text{ (complemento a dos)}
 \end{array}$$

Otra técnica (probablemente más sencilla) para complementar un número a dos empieza con el dígito más a la derecha. Se empieza anotando el número de derecha a izquierda. Hay que escribir el número exactamente como aparece hasta el primer uno. Se anota el primer uno y luego se invierten todos los bits a su izquierda. El ejemplo 1-23 muestra esta técnica con el mismo número del ejemplo 1-22.

EJEMPLO 1-23

$$\begin{array}{r}
 +8 = 00001000 \\
 \quad\quad\quad 1000 \text{ (escriba el número hasta el primer 1)} \\
 \quad\quad\quad 1111 \text{ (invierta los bits restantes)} \\
 -8 = 11111000
 \end{array}$$

Para almacenar datos de 8 bits en la memoria mediante el uso del programa ensamblador, utilice la directiva DB como en los ejemplos anteriores, o char como en los ejemplos de Visual C++. El ejemplo 1-24 lista muchos formatos de números de 8 bits almacenados en memoria mediante el programa ensamblador. Observe en el ejemplo que un número hexadecimal se define con la letra H después del número, y que un número decimal se escribe como está, sin nada especial. El ejemplo 1-25 muestra los mismos datos de bytes definidos para usarse con un programa en Visual C++.

EJEMPLO 1-24

```

; Datos sin signo del tamaño de un byte
;
0000 FE    DATOS1 DB      254    ;define el 254 decimal
0001 87    DATOS2 DB      87H    ;define el 87 hexadecimal
0002 47    DATOS3 DB      71     ;define el 71 decimal
;
; Datos con signo del tamaño de un byte
;
0003 9C    DATOS4 DB      -100;   ;define el -100 decimal
0004 64    DATOS5 DB      +100   ;define el +100 decimal
0005 FF    DATOS6 DB      -1     ;define el -1 decimal
0006 38    DATOS7 DB      56     ;define el 56 decimal

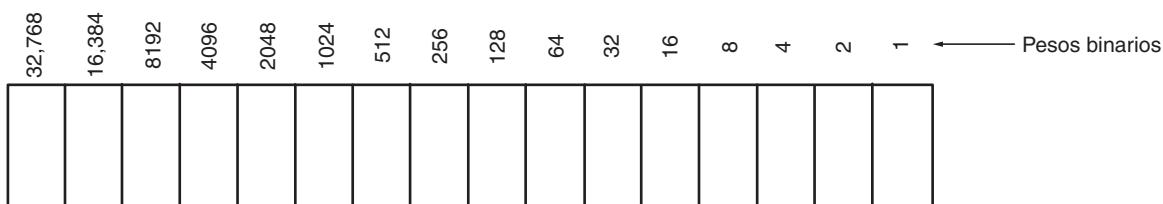
```

EJEMPLO 1-25

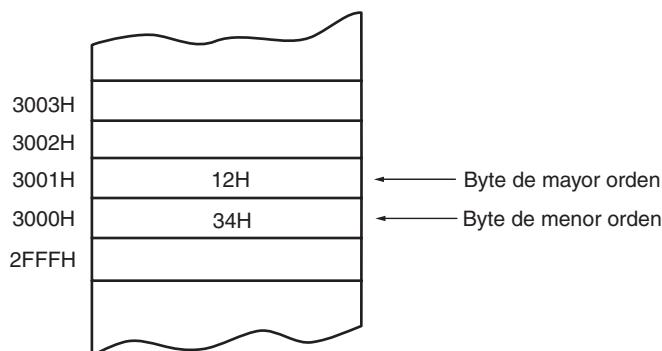
```
// Datos sin signo del tamaño de un byte
//
unsigned char Datos1 = 254;           // define el 254 decimal
unsigned char Datos2 = 0x87;          // define el 87 hexadecimal
unsigned char Datos3 = 71;            // define el 71 decimal
//
// Datos con signo del tamaño de un byte
//
char Datos4 = -100;                 // define el -100 decimal
char Datos5 = +100;                 // define el +100 decimal
char Datos6 = -1;                   // define el -1 decimal
char Datos7 = 56;                   // define el 56 decimal
```

Datos del tamaño de una palabra (word)

Una palabra (16 bits) se forma con dos bytes de datos. El byte menos significativo se almacena siempre en la posición de memoria con la menor numeración, y el byte más significativo se almacena en la de mayor numeración. A este método de almacenar un número se le llama formato **LittleEndian**. Un método alternativo que no se utiliza con la familia de microprocesadores Intel es el formato **BigEndian**. En este formato los números se almacenan de manera que la posición de menor numeración contenga los datos más significativos. El formato Big Endian se utiliza en la familia de microprocesadores Motorola. La figura 1-15(a) muestra los pesos de cada posición de bit en una palabra de datos, y la figura 1-15(b) muestra cómo aparece el número 1234H cuando se almacena en las posiciones de memoria 3000H y 3001H. La única diferencia entre una palabra con signo y una sin signo es la posición del bit más a la izquierda. En el formato sin signo, el bit más a la izquierda no tiene signo y tiene un peso de 32,768; en el formato con signo su peso es de -32,768. Al igual que con los datos con signo del tamaño de un bit, la palabra con signo está en formato de complemento a dos cuando se representa un número negativo. Además, observe que el byte de menor orden está almacenado en la posición de memoria de menor numeración (3000H) y el byte de mayor orden está almacenado en la posición de mayor numeración (3001H).



(a) Palabra sin signo



(b) El contenido de las posiciones de memoria 300H y 3001H es la palabra 1234H.

FIGURA 1-15 El formato de almacenamiento para una palabra de 16 bits en (a) un registro y en (b) dos bytes de memoria.

El ejemplo 1-26 muestra varios datos con signo y sin signo del tamaño de una palabra, almacenados en memoria mediante el programa ensamblador. El ejemplo 1-27 muestra cómo almacenar los mismos números en un programa en Visual C++ (suponiendo que se utiliza la versión 5.0 o más reciente), el cual utiliza la directiva **short** para almacenar un entero de 16 bits. Observe que la directiva **define word(s)** (o **DW**) hace que el ensamblador almacene palabras en la memoria en vez de bytes, como en ejemplos anteriores. También puede usarse la directiva WORD para definir una palabra. Observe que el ensamblador muestra la palabra *datos* en el mismo formato en que se introdujo. Por ejemplo, el ensamblador muestra el número 1000H como 1000. Esto es para conveniencia nuestra, ya que el número en realidad se almacena en la memoria como 00 10 en dos bytes consecutivos de memoria.

EJEMPLO 1-26

```
; Datos sin signo del tamaño de una palabra
;
0000 09F0 DATOS1 DW 2544 ;define el 2544 decimal
0002 87AC DATOS2 DW 87ACH ;define el 87AC hexadecimal
0004 02C6 DATOS3 DW 710 ;define el 710 decimal
;
; Datos con signo del tamaño de una palabra
;
0006 CBA8 DATOS4 DW -13400 ;define el -13400 decimal
0008 00C6 DATOS5 DW +198 ;define el +198 decimal
000A FFFF DATOS6 DW -1 ;define el -1 decimal
```

EJEMPLO 1-27

```
//Datos sin signo del tamaño de una palabra
//
unsigned short Datos1 = 2544; //define el 2544 decimal
unsigned short Datos2= 0x87AC //define el 87AC hexadecimal
unsigned short Datos2= 710; //define el 710 decimal
//
//Datos con signo del tamaño de una palabra
//
short Datos4 = -13400; //define el -13400 decimal
short Datos5 = +198 //define el +198 decimal
short Datos6 = -1 //define el -1 decimal
```

Datos del tamaño de una doble palabra (doubleword)

Este tipo de datos requiere cuatro bytes de memoria, ya que es un número de 32 bits. Los datos tipo doble palabra aparecen como producto después de una multiplicación y también como dividendo antes de una división. En los procesadores 80386 hasta el Pentium 4, la memoria y los registros también son de 32 bits de ancho. La figura 1-16 muestra el formato utilizado para almacenar dobles palabras en la memoria y los pesos binarios de cada posición de bit.

Cuando se almacena una doble palabra en memoria, su byte menos significativo se almacena en la posición de memoria con menor numeración, y su byte más significativo se almacena en la posición de memoria con mayor numeración, mediante el uso del formato Little Endian. Recuerde que esto también es cierto para los datos del tamaño de una palabra. Por ejemplo, el número 12345678H que se almacena en las posiciones de memoria 00100H-00103H se almacena con el 78H en la posición 00100H, el 56H en la posición 00101H, el 34H en la posición 00102H y el 12H en la posición 00103H.

Para definir datos del tamaño de una doble palabra se utiliza la directiva **define doubleword(s)**, o **DD**. (También se puede usar la directiva **DWORD** en vez de **DD**.) El ejemplo 1-28 muestra números con signo y sin signo almacenados en memoria mediante el uso de la directiva **DD**. El ejemplo 1-29 muestra cómo definir las mismas dobles palabras en Visual C++ por medio de la directiva **int**.

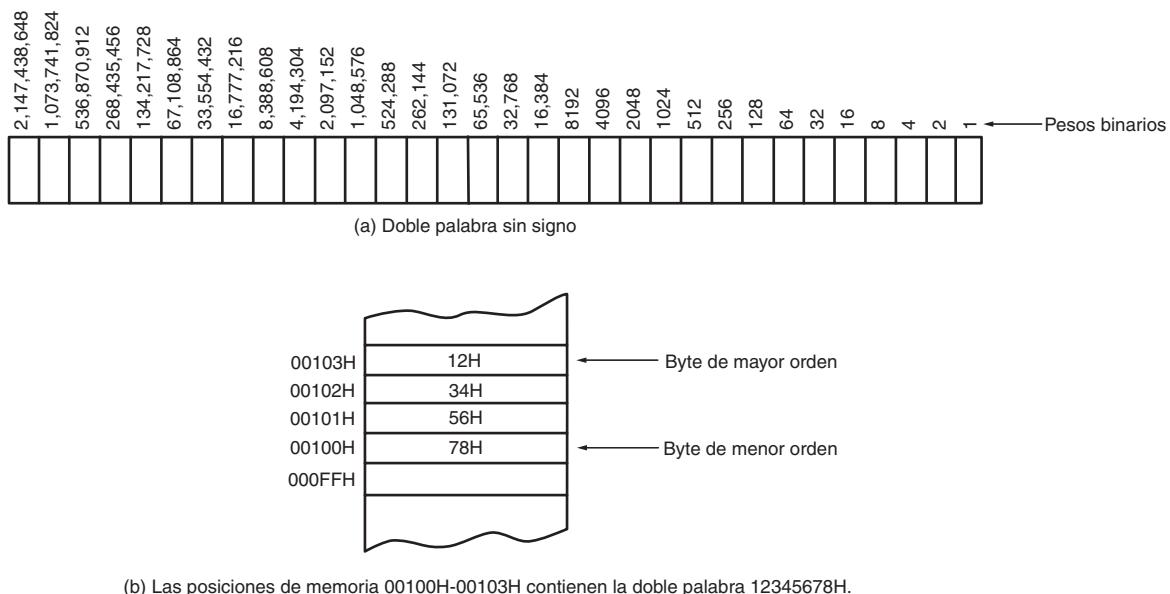


FIGURA 1-16 El formato de almacenamiento para una palabra de 32 bits en (a) un registro y en (b) cuatro bytes de memoria.

EJEMPLO 1-28

```

; Datos sin signo del tamaño de una doble palabra
;
0000 0003E1C0 DATOS1 DD 254400 ;define el 254400 decimal
0004 87AC1234 DATOS2 DD 87AC1234H ;define el 87AC1234 hexadecimal
0008 00000046 DATOS3 DD 70 ;define el 70 decimal
;
; Datos con signo del tamaño de una doble palabra
000C FFEB8058 DATOS4 DD -1343400 ;define el -1343400 decimal
0010 000000C6 DATOS5 DD +198 ;define el +198 decimal
0014 FFFFFFFF DATOS6 DD -1 ;define el -1 decimal

```

EJEMPLO 1-29

```

//Datos sin signo del tamaño de una doble palabra
//
unsigned int Datos1 = 254400; //define el 254400 decimal
unsigned int Datos2 = 0x87AC1234; //define el 87AC1234 hexadecimal
unsigned int Datos3 = 70; //define el 70 decimal
//
//Datos con signo del tamaño de una doble palabra
int Datos4 = -1343400; //define el -1343400 decimal
int Datos5 = +198; //define el +198 decimal
int Datos6 = -1; //define el -1 decimal

```

Los enteros también pueden almacenarse en la memoria de cualquier anchura. Los formatos que se listan aquí son estándar, pero eso no significa que un entero de 256 bytes de anchura no pueda almacenarse en la memoria. El microprocesador es lo suficientemente flexible como para permitir cualquier tamaño de datos en lenguaje ensamblador. Cuando se almacenan números no estándar en memoria, por lo general se utiliza la directiva DB para almacenarlos. Por ejemplo, el número de 24 bits 123456H se almacena mediante el uso de la directiva DB 56H,34H,12H. Observe que esto se apega al formato Little Endian. También se puede utilizar la directiva char en Visual C++.

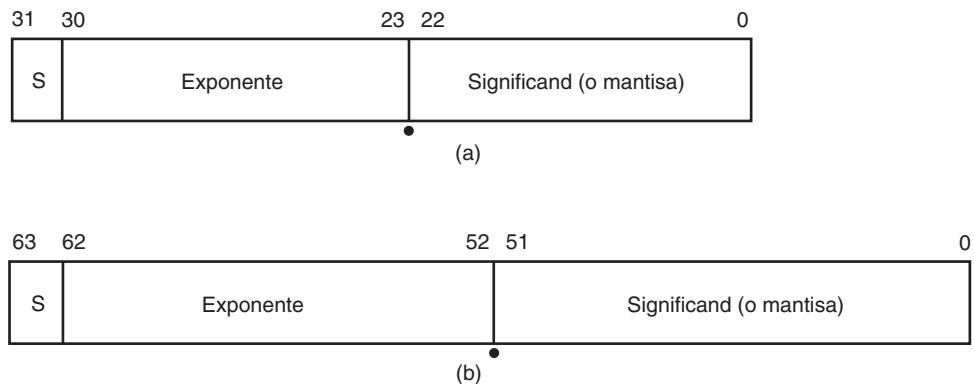


FIGURA 1-17 Los números de punto flotante en (a) precisión simple mediante el uso de una desviación (bias) de 7FH y en (b) doble precisión mediante el uso de una desviación (bias) de 3FFH.

Números reales

Como muchos lenguajes de alto nivel utilizan la familia de microprocesadores Intel, es muy común encontrar números reales. Un número real, o **número de punto flotante**, contiene dos partes: una mantisa, significand o fracción; y un exponente. La figura 1-17 muestra los formatos de 4 y 8 bits de los números reales, como se almacenan en un sistema Intel. Al número de 4 bytes se le llama de **precisión simple** y al formato de 8 bytes se le llama de **doble precisión**. El formato que se presenta aquí es el mismo que especifica el estándar IEEE¹⁰ identificado como IEEE-754, versión 10.0. Este estándar se ha adoptado como el formato estándar de los números reales en casi todos los lenguajes de programación y en muchos paquetes de aplicaciones. El estándar también se aplica a los datos manipulados por el coprocesador numérico en la computadora personal. La figura 1-17(a) muestra un formato de precisión simple que contiene un bit de signo, un exponente de 8 bits y una fracción de 24 bits (mantisa). Como las aplicaciones requieren a menudo de números de punto flotante con doble precisión (vea la figura 1-17(b)), los microprocesadores Pentium-Pentium 4 con su bus de datos de 64 bits realizan transferencias de memoria al doble de la velocidad de los microprocesadores 80386/80486.

La aritmética simple indica que se necesitan 33 bits para almacenar las tres piezas de datos. Esto no es cierto; la mantisa de 24 bits contiene un bit **implícito** (oculto) que le permite representar números de 24 bits aunque sólo se almacenen en 23 bits. El bit oculto es el primer bit del número real normalizado. Al normalizar un número, se ajusta de forma que su valor sea cuando menos 1, pero menos de 2. Por ejemplo, si se convierte el 12 en binario (1100_2), se normaliza y el resultado es 1.1×2^3 . El número entero 1 no se almacena en la porción de la mantisa de 23 bits del número; el 1 es el bit oculto. La tabla 1-11 muestra el formato de precisión simple de este número y de otros.

TABLA 1-11 Números reales de precisión simple.

Decimal	Binario	Normalizado	Signo	Exponente con desviación		Mantisa
+12	1100	1.1×2^3	0	10000010	10000000 00000000 00000000	
-12	1100	1.1×2^3	1	10000010	10000000 00000000 00000000	
+100	1100100	1.1001×2^6	0	10000101	10010000 00000000 00000000	
-1.75	1.11	1.11×2^0	1	01111111	11000000 00000000 00000000	
+0.25	0.01	1.0×2^{-2}	0	01111101	00000000 00000000 00000000	
+0.0	0	0	0	00000000	00000000 00000000 00000000	

¹⁰IEEE es el Institute of Electrical and Electronic Engineers.

El exponente se almacena como un **exponente con desviación**. Con el formato de precisión simple del número real, la desviación es 127 (7FH) y con el formato de doble precisión es 1023 (3FFH). La desviación y el exponente se suman antes de almacenarse en la porción del exponente del número de punto flotante. En el ejemplo anterior hay un exponente de 2^3 , el cual se representa como un exponente con desviación de 127 + 3 o 130 (82H) en el formato de precisión simple, o como 1026 (402H) en el formato de doble precisión.

Hay dos excepciones a las reglas para los números de punto flotante. El número 0.0 se almacena como ceros en todas las posiciones. El número infinito se almacena como unos en el exponente y ceros en la mantisa. El bit de signo indica un infinito positivo o negativo.

Al igual que con otros tipos de datos, puede usarse el ensamblador para definir números reales en formatos de precisión simple o de doble precisión. Como los números de precisión simple son de 32 bits se utiliza la directiva DD o la directiva **define quadword(s)**, o DQ, para definir números reales de 64 bits con doble precisión. Las directivas opcionales para los números reales son REAL4, REAL8 y REAL10 para definir números reales de precisión simple, doble y extendida. El ejemplo 1-30 muestra números definidos en formato de número real para el ensamblador. Si se utiliza el ensamblador en línea de Visual C++, los números de precisión simple se definen como **float** y los números de doble precisión se definen como **double**, según se muestra en el ejemplo 1-31. No hay manera de definir el número de punto flotante con precisión extendida para usarlo en Visual C++.

EJEMPLO 1-30

```
;números reales de precisión simple
;
0000 3F9DF3B6      NUM1      DD      1.234      ;define el 1.234
0004 C1BB3333      NUM2      DD      -23.4      ;define el -23.4
0008 43D20000      NUM3      REAL4     4.2E2      ;define el 420
;
;números reales de doble precisión
000C 405ED999999999A  NUM4      DQ      123.4      ;define el 123.4
0014 C1BB333333333333  NUM5      REAL8     -23.4      ;define el -23.4
;
;números reales de precisión extendida
001C 4005F6CCCCCCCCCCCCD  NUM6      REAL10    123.4      ;define el 123.4
```

EJEMPLO 1-31

```
//Números reales de precisión simple
//
float Num1 = 1.234;
float Num2 = -23.4;
float Num3 = 4.3e2;
//
Números reales de doble precisión
double Num4 = 123.4;
double Num5 = -23.4;
```

1. La era de la computadora mecánica empezó con la llegada del ábaco en el año 500 a.C. Esta primera calculadora mecánica permaneció sin cambio hasta el año 1642, cuando Blaise Pascal la mejoró. Uno de los primeros sistemas computacionales mecánicos fue la Máquina Analítica, desarrollada por Charles Babbage en 1823. Desafortunadamente esta máquina nunca funcionó, debido a la incapacidad de crear las piezas necesarias.
2. La primera máquina calculadora electrónica se desarrolló durante la Segunda Guerra Mundial por Konrad Zuse, uno de los primeros pioneros de la electrónica digital. Su computadora Z3 se utilizó en el diseño de aeronaves y misiles para el ejército alemán.

3. La primera computadora electrónica, que utilizaba tubos al vacío, se puso en operación en 1943 para descifrar los códigos militares de los alemanes. Este primer sistema computacional electrónico llamado Colossus, fue inventado por Alan Turing. Su único problema era que el programa estaba fijo y no podía cambiarse.
4. El primer sistema computacional electrónico programable de propósito general se desarrolló en 1946, en la Universidad de Pensilvania. Esta primera computadora moderna se llamó ENIAC (Calculadora e Integradora Numérica Electrónica).
5. El primer lenguaje de programación de alto nivel, llamado FLOWMATIC, se desarrolló para la computadora UNIVAC I por Grace Hopper a principios de la década de 1950. Este lenguaje ayudó a que surgiera FORTRAN y algunos de los primeros lenguajes de programación, como COBOL.
6. El primer microprocesador en el mundo, el Intel 4004, era un microprocesador de 4 bits (un controlador programable en un chip) que era mediocre según los estándares de la actualidad. Direccionaba tan sólo 4096 posiciones de memoria de 4 bits. Su conjunto de instrucciones contenía solamente 45 instrucciones distintas.
7. Entre los microprocesadores que son comunes actualmente están el 8088 y el 8086, que fueron los primeros microprocesadores de 16 bits. Después les siguieron los microprocesadores 80286, 80386, 80486, Pentium, Pentium Pro, Pentium II, Pentium III y Pentium 4. La arquitectura cambió de 16 bits a 32 bits y, con el Itanium, a 64 bits. Con cada nueva versión se incluían mejoras para aumentar la velocidad y el rendimiento del procesador. De todas las indicaciones, este proceso de mejora de la velocidad y el rendimiento seguirá en efecto, aunque los aumentos en rendimiento tal vez no siempre provengan de un aumento en la frecuencia del reloj.
8. Las computadoras personales basadas en DOS contienen sistemas de memoria que incluyen tres áreas principales: la TPA (área de programa transitorio), el área del sistema y la memoria extendida. La TPA guarda los programas de aplicación, el sistema operativo y los controladores. El área del sistema contiene la memoria utilizada para las tarjetas de visualización de vídeo, las unidades de disco y el ROM del BIOS. El área de memoria extendida está disponible sólo para los microprocesadores del 80286 al Pentium 4 en un sistema de computadora personal estilo AT o ATX. Las computadoras personales basadas en Windows contienen sistemas de memoria que incluyen dos áreas principales: la TPA y el área de sistemas.
9. El microprocesador 8086/8088 direcciona 1 Mbyte de memoria de las posiciones 00000H-FFFFFH. Los microprocesadores 80286 y 80386 direccionan 16 Mbytes de memoria de las posiciones 000000H-FFFFFFFH. El 80386SL direcciona 32 Mbytes de memoria de las posiciones 0000000H-1FFFFFFFH. Los microprocesadores del 80386DX al Pentium 4 direccionan 4 Gbytes de memoria, de las posiciones 00000000H-FFFFFFFFFH. Además, los microprocesadores del Pentium Pro al Pentium 4 pueden operar con una dirección de 36 bits y tienen acceso hasta a 64 Gbytes de memoria, de las posiciones 00000000H-FFFFFFFFFFH.
10. Todas las versiones de los microprocesadores 8086 al Pentium 4 direccionan 64 Kbytes de espacio de direcciones de E/S. Estos puertos de E/S están numerados de 0000H a FFFFH, y los puertos de E/S del 0000H al 03FFH están reservados para el uso del sistema de computadora personal. El bus PCI permite usar los puertos 0400H-FFFFH.
11. El sistema operativo en las primeras computadoras personales era MSDOS (sistema operativo en disco de Microsoft) o PCDOS (sistema operativo en disco de computadora personal de IBM). El sistema operativo realiza la tarea de operar o controlar el sistema computacional, junto con sus dispositivos de E/S. Las computadoras modernas utilizan Microsoft Windows en vez de DOS como sistema operativo.
12. El microprocesador es el elemento controlador en un sistema computacional. Realiza transferencias de datos, realiza operaciones simples de aritmética y de lógica, y realiza decisiones simples. El microprocesador ejecuta programas almacenados en el sistema de memoria para realizar operaciones complejas en períodos de tiempo cortos.
13. Todos los sistemas computacionales contienen tres buses para controlar la memoria y los dispositivos de E/S. El bus de dirección se utiliza para solicitar una posición de memoria o un dispositivo de E/S. El bus de datos transfiere información entre el microprocesador y su memoria o sus espacios de E/S. El bus de control controla la memoria y la E/S, y solicita la lectura o la escritura de datos. El control se logra mediante las conexiones \overline{IORC} (control de lectura de E/S), \overline{IOWC} (control de escritura de E/S), \overline{MRDC} (control de lectura de memoria) y \overline{MWTC} (control de escritura de memoria).

14. Los números se convierten de cualquier base numérica a decimal observando los pesos de cada posición. El peso de la posición a la izquierda del punto de la base es siempre la posición de las unidades en cualquier sistema numérico. La posición a la izquierda de la posición de las unidades es siempre la base por uno. Las siguientes posiciones se determinan multiplicando por la base. El peso de la posición a la derecha del punto de la base siempre se determina dividiendo por la base.
15. La conversión de un número decimal entero a cualquier otra base se logra dividiendo por la base. La conversión de un número decimal fraccionario se logra multiplicando por la base.
16. Los datos hexadecimales se representan en formato hexadecimal o en un código llamado hexadecimal codificado en binario (BCH). Un número hexadecimal codificado en binario se escribe con un número binario de 4 bits que representa a cada dígito hexadecimal.
17. El código ASCII se utiliza para almacenar datos alfanuméricos o numéricos. El código ASCII es un código de 7 bits; puede tener un octavo bit que se utiliza para extender el conjunto de caracteres de 128 a 256. El código de retorno de carro (Intro) regresa la cabeza de impresión o el cursor al margen izquierdo. El código de salto de línea desplaza el cursor o la cabeza de impresión una línea hacia abajo. La mayoría de las aplicaciones modernas utilizan Unicode, el cual contiene el ASCII en los códigos 0000H-00FFH.
18. Los datos decimales codificados en binario (BCD) se utilizan algunas veces en un sistema computacional para almacenar datos decimales. Estos datos se almacenan ya sea en formato empaque-tado (dos dígitos por byte) o desempaque-tado (un dígito por byte).
19. Los datos binarios se almacenan como un byte (8 bits), una palabra (16 bits) o una doble palabra (32 bits) en un sistema computacional. Estos datos pueden ser con signo o sin signo. Los datos con signo negativo siempre se almacenan en el formato de complemento a dos. Los datos de más de 8 bits se almacenan siempre en el formato Little Endian. En Visual C++ de 32 bits, estos datos se representan con char (8 bits), short (16 bits) e int (32 bits).
20. Los datos de punto flotante se utilizan en los sistemas computacionales para almacenar números enteros, mixtos y fraccionarios. Un número de punto flotante está compuesto de un signo, una mantisa y un exponente.
21. Las directivas de ensamblador DB o BYTE definen bytes; DW o WORD definen palabras; DD o DWORD definen dobles palabras; y DQ o QWORD definen palabras cuádruples.

1-6**PREGUNTAS Y PROBLEMAS**

1. ¿Quién desarrolló la Máquina Analítica?
2. En el censo de 1890 se utilizó un nuevo dispositivo llamado tarjeta perforada. ¿Quién desarrolló la tarjeta perforada?
3. ¿Quién fue el fundador de IBM Corporation?
4. ¿Quién desarrolló la primera calculadora electrónica?
5. ¿Para qué propósito se desarrolló el primer sistema computacional electrónico?
6. La primera computadora programable de propósito general se llamó _____.
7. El primer microprocesador en el mundo fue desarrollado en 1971 por _____.
8. ¿Quién era la Condesa de Lovelace?
9. ¿Quién desarrolló el primer lenguaje de programación de alto nivel llamado FLOWMATIC?
10. ¿Qué es una máquina de von Neumann?
11. ¿Qué microprocesador de 8 bits marcó la era del microprocesador?
12. El microprocesador 8085, presentado en 1977, ha vendido _____ copias.
13. ¿Cuál microprocesador Intel fue el primero en direccionar 1 Mbyte de memoria?
14. El 80286 direcciona _____ bytes de memoria.

15. ¿Cuánta memoria hay disponible para el microprocesador 80486?
16. ¿Cuándo introdujo Intel el microprocesador Pentium?
17. ¿Cuándo introdujo Intel el microprocesador Pentium Pro?
18. ¿Cuándo introdujo Intel el microprocesador Pentium 4?
19. ¿Cuáles microprocesadores Intel direccionan 64 G de memoria?
20. ¿Qué significa el acrónimo MIPS?
21. ¿Qué significa el acrónimo CISC?
22. Un bit binario almacena un _____ o un _____.
23. Un K (kilo) en la computadora es igual a _____ bytes.
24. Un M (mega) en la computadora es igual a _____ bytes.
25. Un G (giga) en la computadora es igual a _____ bytes.
26. ¿Cuántas páginas mecanografiadas de información se almacenan en una memoria de 4 Gbytes?
27. El primer 1 Mbyte de memoria en un sistema computacional basado en DOS contiene un área _____ y un área _____.
28. ¿Qué tan grande es el área de programación de aplicaciones Windows?
29. ¿Cuánta memoria hay en el área de programas transitorios del DOS?
30. ¿Cuánta memoria hay en el área de sistemas de Windows?
31. El microprocesador 8086 direcciona _____ bytes de memoria.
32. El microprocesador Pentium 4 direcciona _____ bytes de memoria.
33. ¿Cuáles microprocesadores direccionan 4 Gbytes de memoria?
34. La memoria por arriba del primer 1 Mbyte se llama memoria _____.
35. ¿Qué es el BIOS del sistema?
36. ¿Qué es DOS?
37. ¿Cuál es la diferencia entre un sistema computacional XT y un AT?
38. ¿Qué es el bus local VESA?
39. El bus ISA guarda tarjetas de interfaz de _____ bits.
40. ¿Qué es el USB?
41. ¿Qué es el AGP?
42. ¿Qué es el XMS?
43. Un controlador se almacena en el área _____.
44. El sistema de computadora personal direcciona _____ bytes de espacio de E/S.
45. ¿Cuál es el propósito del BIOS?
46. Dibuje el diagrama de bloques de un sistema computacional.
47. ¿Cuál es el propósito del microprocesador en una computadora basada en microprocesador?
48. Mencione los tres buses que se encuentran en todos los sistemas computacionales.
49. ¿Cuál bus transfiere la dirección de memoria al dispositivo de E/S o a la memoria?
50. ¿Qué señal de control hace que la memoria realice una operación de lectura?
51. ¿Cuál es el propósito de la señal IORC?
52. Si la señal MRDC es un 0 lógico, ¿qué operación realiza el microprocesador?
53. Defina el propósito de las siguientes directivas de ensamblador:
 - (a) DB.
 - (b) DQ.
 - (c) DW.
 - (d) DD.
54. Defina el propósito de las siguientes directivas de Visual C++ de 32 bits:
 - (a) char.
 - (b) short.
 - (c) int.
 - (d) float.
 - (e) double.
55. Convierta los siguientes números binarios a decimal:
 - (a) 1101.01
 - (b) 111001.0011
 - (c) 101011.0101
 - (d) 111.0001

56. Convierta los siguientes números octales a decimal:
- (a) 234.5
 - (b) 12.3
 - (c) 7767.07
 - (d) 123.45
 - (e) 72.72
57. Convierta los siguientes números hexadecimales a decimal:
- (a) A3.3
 - (b) 129.C
 - (c) AC.DC
 - (d) FAB.3
 - (e) BB8.0D
58. Convierta los siguientes enteros decimales a binario, octal y hexadecimal:
- (a) 23
 - (b) 107
 - (c) 1238
 - (d) 92
 - (e) 173
59. Convierta los siguientes números decimales a binario, octal y hexadecimal:
- (a) 0.625
 - (b) .00390625
 - (c) .62890625
 - (d) 0.75
 - (e) .9375
60. Convierta los siguientes números hexadecimales a código hexadecimal codificado en binario (BCH):
- (a) 23
 - (b) AD4
 - (c) 34.AD
 - (d) BD32
 - (e) 234.3
61. Convierta los siguientes números de hexadecimal codificado en binario a hexadecimal:
- (a) 1100 0010
 - (b) 0001 0000 1111 1101
 - (c) 1011 1100
 - (d) 0001 0000
 - (e) 1000 1011 1010
62. Convierta los siguientes números binarios al formato de complemento a uno:
- (a) 1000 1000
 - (b) 0101 1010
 - (c) 0111 0111
 - (d) 1000 0000
63. Convierta los siguientes números binarios al formato de complemento a dos:
- (a) 1000 0001
 - (b) 1010 1100
 - (c) 1010 1111
 - (d) 1000 0000
64. Defina byte, palabra y doble palabra.
65. Convierta las siguientes palabras en cadenas de caracteres en código ASCII:
- (a) RANA.
 - (b) Arco.
 - (c) Agua.
 - (d) Pozo.
66. ¿Cuál es el código ASCII para la tecla Intro y cuál es su propósito?
67. ¿Qué es el Unicode?

68. Use una directiva de ensamblador para almacenar la cadena de caracteres ASCII ‘Que hora es?’ en la memoria.
69. Convierta los siguientes números decimales a números binarios de 8 bits con signo:
 - (a) +32
 - (b) -12
 - (c) +100
 - (d) -92
70. Convierta los siguientes números decimales a palabras binarias con signo:
 - (a) +1000
 - (b) -120
 - (c) +800
 - (d) -3212
71. Use una directiva de ensamblador para almacenar un byte de tamaño 34 en memoria.
72. Cree una variable de tamaño byte llamada Fred1 y almacene el número -34 en ella, en Visual C++.
73. Muestre cómo se almacenan en el sistema de memoria los siguientes números hexadecimales de 16 bits (use el formato estándar Little Endian de Intel):
 - (a) 1234H
 - (b) A122H
 - (c) B100H
74. ¿Cuál es la diferencia entre los formatos Big Endian y Little Endian para almacenar números mayores de 8 bits?
75. Use una directiva de ensamblador para almacenar el número hexadecimal 123A en memoria.
76. Convierta los siguientes números decimales a los formatos BCD empaquetado y desempaquetado:
 - (a) 102
 - (b) 44
 - (c) 301
 - (d) 1000
77. Convierta los siguientes números binarios en números decimales con signo:
 - (a) 10000000
 - (b) 00110011
 - (c) 10010010
 - (d) 10001001
78. Convierta los siguientes números BCD (suponga que son números empaquetados) a números decimales:
 - (a) 10001001
 - (b) 00001001
 - (c) 00110010
 - (d) 00000001
79. Convierta los siguientes números decimales a números de punto flotante con precisión simple:
 - (a) +1.5
 - (b) -10.625
 - (c) +100.25
 - (d) -1200
80. Convierta los siguientes números de punto flotante con precisión simple a números decimales:
 - (a) 0 10000000 11000000000000000000000000000000
 - (b) 1 01111111 00000000000000000000000000000000
 - (c) 0 10000010 10010000000000000000000000000000
81. Use Internet para escribir un informe breve sobre cualquiera de los siguientes pioneros de la computación:
 - (a) Charles Babbage.
 - (b) Konrad Zuse.
 - (c) Joseph Jacquard.
 - (d) Herman Hollerith.

82. Use Internet para escribir un informe breve sobre cualquiera de los siguientes lenguajes computacionales:
 - (a) COBOL.
 - (b) ALGOL.
 - (c) FORTRAN.
 - (d) PASCAL.
83. Use Internet para escribir un informe breve en el que se detallen las características del microprocesador Itanium 2.
84. Use Internet para explicar los detalles de la tecnología de fabricación de 90 nm (nanómetros) de Intel.

CAPÍTULO 2

El microprocesador y su arquitectura

INTRODUCCIÓN

En este capítulo se presenta al microprocesador como un dispositivo programable. Primero se analiza el modelo de programación interno y luego la manera en que se direcciona su espacio de memoria. Se hace una presentación en forma simultánea de la arquitectura de la familia completa de microprocesadores Intel, así como la manera en que los miembros de la familia direccionan el sistema de memoria.

Se describen los modos de direccionamiento de esta poderosa familia de microprocesadores para los modos real y protegido. La memoria de modo real (memoria de DOS) existe en las posiciones 00000H-FFFFFH (el primer 1 Mbyte del sistema de memoria) y está presente en todas las versiones del microprocesador. La memoria de modo protegido (memoria de Windows) existe en cualquier posición de todo el sistema de memoria, pero está disponible sólo para los procesadores del 80286 hasta el Pentium 4, no para el 8086 o el 8088. La memoria de modo protegido para el 80286 contiene 16 Mbytes; para los microprocesadores del 80386 al Pentium contiene 4 Gbytes; y para los microprocesadores del Pentium Pro al Pentium 4 contiene 4 G o 64 Gbytes.

OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

1. Describir la función y el propósito de cada registro visible para un programa en los microprocesadores del 8086 al Pentium 4.
2. Detallar el registro de banderas y el propósito de cada bit de bandera.
3. Describir cómo se accede a la memoria mediante las técnicas de direccionamiento de memoria en modo real.
4. Describir cómo se accede a la memoria mediante las técnicas de direccionamiento de memoria en modo protegido.
5. Describir los registros invisibles para un programa que existen en los microprocesadores del 80286 al Pentium 4.
6. Detallar la operación del mecanismo de paginación de memoria.

2-1

ARQUITECTURA INTERNA DEL MICROPROCESADOR

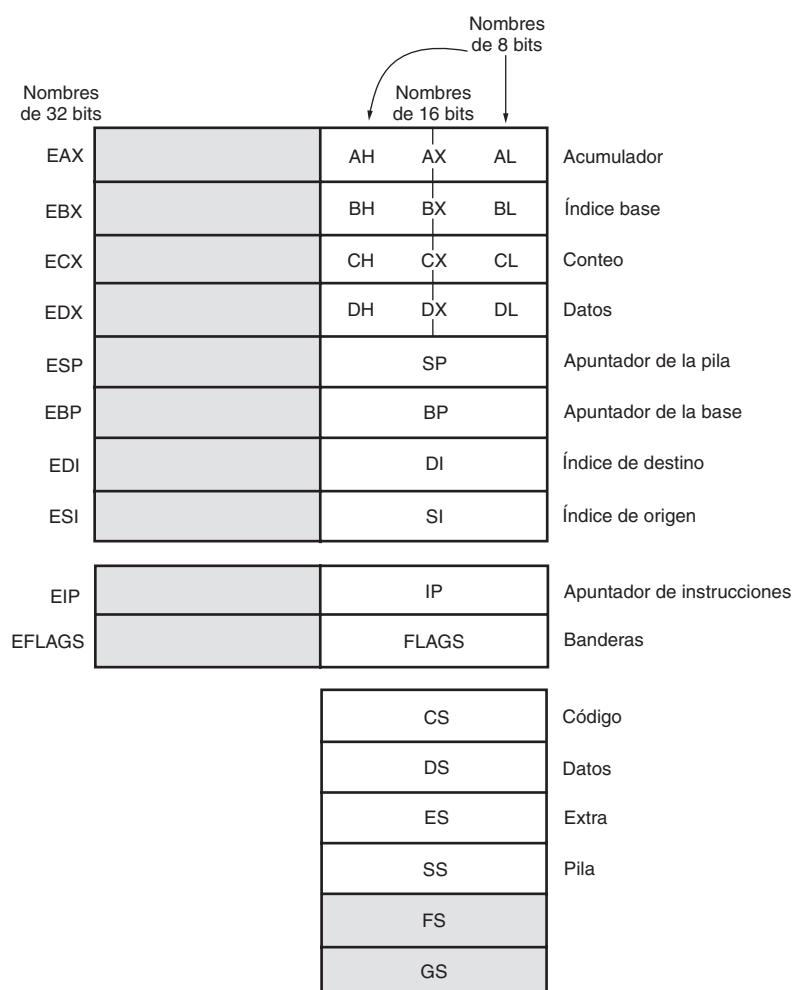
Antes de escribir un programa o de investigar una instrucción, hay que conocer la configuración interna del microprocesador. En esta sección veremos detalladamente la arquitectura interna de los microprocesadores del 8086 al Pentium 4, la cual está visible para los programas. También detallaremos la función y el propósito de cada uno de estos registros internos.

El modelo de programación

El modelo de programación del 8086 al Pentium 4 se considera como **visible para los programas**, ya que sus registros se utilizan durante la programación de aplicaciones y se especifican mediante las instrucciones. Otros registros, que veremos con detalle más adelante en este capítulo, se consideran como **invisibles para los programas** y que no pueden utilizarse en forma directa durante la programación de aplicaciones, pero sí pueden usarse en forma indirecta. Sólo los microprocesadores 80286 y superiores contienen los registros invisibles para los programas que se utilizan para controlar y operar el sistema de memoria protegida y otras características.

La figura 2-1 ilustra el modelo de programación de los microprocesadores del 8086 al Pentium 4. Los primeros microprocesadores 8086, 8088 y 80286 contienen arquitecturas internas de **16 bits**, un subconjunto de los registros mostrados en la figura 2-1. Los microprocesadores del 80386 al Pentium 4 contienen todos, arquitecturas internas de **32 bits**. Las arquitecturas de los microprocesadores del 8086 al 80286 son completamente compatibles con los microprocesadores del 80386 hasta el Pentium 4. Las áreas sombreadas de la figura 2-1 representan registros que no se encuentran en los microprocesadores 8086, 8088 u 80286, ya que son mejoras que se proporcionan en los microprocesadores del 80386 al Pentium 4. En el capítulo 19 podrá consultar la arquitectura de 64 bits de algunos microprocesadores Pentium 4.

FIGURA 2-1
El modelo de
programación de los
microprocesadores
Intel, del 8086 hasta
el Pentium 4.



Observaciones:

- Los registros sombreados sólo existen en los microprocesadores del 80386 hasta el Pentium 4.
- Los registros FS y GS no tienen nombres especiales.

El modelo de programación contiene registros de 8, 16 y 32 bits. Los registros de 8 bits son AH, AL, BH, BL, CH, CL, DH y DL, y se hace referencia a ellos cuando se forma una instrucción en la que se utilizan estas designaciones de dos letras. Por ejemplo, una instrucción ADD AL,AH suma el contenido de 8 bits de AH con el de AL. (Sólo AL cambia debido a esta instrucción.) Los registros de 16 bits son AX, BX, CX, DX, SP, BP, DI, SI, IP, FLAGS, CS, DS, ES, SS, FS y GS. También se hace referencia a estos registros con las designaciones de dos letras. Por ejemplo, una instrucción ADD DX, CX suma el contenido de 16 bits de CX con el de DX. (Sólo DX cambia debido a esta instrucción.) Los registros extendidos de 32 bits son EAX, EBX, ECX, EDX, ESP, EBP, EDI, ESI, EIP y EFLAGS. Estos registros extendidos de 32 bits, junto con los registros de 16 bits FS y GS, están disponibles sólo en el microprocesador 80386 y superiores. Se hace referencia a estos registros mediante las designaciones FS y GS para los dos nuevos registros de 16 bits, y mediante una designación de tres letras para los registros de 32 bits. Por ejemplo, una instrucción ADD ECX, EBX suma el contenido de 32 bits de EBX con el de ECX. (Sólo ECX cambia debido a esta instrucción.)

Algunos registros son de propósito general (o registros multipropósito), mientras que otros tienen propósitos especiales. Los **registros multipropósito** son EAX, EBX, ECX, EDX, EBP, EDI y ESI. Estos registros guardan varios tamaños de datos (bytes, palabras o dobles palabras) y se utilizan para casi cualquier propósito, según lo dicte un programa.

Registros multipropósito

EAX (acumulador)	Se puede hacer referencia a EAX como un registro de 32 bits (EAX), como un registro de 16 bits (AX) o como uno de dos registros de 8 bits (AH y AL). Si se direcciona un registro de 8 o de 16 bits, sólo cambia esa porción del registro de 32 bits sin afectar a los bits restantes. El acumulador se utiliza para instrucciones tales como multiplicación, división y algunas de las instrucciones de ajuste. Para estas instrucciones el acumulador tiene un propósito especial, pero por lo general se considera como un registro multipropósito. En los microprocesadores 80386 y superiores, el registro EAX puede guardar también la dirección de desplazamiento de una posición en el sistema de memoria.
EBX (índice base)	Este índice base puede direccionarse como EBX, BX, BH o BL. Algunas veces el registro BX guarda la dirección de desplazamiento de una posición en el sistema de memoria, en todas las versiones del microprocesador. En el 80386 y superiores, EBX también puede direccionar datos de la memoria.
ECX (Conteo)	ECX es un registro de propósito general que también guarda la cuenta de varias instrucciones. En el 80386 y superiores, el registro ECX también puede guardar la dirección de desplazamiento de datos de la memoria. Las instrucciones que utilizan un conteo son las instrucciones de cadena repetida (REP/REPNE/REPNE); y las instrucciones desplazamiento (shift), rotación (rotate) y LOOP/LOOPD. Las instrucciones desplazamiento y rotación utilizan CL como el conteo, las instrucciones de cadena repetida usan CX y las instrucciones LOOP/LOOPD utilizan CX o ECX.
EDX (Datos)	EDX es un registro de propósito general que guarda una parte del resultado de una multiplicación, o parte del dividendo antes de una división. En el 80386 y superiores, este registro también puede direccionar datos de la memoria.
EBP (apuntador de la base)	EBP apunta a una posición de memoria en todas las versiones del microprocesador para las transferencias de datos de memoria. Este registro se direcciona como BP o EBP.
EDI (índice de destino)	Frecuentemente, EDI direcciona datos de destino de cadena para las instrucciones de cadenas. También funciona como un registro de propósito general de 32 bits (EDI) o de 16 bits (DI).

**ESI
(índice de origen)**

El índice de origen se utiliza como ESI o SI. A menudo el registro de índice de origen dirige datos de cadena de origen para las instrucciones de cadenas. Al igual que EDI, ESI también funciona como registro de propósito general. Se dirige como SI al utilizarlo como registro de 16 bits; se dirige como ESI al utilizarlo como registro de 32 bits.

Registros de propósito especial. Estos registros son EIP, ESP y EFLAGS; y los registros de segmento CS, DS, ES, SS, FS y GS.

**EIP
(apuntador de instrucciones)**

EIP dirige la siguiente instrucción en una sección de memoria definida como segmento de código. Este registro es IP (16 bits) cuando el microprocesador opera en modo real y EIP (32 bits) cuando el 80386 y superiores operan en modo protegido. Los microprocesadores 8086, 8088 y 80286 no contienen un registro EIP, por lo que sólo el 80286 y superiores operan en modo protegido. El apuntador de instrucciones, que apunta a la siguiente instrucción en un programa, se utiliza por el microprocesador para encontrar la siguiente instrucción secuencial en un programa ubicado dentro del segmento de código. El apuntador de instrucciones puede modificarse mediante un salto (jump) o una instrucción de llamada (call).

**ESP
(apuntador de la pila)**

ESP dirige un área de la memoria llamada pila. La memoria de la pila almacena datos a través de este apuntador; explicaremos su funcionamiento más adelante en el texto, con las instrucciones que dirigen los datos de la pila. Se hace referencia a este registro como SP si se utiliza como un registro de 16 bits, y como ESP si se utiliza como registro de 32 bits.

**EFLAGS
(banderas)**

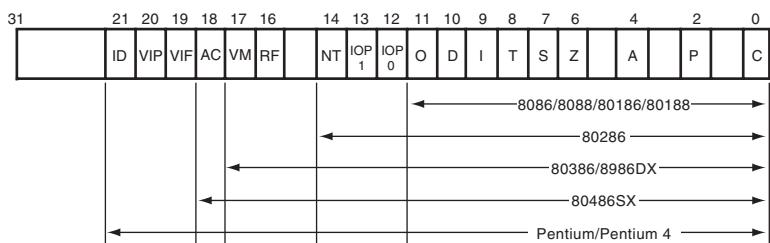
Este registro indica la condición del microprocesador y controla su operación. La figura 2-2 muestra los registros de bandera de todas las versiones del microprocesador. Las banderas son compatibles hacia arriba desde los microprocesadores 8086/8088 hasta el Pentium 4. Los microprocesadores del 8086-80286 contienen un registro FLAG (16 bits) y los microprocesadores 80386 y superiores contienen un registro EFLAG (registro extendido de bandera de 32 bits).

Los cinco bits de bandera de la extrema derecha y la bandera de desbordamiento cambian después de la ejecución de muchas instrucciones aritméticas y lógicas. Las banderas nunca cambian cuando hay transferencia de datos o cuando se realiza una operación de control del programa. Algunas de las banderas se utilizan también para controlar características incluidas en el microprocesador. A continuación se muestra una lista de cada bit de bandera, con una breve descripción de su función. A medida que vayamos presentando instrucciones en los siguientes capítulos, proporcionaremos detalles adicionales sobre los bits de bandera. Las cinco banderas de más a la derecha y la bandera de desbordamiento cambian en base a la mayoría de las operaciones aritméticas y lógicas, aunque las transferencias de datos no les afectan.

C (acarreo)

Éste guarda el valor del acarreo después de la suma, o la sustracción después de la resta. La bandera de acarreo también indica condiciones de error, según lo indiquen algunos programas y procedimientos. Esto es definitivamente cierto en las llamadas a las funciones del DOS.

FIGURA 2-2 El conteo de los registros EFLAG y FLAG para la familia completa de microprocesadores 80X86 y Pentium.



P (paridad)	La paridad es un 0 lógico para paridad impar y un 1 lógico para paridad par. La paridad es el conteo de los unos en un número expresado como par o impar. Por ejemplo, si un número contiene tres bits uno binarios, tiene una paridad impar. Si un número no contiene bits uno, tiene una paridad par. La bandera de paridad tiene poca aplicación en la programación moderna; se implementó en los primeros microprocesadores Intel para comprobar los datos en los entornos de comunicaciones. En la actualidad la comprobación de paridad a menudo la realiza el equipo de comunicaciones de datos, en vez del microprocesador.
A (acarreo auxiliar)	El acarreo auxiliar guarda el acarreo (medio acarreo) después de la suma, o la sustracción después de la resta entre las posiciones de bit 3 y 4 del resultado. Este bit de bandera altamente especializado lo comprueban las instrucciones DAA y DAS para ajustar el valor de AL después de una suma o resta BCD. En cualquier otro caso, ni el microprocesador ni otras instrucciones utilizan este bit de bandera.
Z (cero)	La bandera cero indica que el resultado de una operación aritmética o lógica es cero. Si Z = 1, el resultado es cero; si Z = 0, el resultado no es cero. Esto puede ser confuso, pero es la manera en que Intel decidió nombrar esta bandera.
S (signo)	La bandera de signo guarda el signo aritmético del resultado después de la ejecución de una instrucción aritmética o lógica. Si S = 1, el bit de signo (el bit está más a la izquierda de un número) está activado o es negativo; si S = 0, el bit de signo está desactivado o es positivo.
T (trampa)	La bandera de trampa habilita el atrapamiento a través de una característica de depuración integrada en el chip. (Un programa se depura para encontrar un error o bug.) Si la bandera T está habilitada (1), el microprocesador interrumpe el flujo del programa basándose en las condiciones indicadas por los registros de depuración y los registros de control. Si la bandera T es un 0 lógico, se deshabilita la característica de atrapamiento (depuración). La herramienta de depuración de Visual C++ utiliza la característica de trampa y los registros de depuración para depurar el software con fallas.
I (interrupción)	La bandera de interrupción controla la operación de la terminal de entrada INTR (petición de interrupción). Si I = 1, se habilita la terminal INTR; si I = 0, se deshabilita la terminal INTR. El estado del bit de bandera I se controla mediante las instrucciones STI (establecer bandera I) y CLI (borrar bandera I).
D (dirección)	La bandera de dirección selecciona el modo de incremento o de decremento para los registros DI y/o SI durante las instrucciones de cadena. Si D = 1, los registros se decrementan automáticamente; si D = 0, los registros se incrementan automáticamente. La bandera D se establece con la instrucción STD (establecer dirección) y se borra con la instrucción CLD (borrar dirección).
O (desbordamiento)	Un desbordamiento ocurre cuando se suman o restan números con signo. Un desbordamiento indica que el resultado ha excedido la capacidad de la máquina. Por ejemplo, si se suma 7FH (+128) (usando una suma de 8 bits) con 01H (+1), el resultado es 80H (-128). Este resultado representa una condición de desbordamiento indicada por la bandera de desbordamiento para la suma con signo. En las operaciones sin signo se ignora esta bandera.
IOPL (nivel de privilegio de E/S)	Esta bandera se utiliza en operación de modo protegido para seleccionar el nivel de privilegio para los dispositivos de E/S. Si el nivel de privilegio actual es mayor o de más confianza que el IOPL, la operación de E/S se

ejecuta sin impedimento. Si el IOPL es menor que el nivel de privilegio actual se produce una interrupción, haciendo que se suspenda la ejecución. Hay que tener en cuenta que un IOPL de 00 es el más alto o de mayor confianza, y un IOPL de 11 es el más bajo o de menor confianza.

NT (tarea anidada)

La bandera de tarea anidada indica que la tarea actual está anidada dentro de otra tarea en operación de modo protegido. Esta bandera se establece cuando la tarea se anida mediante software.

RF (continuación)

La bandera de continuación se utiliza con la depuración para controlar la continuación de la ejecución después de la siguiente instrucción.

VM (modo virtual)

El bit de bandera VM selecciona la operación en modo virtual en un sistema de modo protegido. Un sistema de modo virtual permite que coexistan varias particiones de memoria de DOS de 1 Mbyte de longitud en el sistema de memoria. En esencia, esto permite al programa del sistema ejecutar varios programas de DOS. VM se utiliza para simular el DOS en el entorno moderno Windows.

**AC
(comprobación
de alineación)**

El bit de bandera de comprobación de alineación se activa si se direcciona una palabra o doble palabra en un límite que no sea de palabra o de doble palabra. Sólo el microprocesador 80486SX contiene el bit de comprobación de alineación que es utilizado para sincronización principalmente por el coprocesador numérico 80487SX que lo acompaña.

**VIF
(interrupción virtual)**

El VIF es una copia del bit de bandera de interrupción disponible para los microprocesadores del Pentium al Pentium 4.

**VIP (interrupción
virtual pendiente)**

VIP proporciona información sobre una interrupción en modo virtual para los microprocesadores del Pentium al Pentium 4. Se utiliza en entornos multitarea para proporcionar banderas de interrupción virtual al sistema operativo, además de información de interrupciones pendientes.

ID (identificación)

La bandera ID indica que los microprocesadores del Pentium al Pentium 4 soportan la instrucción CPUID. Esta instrucción proporciona información al sistema sobre el microprocesador Pentium, como su número de versión y el fabricante.

Registros de segmento. Los registros adicionales, llamados registros de segmento, generan direcciones de memoria cuando se combinan con otros registros en el microprocesador. Hay de cuatro a seis registros de segmento en diversas versiones del microprocesador. Un registro de segmento funciona en forma distinta en el modo real, cuando se compara con la operación del microprocesador en modo protegido. Más adelante en este capítulo proporcionaremos los detalles sobre su función en modo real y en modo protegido. A continuación se muestra una lista de cada uno de los registros de segmento, junto con su función en el sistema:

CS (código)

El segmento de código es una sección de la memoria que guarda el código (programas y procedimientos) utilizado por el microprocesador. El registro del segmento de código define la dirección inicial de la sección de memoria que guarda el código. En la operación en modo real, define el inicio de una sección de 64 Kbytes de memoria; en modo protegido selecciona un descriptor que describe la dirección inicial y la longitud de una sección de memoria que guarda el código. El segmento de código está limitado a 64 Kbytes en los microprocesadores del 8088 al 80286, y a 4 Gbytes en los microprocesadores 80386 y superiores cuando éstos operan en modo protegido.

DS (datos)

El segmento de datos es una sección de memoria que contiene la mayor parte de los datos utilizados por un programa. Se accede a los datos en el segmento de datos mediante una dirección de desplazamiento o el contenido

de otros registros que guardan la dirección de desplazamiento. Al igual que con el segmento de código y otros segmentos, la longitud está limitada a 64 Kbytes en los microprocesadores del 8086 al 80286, y a 4 Gbytes en los microprocesadores 80386 y superiores.

ES (extra)	El segmento extra es un segmento de datos adicional utilizado por algunas de las instrucciones de cadena para guardar datos de destino.
SS (pila)	El segmento de pila define el área de memoria utilizada para la pila. El punto de entrada de la pila se determina mediante los registros segmento de pila y apuntador de pila. El registro BP también direcciona datos dentro del segmento de pila.
FS y GS	Los segmentos FS y GS son registros de segmento suplementario, disponibles en los microprocesadores del 80386 al Pentium 4 para que los programas puedan acceder a dos segmentos de memoria adicionales. Windows utiliza estos segmentos para operaciones internas, pero no hay disponible una definición de su uso.

2-2

DIRECCIONAMIENTO DE MEMORIA EN MODO REAL

Los microprocesadores 80286 y superiores operan ya sea en modo real o en modo protegido. Sólo el 8086 y el 8088 operan exclusivamente en modo protegido. En esta sección veremos con detalle la operación del microprocesador en el modo real. La **operación en modo real** permite al microprocesador direccionar sólo el primer 1 Mbyte de espacio de memoria; aún si es el microprocesador Pentium 4. A este primer 1 Mbyte de memoria se le llama sistema de **memoria real, memoria convencional o memoria DOS**. El sistema operativo DOS requiere que el microprocesador opere en modo real. Windows no utiliza este modo. La operación en modo real permite que el software de aplicación escrito para los microprocesadores 8086/8088 (que sólo contienen 1 Mbyte de memoria) funcione en los microprocesadores 80286 y superiores, sin necesidad de cambiar el software. La compatibilidad hacia arriba del software es en parte responsable el éxito continuo de la familia de microprocesadores Intel. En todos los casos, cada uno de estos microprocesadores empieza su operación en modo real de manera predeterminada, siempre que se aplica energía o cuando se reinicia el microprocesador.

Segmentos y desplazamientos

Para acceder a una posición de memoria en el modo real se utiliza la combinación de una dirección de segmento y una dirección de desplazamiento. Todas las direcciones de memoria en modo real deben consistir de una dirección de segmento más una dirección de desplazamiento. La **dirección de segmento**, ubicada dentro de uno de los registros de segmento, define la dirección inicial de cualquier segmento de memoria de 64 Kbytes. La **dirección de desplazamiento** selecciona cualquier posición dentro del segmento de memoria de 64 Kbytes. Los segmentos en el modo real siempre tienen una longitud

FIGURA 2-3 El esquema de direccionamiento de memoria en modo real, en donde se utiliza una dirección de segmento más un desplazamiento.

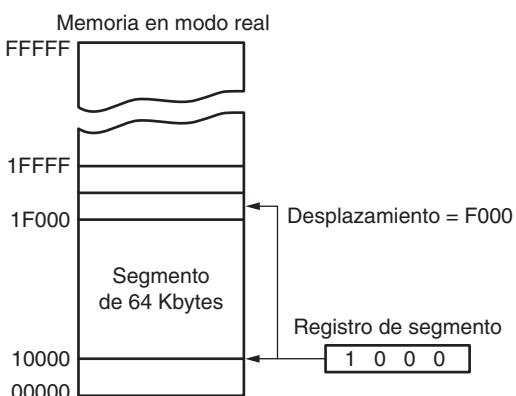


TABLA 2-1 Ejemplo de direcciones de segmento en modo real.

Registro de segmento	Dirección inicial	Dirección final
2000H	20000H	2FFFFH
2001H	20010H	3000FH
2100H	21000H	30FFFH
AB00H	AB000H	BAFFFH
1234H	12340H	2233FH

de 64 Kbytes. La figura 2-3 muestra cómo el esquema de direccionamiento tipo **segmento más desplazamiento** selecciona una posición de memoria. Esta ilustración muestra un segmento de memoria que empieza en la posición 10000H y termina en la posición 1FFFFH; 64 Kbytes de longitud. También muestra cómo una dirección de desplazamiento (algunas veces conocida como **desplazamiento**) de F000H selecciona la posición 1F000H en el sistema de memoria. El desplazamiento es la distancia por encima del inicio del segmento, como se muestra en la figura 2-3.

El registro de segmento en la figura 2-3 contiene 1000H y aún así dirige a un segmento inicial en la posición 10000H. En el modo real, a cada registro de segmento se le adjunta de forma interna un **0H** en su extremo más a la derecha. Con esto se forma una dirección de memoria de 20 bits, lo cual le permite acceder al inicio de un segmento. El microprocesador debe generar una dirección de memoria de 20 bits para acceder a una posición dentro del primer 1 M de memoria. Por ejemplo, cuando un registro de segmento contiene 1200H, dirige a un segmento de memoria de 64 Kbytes que comienza en la posición 12000H. De igual forma, si un registro de segmento contiene 1201H, dirige a un segmento de memoria que comienza en la posición 12010H. Debido al 0H que se adjunta de forma interna, los segmentos de modo real pueden comenzar sólo dentro de un límite de 16 bits en el sistema de memoria. A este límite de 16 bits se le llama **párrafo**.

Como un segmento de memoria en modo real es de 64 K de longitud, una vez que se conoce la dirección inicial se encuentra la **dirección final** si se le suma FFFFH. Por ejemplo, si un registro de segmento contiene 3000H, la primera dirección del segmento es 30000H y la última dirección es 30000H + FFFFH, o 3FFFFH. La tabla 2-1 muestra varios ejemplos del contenido del registro de segmento y las direcciones inicial y final de los segmentos de memoria seleccionados por cada dirección de segmento.

La dirección de desplazamiento, que forma parte de la dirección, se suma al inicio del segmento para dirigir a una posición de memoria dentro del segmento de memoria. Por ejemplo, si la dirección de segmento es 1000H y la dirección de desplazamiento es 2000H, el microprocesador dirige a la posición de memoria 12000H. La dirección de desplazamiento siempre se suma a la dirección inicial del segmento para localizar los datos. La dirección de segmento y desplazamiento algunas veces se escribe como 1000:2000 para representar una dirección de segmento de 1000H, con un desplazamiento de 2000H.

En el microprocesador 80286 (mediante circuitos externos especiales) y en los microprocesadores del 80386 al Pentium 4, puede dirigirse una cantidad extra de 64 K menos 16 bytes de memoria cuando la dirección de segmento es FFFFH y está instalado el controlador HIMEM.SYS para DOS en el sistema. Esta área de memoria (0FFFF0H–10FFEH) se conoce como **memoria alta**. Cuando se genera una dirección utilizando una dirección de segmento de FFFFH, se habilita la terminal de dirección A20 (si está soportada) cuando se suma un desplazamiento. Por ejemplo, si la dirección de segmento es FFFFH y la dirección de desplazamiento es 4000H, la máquina dirige a la posición de memoria FFFF0H + 4000H, o 103FF0H. Observe que la línea de dirección A20 es la que se encuentra en la dirección 103FF0H. Si no se soporta la línea A20, la dirección se genera como 03FF0H ya que A20 permanece como cero lógico.

Algunos modos de direccionamiento combinan más de un registro y un valor de desplazamiento para formar una dirección de desplazamiento. Cuando esto ocurre, la suma de estos valores podría exceder de FFFFH. Por ejemplo, la dirección a la que se accede en un segmento cuya dirección de segmento es 4000H y cuya dirección de desplazamiento se especifica como la suma de F000H más 3000H, accederá a la posición de memoria 42000H en vez de la posición 52000H. Al sumar F000H y 3000H forman una suma de 16 bits (**módulo 16**) de 2000H, que se utiliza como la dirección de desplazamiento, y no 12000H, la verdadera suma. El acarreo de 1 (F000H + 3000H = 12000H) se descarta en esta suma para formar la dirección de desplazamiento de 2000H. La dirección se genera como 4000:2000, o 42000H.

TABLA 2-2

Combinaciones predeterminadas de segmento y desplazamiento de 16 bits.

<i>Segmento</i>	<i>Desplazamiento</i>	<i>Propósito especial</i>
CS	IP	Dirección de la instrucción
SS	SP o BP	Dirección de la pila
DS	BX, DI, SI, un número de 8 o de 16 bits	Dirección de los datos
ES	DI para instrucciones de cadena	Dirección de destino de la cadena

Registros de segmento y de desplazamiento predeterminados

El microprocesador tiene un conjunto de reglas que se aplican a los segmentos, siempre que se direcciona la memoria. Estas reglas, que se aplican en modo real y en modo protegido, definen la combinación del registro de segmento y del registro de desplazamiento. Por ejemplo, el registro de segmento de código siempre se utiliza con el apuntador de instrucciones para direccionar la siguiente instrucción en un programa. Esta combinación es **CS:IP** o **CS:EIP**, dependiendo del modo de operación del microprocesador. El registro de **segmento de código** define el inicio del segmento de código y el **apuntador de instrucciones** localiza la siguiente instrucción dentro del segmento de código. Esta combinación (CS:IP o CS:EIP) localiza la siguiente instrucción que va a ejecutar el microprocesador. Por ejemplo, si CS = 1400H e IP/EIP = 1200H, el microprocesador obtiene su siguiente instrucción de la posición de memoria 14000H + 1200H, o 15200H.

Otra de las combinaciones predeterminadas es la **pila**. Se hace referencia a los datos de la pila a través del segmento de pila en la posición de memoria direccionada por el apuntador de la pila (SP/ESP) o por el apuntador (BP/EBP). Se hace referencia a estas combinaciones como SS:SP (SS:ESP) o SS:BP (SS:EBP). Por ejemplo, si SS = 2000H y BP = 3000H, el microprocesador direcciona la posición de memoria 23000H para la posición de memoria del segmento de la pila. En modo real sólo los 16 bits más a la derecha de la dirección del registro extendido direccionan una posición dentro del segmento de memoria. En los microprocesadores del 80386 al Pentium 4 nunca se debe colocar un número mayor de FFFFH en un registro de desplazamiento si el microprocesador se opera en modo real. Esto hace que el sistema se detenga y que indique un error de direccionamiento.

En la tabla 2-2 se muestran otras combinaciones predeterminadas para direccionar memoria mediante un microprocesador Intel con registros de 16 bits. La tabla 2-3 muestra las combinaciones predeterminadas que se asumen en los microprocesadores 80386 y superiores, los cuales utilizan registros de 32 bits. Estos microprocesadores tienen una mayor selección de combinaciones de dirección de segmento/desplazamiento que los microprocesadores del 8086 al 80286.

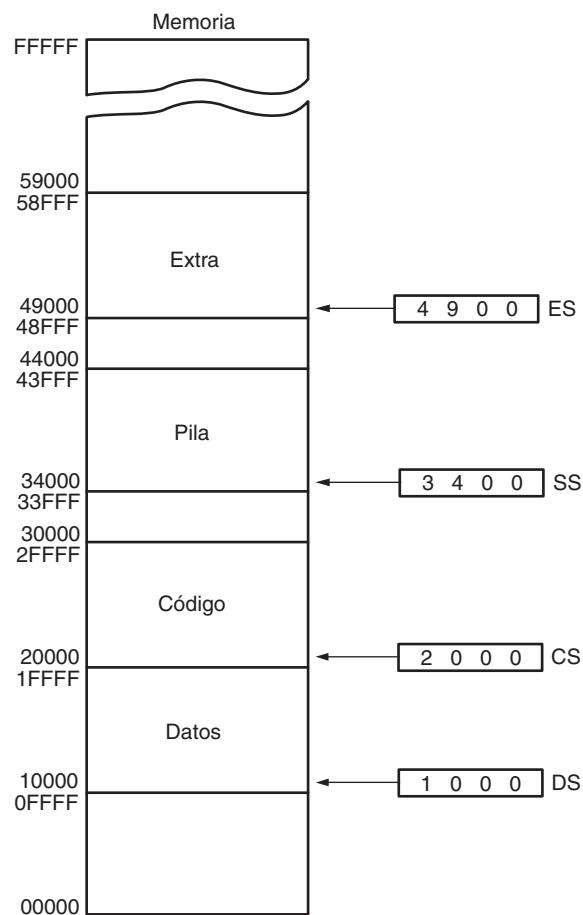
Los microprocesadores del 8086 al 80286 permiten cuatro segmentos de memoria, y los microprocesadores del 80386 al Pentium 4 permiten seis. La figura 2-4 muestra un sistema que contiene cuatro segmentos de memoria. Hay que tener en cuenta que un segmento de memoria puede invadir

TABLA 2-3

Combinaciones predeterminadas de segmento y desplazamiento de 32 bits.

<i>Segmento</i>	<i>Desplazamiento</i>	<i>Propósito especial</i>
CS	EIP	Dirección de la instrucción
SS	ESP o EBP	Dirección de la pila
DS	EAX, EBX, ECX, EDX, ESI, EDI, un número de 8 o de 32 bits	Dirección de los datos
ES	EDI para las instrucciones de cadena	Dirección de destino de cadena
FS	No hay predeterminado	Dirección general
GS	No hay predeterminado	Dirección general

FIGURA 2-4 Un sistema de memoria que muestra la colocación de cuatro segmentos de memoria.



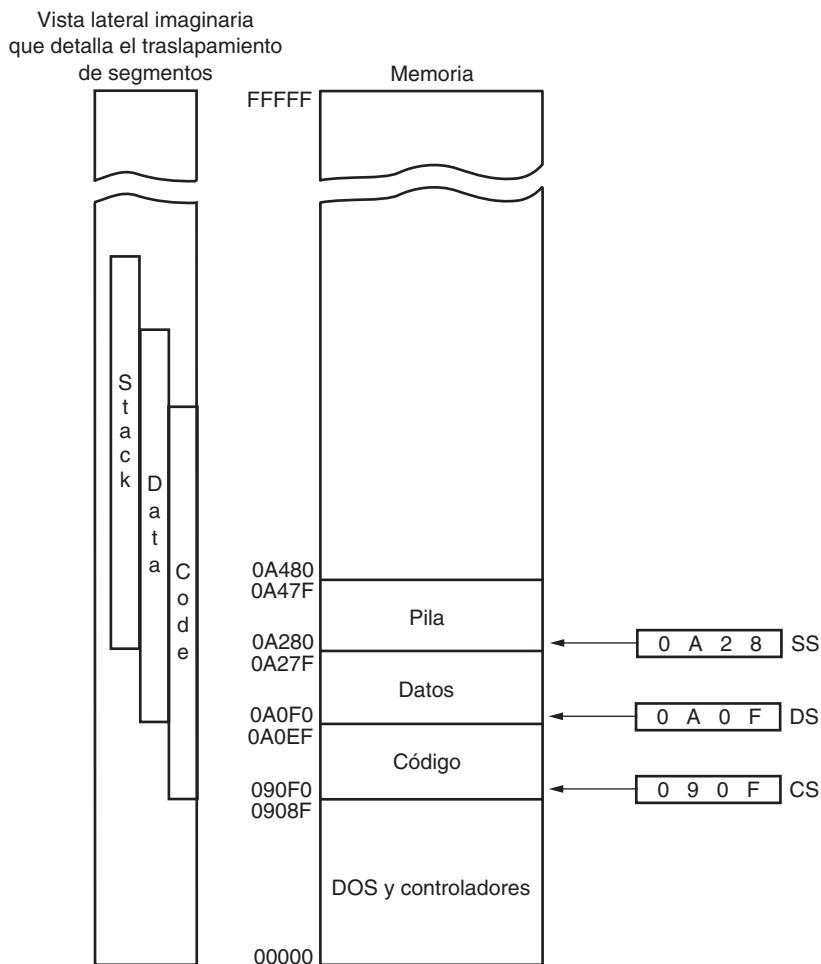
o incluso traslaparse con otro, si no se requieren 64 Kbytes de memoria para un segmento. Piense en los segmentos como si fueran ventanas que pueden desplazarse sobre cualquier área de memoria para acceder a los datos o al código. Además, un programa puede tener más de cuatro o de seis segmentos, pero sólo puede acceder a cuatro o seis segmentos a la vez.

Suponga que un programa de aplicación requiere 1000H bytes de memoria para su código, 190H bytes de memoria para sus datos y 200H bytes de memoria para su pila. Esta aplicación no requiere un segmento extra. Cuando el DOS coloca este programa en el sistema de memoria, se carga en la TPA en la primer área disponible de memoria por encima de los controladores y otros programas de la TPA. Esta área se indica mediante un **apuntador libre** que es mantenido por el DOS. La carga del programa se maneja automáticamente mediante el **cargador de programas** que se encuentra dentro del DOS. La figura 2-5 muestra cómo se almacena una aplicación en el sistema de memoria. Los segmentos muestran un traslapamiento ya que no se requieren 64 Kbytes de memoria para la cantidad de datos en ellos. La vista lateral de los segmentos muestra claramente el traslapamiento. También muestra cómo pueden desplazarse los segmentos sobre cualquier área de la memoria, si se cambia la dirección inicial del segmento. Afortunadamente, el cargador de programas del DOS calcula y asigna las direcciones iniciales de los segmentos.

El esquema de direccionamiento mediante segmento y desplazamiento permite la reubicación

El esquema de direccionamiento mediante segmento y desplazamiento parece excesivamente complicado. En realidad es complicado, pero también ofrece una ventaja para el sistema. Este complicado esquema de direccionamiento de segmento más desplazamiento permite reasignar los programas DOS en

FIGURA 2-5 Un programa de aplicación que contiene un segmento de código, uno de datos y uno de pila, cargado en la memoria de un sistema DOS.



el sistema de memoria. También permite que los programas escritos para funcionar en modo real operen en un sistema en modo protegido. Un **programa reubicable** puede colocarse en cualquier área de memoria, y puede utilizarse sin necesidad de hacerle cambios. El esquema de direccionamiento mediante segmento y desplazamiento permite la reubicación de programas y datos sin necesidad de hacer cambios. Esto es ideal para usarse en un sistema computacional de propósito general, en el que no todos los equipos contienen las mismas áreas de memoria. La estructura de memoria de la computadora personal es distinta dependiendo del equipo, por lo que se necesita usar datos y software que puedan reubicarse.

Como la memoria se direcciona dentro de un segmento mediante una dirección de desplazamiento, el segmento de memoria puede moverse a cualquier lugar del sistema de memoria sin necesidad de cambiar las direcciones de desplazamiento. Esto se logra mediante el desplazamiento del programa completo, como un bloque, hacia una nueva área y después se cambia sólo el contenido de los registros de segmento. Si una dirección está cuatro bytes por encima del inicio del segmento, su dirección de desplazamiento es de 4. Si todo el programa se desplaza hacia una nueva área de memoria, esta dirección de desplazamiento de 4 sigue apuntando a cuatro bytes por encima del inicio del segmento. Sólo hay que cambiar el contenido del registro de segmento para direccionar el programa en la nueva área de memoria. Sin esta característica, un programa tendría que reescribirse o alterarse exhaustivamente antes de poder moverlo. Para ello se requeriría de tiempo adicional o de muchas versiones del programa para las distintas configuraciones de sistemas computacionales. Este concepto también se aplica a los programas escritos para ejecutarse en el modo protegido para Windows. En el entorno Windows todos los programas se escriben suponiendo que los primeros 2 G de memoria están disponibles para código y datos. Cuando el programa se carga, se coloca en la memoria actual, que podría estar en cualquier parte, y una porción podría estar localizada en el disco, en forma de archivo de intercambio.

2-3

INTRODUCCIÓN AL DIRECCIONAMIENTO DE MEMORIA EN MODO PROTEGIDO

Este método de direccionamiento (disponible para el 80286 y superiores) permite el acceso a los datos y programas localizados por encima del primer 1 Mbyte de memoria, así como los que están dentro del primer 1 Mbyte de memoria. El **modo protegido** es en el que opera Windows. Para direccionar esta sección extendida del sistema de memoria se requiere un cambio en el esquema de direccionamiento mediante un segmento más un desplazamiento que se utiliza con el direccionamiento de memoria en modo real. Cuando los datos y los programas se direccionan en memoria extendida, la dirección de desplazamiento se sigue utilizando para acceder a la información localizada dentro del segmento de memoria. Una diferencia es que la dirección de segmento, según lo visto con el direccionamiento de memoria en modo real, ya no está presente en el modo protegido. En lugar de la dirección de segmento, el registro de segmento contiene un **selector** que escoge un descriptor de una tabla de descriptores. El **descriptor** presenta la posición, longitud y derechos de acceso del segmento de memoria. Como el registro de segmento y la dirección de desplazamiento siguen accediendo a la memoria, las instrucciones en modo protegido son idénticas a las instrucciones en modo real. De hecho, la mayoría de los programas escritos para funcionar en el modo real funcionarán sin cambios en el modo protegido. La diferencia entre los modos es la manera en que el microprocesador interpreta el registro de segmento para acceder al segmento de memoria. Otra diferencia en los microprocesadores 80386 y superiores es que la dirección de desplazamiento puede ser de 32 bits, en vez de 16 bits en modo protegido. Una dirección de desplazamiento de 32 bits permite que el microprocesador acceda a los datos dentro de un segmento que puede tener hasta 4 Gbytes de longitud.

Selectores y descriptores

El selector, que se encuentra en el registro de segmento, selecciona uno de 8192 descriptores de una o dos tablas de descriptores. El **descriptor** presenta la posición, longitud y derechos de acceso del segmento de memoria. El registro de segmento aún selecciona un segmento de memoria (aunque de forma indirecta), pero no de forma directa como en el modo real. Por ejemplo, en el modo real si CS = 0008H, el segmento de código empieza en la posición 00080H. En el modo protegido, este número de segmento puede direccionar cualquier posición de memoria en todo el sistema para el segmento de código, como explicaremos en breve.

Hay dos tablas de descriptores que se utilizan con los registros de segmento: una contiene descriptores globales y la otra contiene descriptores locales. Los **descriptores globales** contienen definiciones de segmentos que se aplican a todo los programas, mientras que los **descriptores locales** son por lo general únicos para una aplicación. Al descriptor global lo podemos llamar **descriptor de sistema** y al descriptor local lo podemos llamar **descriptor de aplicación**. Cada tabla contiene 8192 descriptores, por lo que hay un total de 16,384 descriptores disponibles para una aplicación en un momento dado. Como el descriptor describe un segmento de memoria, esto permite que se describan hasta 16,384 segmentos de memoria para cada aplicación. Como un segmento de memoria puede tener hasta 4 Gbytes de longitud, esto significa que una aplicación podría tener acceso a $4\text{ G} \times 16,384$ bytes de memoria, o 64 Tbytes.

Descriptor del 80286		Descriptor del 80386 al Pentium 4	
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	Base (B31–B24)	G D 0 A Límite (L19–L16)
Derechos de acceso	Base (B23–B16)	Derechos de acceso	Base (B23–B16)
Base (B15–B0)		Base (B15–B0)	
Límite (L15–L0)		Límite (L15–L0)	

FIGURA 2-6 Los formatos de descriptores para los microprocesadores 80286 y del 80386 al Pentium 4.

La figura 2-6 muestra el formato de un descriptor para los microprocesadores del 80286 al Pentium 4. Observe que cada descriptor es de ocho bytes de longitud, por lo que las tablas de descriptores globales y de descriptores locales tienen cada una un máximo de 64 Kbytes de longitud. Los descriptores para el microprocesador 80286 y para los microprocesadores del 80386 al Pentium 4 difieren ligeramente, pero el descriptor del 80286 es compatible con los más recientes.

La porción de **dirección base** del descriptor indica la posición inicial del segmento de memoria. Para el microprocesador 80286 la dirección base es de 24 bits, por lo que los segmentos empiezan en cualquier posición de sus 16 Mbytes de memoria. La limitación en cuanto al límite de los párrafos se elimina en estos microprocesadores cuando se operan en modo protegido, por lo que los segmentos pueden comenzar en cualquier dirección. Los microprocesadores 80386 y superiores pueden utilizar una dirección base de 32 bits que permite que los segmentos inicien en cualquier posición dentro de sus 4 Gbytes de memoria. Observe cómo la dirección base del descriptor del 80286 es compatible hacia arriba con el descriptor de los microprocesadores 80386 al Pentium 4, ya que sus 16 bits más significativos son 0000H. Consulte los capítulos 18 y 19 para ver los detalles adicionales sobre el espacio de memoria de 64 G que proporcionan los microprocesadores del Pentium Pro al Pentium 4.

El límite de segmento contiene la última dirección de desplazamiento de un segmento. Por ejemplo, si un segmento comienza en la posición de memoria F00000H y termina en la posición F000FFH, la dirección base es F00000H y el límite es FFH. Para el microprocesador 80286, la dirección base es F00000H y el límite es 00FH. Para los microprocesadores 80386 y superiores, la dirección base es 00F00000H y el límite es 000FFH. Observe que el microprocesador 80286 tiene un límite de 16 bits y los microprocesadores del 80386 al Pentium 4 tienen un límite de 20 bits. Un 80286 accede a los segmentos de memoria que tienen entre 1 y 64 Kbytes de longitud. El 80386 y superiores acceden a los segmentos de memoria que tienen entre 1 y 1 Mbyte, o entre 4 K y 4 Gbytes de longitud.

Hay otra característica incluida en el descriptor de los microprocesadores del 80386 al Pentium 4 que no se encuentra en el descriptor del 80286: el bit G, o **bit de granularidad**. Si G = 0, el límite especifica un límite de segmento de 00000H a FFFFFH. Si G = 1, el valor del límite se multiplica por 4 Kbytes (se adjunta a FFFFH). El límite sería entonces de 00000FFFFH a FFFFFFFFH si G = 1. Esto permite una longitud de segmento de 4 K a 4 Gbytes, en intervalos de 4 bytes. La razón por la que la longitud de segmento es de 64 Kbytes en el 80286 es que la dirección de desplazamiento siempre es de 16 bits, debido a su arquitectura interna de 16 bits. Los microprocesadores 80386 y superiores utilizan una arquitectura de 32 bits que permite una dirección de desplazamiento (en la operación en modo protegido) de 32 bits. Esta dirección de desplazamiento de 32 bits permite longitudes de segmento de 4 Gbytes y la dirección de desplazamiento de 16 bits permite longitudes de segmento de 64 Kbytes. Los sistemas operativos funcionan en un entorno de 16 o de 32 bits. Por ejemplo, DOS utiliza un entorno de 16 bits, mientras que la mayoría de las aplicaciones Windows utilizan un entorno de 32 bits llamado **WIN32**.

El ejemplo 2-1 muestra cómo empieza y termina el segmento si la dirección base es 10000000H, el límite es 001FFH y el bit G = 0.

EJEMPLO 2-1

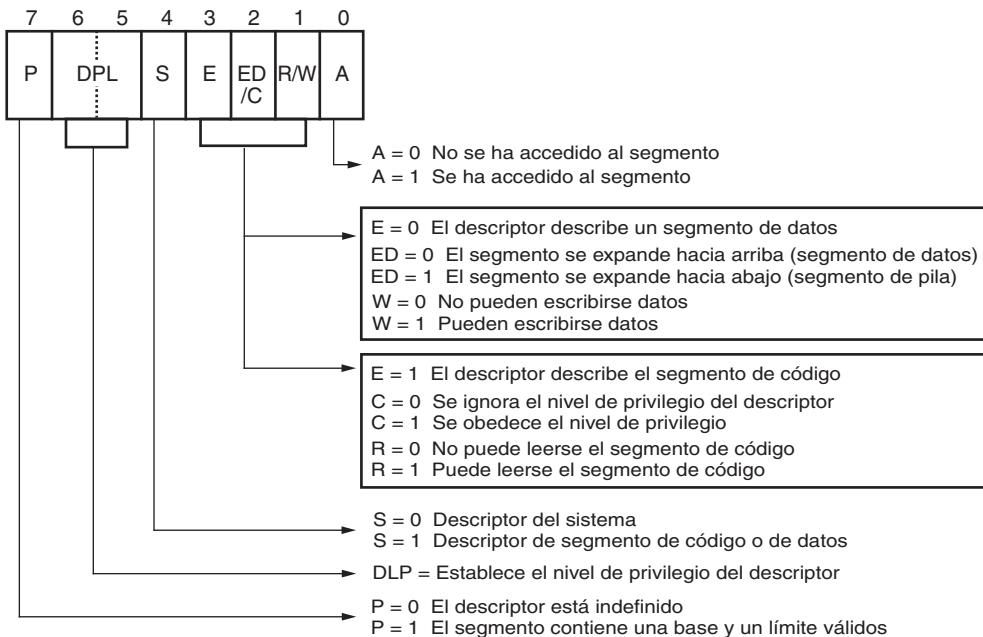
```
Base = Inicio = 10000000H
G = 0
Final = Base + Límite = 10000000H + 001FFH = 100001FFH
```

El ejemplo 2-2 utiliza los mismos datos que el ejemplo 2-1, sólo que el bit G = 1. Observe que el límite se adjunta con FFFFH para determinar la dirección de segmento final.

EJEMPLO 2-2

```
Base = Inicio = 10000000H
G = 1
Final = Base + Límite = 10000000H + 001FFFFH = 101FFFFH
```

El bit AV en el descriptor de los microprocesadores 80386 y superiores es utilizado por algunos sistemas operativos para indicar que el segmento está disponible (AV = 1) o no está disponible (AV = 0). El bit D indica cómo las instrucciones de los microprocesadores del 80386 al Pentium 4 acceden a los



Nota: Algunas de las letras utilizadas para describir los bits en los bytes de derechos de acceso varían en la documentación de Intel.

FIGURA 2-7 El byte de derechos de acceso para el descriptor de los microprocesadores del 80286 al Pentium 4.

datos de los registros y de la memoria en el modo real o protegido. Si D = 0, las instrucciones son de 16 bits, compatibles con los microprocesadores 8086-80286. Esto significa que las instrucciones utilizan direcciones de desplazamiento de 16 bits y registros de 16 bits de manera predeterminada. A este modo se le llama por lo general el modo de instrucciones de 16 bits o modo DOS. Si D = 1, las instrucciones son de 32 bits. De manera predeterminada, el modo de instrucciones de 32 bits supone que todas las direcciones de desplazamiento y los registros son de 32 bits. El valor predeterminado para el tamaño de los registros y la dirección de desplazamiento se redefinen en los modos de instrucciones de 16 bits y de 32 bits. Los sistemas operativos MSDOS y PC-DOS requieren que las instrucciones siempre se utilicen en el modo de instrucciones de 16 bits. Windows 3.1 y cualquier aplicación que haya sido escrita para este sistema también requieren que se seleccione el modo de instrucciones de 16 bits. El modo de instrucciones es accesible sólo en un sistema de modo protegido como Windows XP. En los capítulos 3 y 4 veremos más detalles sobre estos modos y su aplicación en el conjunto de instrucciones.

El **byte de derechos de acceso** (vea la figura 2-7) controla el acceso al segmento en modo protegido. Este byte describe la manera en que funciona el segmento en el sistema. El byte de derechos de acceso permite un completo control sobre el segmento. Si el segmento es de datos, se especifica la dirección de crecimiento. Si el segmento crece más allá de su límite se interrumpe el programa del sistema operativo del microprocesador, indicando una falla de protección general. Incluso se puede especificar si se puede escribir en un segmento de datos o si está protegido contra escritura. El segmento de código también se controla de una forma similar y puede inhibir la lectura para proteger el software.

Los descriptores se seleccionan de la tabla mediante el registro de segmento. La figura 2-8 muestra cómo funciona el registro de segmento en el sistema en modo protegido. El registro de segmento contiene un campo selector de 13 bits, un bit selector de tabla y un campo de nivel de privilegio solicitado. El **selector** de 13 bits selecciona uno de los 8192 descriptores de la tabla. El **bit TI** selecciona ya sea la tabla de descriptores globales (TI = 0) o la tabla de descriptores locales (TI = 1). El **nivel de privilegio solicitado** (RPL) solicita el nivel de privilegio de acceso de un segmento de memoria. El nivel de privilegio más alto es 00 y el menor es 11. Si el nivel de privilegio solicitado concuerda o tiene mayor prioridad que el nivel de privilegio establecido por el byte de derechos de acceso, se otorga el acceso. Por ejemplo, si el nivel de privilegio solicitado es de 10 y el byte de derechos de acceso establece el nivel de privilegio del segmento en 11, se otorga el acceso ya que 10 es mayor en prioridad que el nivel de

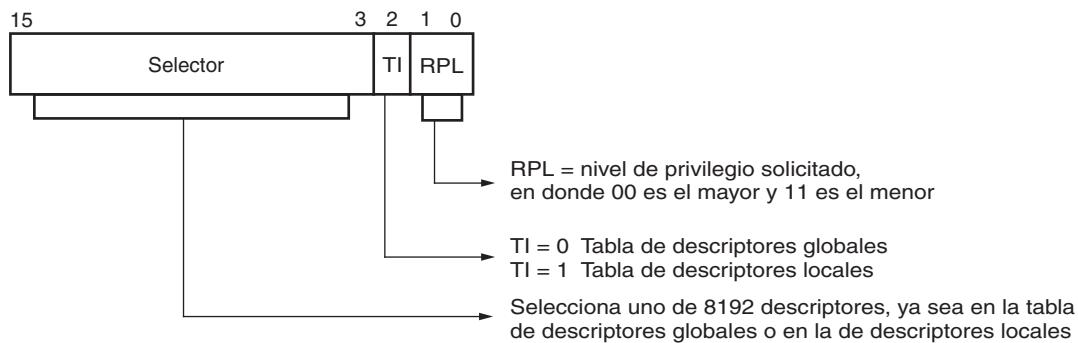


FIGURA 2-8 El contenido de un registro de segmento durante la operación en modo protegido de los microprocesadores del 80286 al Pentium 4.

privilegio 11. Los niveles de privilegio se utilizan en entornos multiusuario. Windows utiliza el nivel de privilegio 00 (**anillo 0**) para el núcleo (kernel) y los programas controladores, y el nivel 11 (**anillo 3**) para las aplicaciones. Windows no utiliza los niveles 01 o 10. Si se violan los niveles de privilegio, el sistema por lo general indica una violación de aplicación o de nivel de privilegio.

La figura 2-9 muestra cómo el registro de segmento, que contiene un selector, escoge un descriptor de la tabla de descriptores globales. La entrada en la tabla de descriptores globales selecciona un segmento en el sistema de memoria. En esta ilustración DS contiene 0008H, que accede al número de descriptor 1 de la tabla de descriptores globales mediante el uso de un nivel de privilegio solicitado de 00. El número de descriptor 1 contiene un descriptor que define la dirección base como 0010000H con un límite de segmento de 000FFH. Esto significa que si se carga un valor de 0008H en DS el microprocesador utilizará las posiciones de memoria 0010000H-001000FFH para el segmento de datos con esta tabla de descriptores de ejemplo. Al descriptor cero se le nombra descriptor nulo, el cual debe contener sólo ceros y no puede utilizarse para acceder a la memoria.

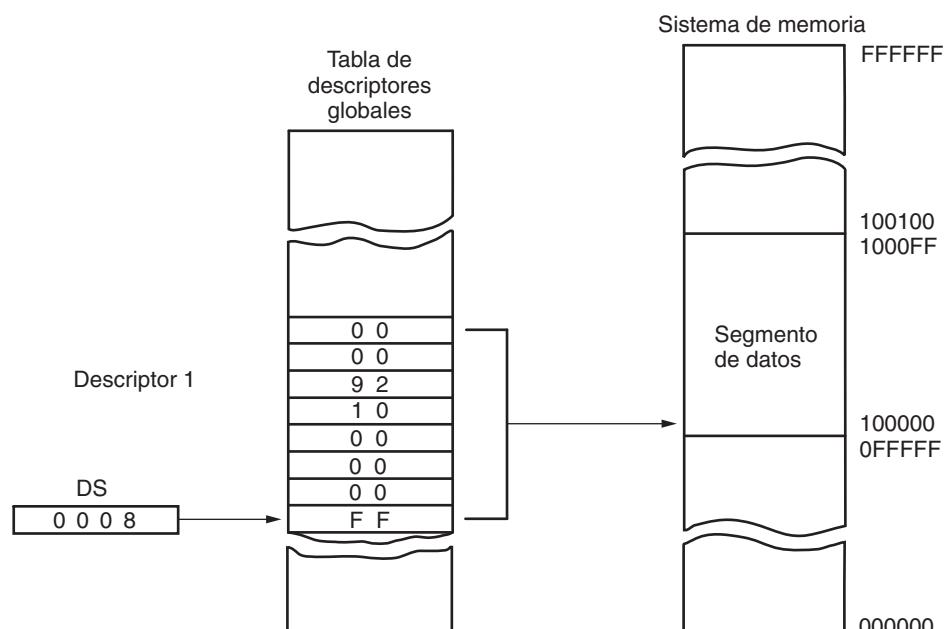


FIGURA 2-9 Uso del registro DS para seleccionar una descripción de la tabla de descriptores globales. En este ejemplo, el registro DS accede a las posiciones de memoria 100000H-1000FFH como un segmento de datos.

Registros invisibles para los programas

Las tablas de descriptores globales y locales se encuentran en el sistema de memoria. Para poder acceder a estas tablas y especificar su dirección, los microprocesadores del 80286 al Pentium 4 contienen registros invisibles para los programas. Como el software no direcciona directamente estos registros invisibles para los programas, por eso se les da ese nombre (aunque el software del sistema accede a algunos de estos registros). La figura 2-10 ilustra la manera en que aparecen los registros invisibles para los programas en los microprocesadores del 80286 al Pentium 4. Estos registros controlan el microprocesador cuando éste opera en modo protegido.

Cada uno de los registros de segmento contiene una porción invisible para los programas que se utilizan en el modo protegido. La porción invisible para los programas de estos registros se llama por lo general memoria caché, ya que la caché es cualquier memoria que almacena información. Esta caché no debe confundirse con las cachés de nivel 1 o de nivel 2 que se encuentran en el microprocesador. La porción del registro de segmento invisible para los programas se carga con la dirección base, con el límite y los derechos de acceso cada vez que se modifica el número del registro de segmento. Cuando se coloca un nuevo número de segmento en un registro de segmento, el microprocesador accede a una tabla de descriptores y carga el descriptor en la porción del registro de segmento invisible para los programas. Se mantiene ahí y se utiliza para acceder al segmento de memoria hasta que el número de segmento se cambie de nuevo. Esto permite al microprocesador acceder varias veces a un segmento de memoria sin hacer referencia a la tabla de descriptores (de ahí que se utilice el término caché).

El GDTR (**registro de tabla de descriptores globales**) y el IDTR (**registro de tabla de descriptor de interrupciones**) contienen la dirección base de la tabla de descriptores y su límite. El límite de cada tabla de descriptores es de 16 bits, ya que la máxima longitud de tabla es de 64 Kbytes. Cuando se desea la operación en modo protegido, se carga la dirección de la tabla de descriptores globales y su límite en el GDTR.

Antes de usar el modo protegido hay que inicializar la tabla de descriptores de interrupciones y el IDTR. Más adelante veremos los detalles sobre la operación en modo protegido. En este punto es imposible ver la programación y la descripción adicional de estos registros.



FIGURA 2-10 El registro invisible para los programas dentro de los microprocesadores del 80286 al Pentium 4.

La posición de la tabla de descriptores locales se selecciona de la tabla de descriptores globales. Uno de los descriptores globales se establece de manera que direccione la tabla de descriptores locales. Para acceder a la tabla de descriptores locales, el LDTR (**registro de tabla de descriptores locales**) se carga con un selector, de igual forma que un registro de segmento se carga con un selector. Este selector accede a la tabla de descriptores globales y carga la dirección, el límite y los derechos de acceso de la tabla de descriptores locales en la porción de la caché del LDTR.

El TR (**registro de tareas**) guarda un selector, el cual accede a un descriptor que define una tarea. Ésta viene siendo por lo general un procedimiento o un programa de aplicación. El descriptor para el procedimiento o el programa de aplicación se almacena en la tabla de descriptores globales, por lo que el acceso puede controlarse a través de los niveles de privilegio. El registro de tareas permite cambiar de contexto o de tarea en aproximadamente 17 μ s. La conmutación de tareas permite al microprocesador conmutar entre varias tareas en un tiempo considerablemente corto. El conmutador de tareas permite a los sistemas de multitarea conmutar de una tarea a otra de una forma simple y ordenada.

2-4

PAGINACIÓN DE MEMORIA

El **mecanismo de paginación de memoria** ubicado dentro de los microprocesadores 80386 y superiores permite asignar cualquier posición de memoria física a cualquier dirección lineal. La **dirección lineal** se define como la dirección generada por un programa. La **dirección física** es la posición actual de memoria utilizada por un programa. Con la unidad de paginación de memoria, la dirección lineal se traduce de forma invisible a cualquier dirección física, lo cual permite que una aplicación escrita para funcionar en una dirección específica pueda reubicarse mediante el mecanismo de paginación. También permite colocar la memoria en áreas en las que no existe. Un ejemplo son los bloques de memoria superior que proporciona EMM386.EXE en un sistema DOS.

El programa EMM386.EXE reasigna la memoria extendida, en bloques de 4 K, a la memoria del sistema que está entre el BIOS del vídeo y los ROMS del BIOS del sistema para los bloques de memoria superior. Sin el mecanismo de paginación es imposible utilizar esta área de la memoria.

En Windows cada aplicación tiene permitido un espacio de direcciones lineales de 2 G en las posiciones 0000000H-7FFFFFFFH, aún y cuando no pueda haber suficiente memoria o memoria disponible en estas direcciones. Cualquier aplicación Windows puede ejecutarse mediante la paginación en la unidad de disco duro y la paginación en la memoria a través de la unidad de paginación de memoria.

Registros de paginación

La unidad de paginación se controla basándose en el contenido de los registros de control del microprocesador. En la figura 2-11 podrá ver el contenido de los registros de control CR0 al CR4. Tenga en cuenta que estos registros están disponibles en los microprocesadores del 80386 al Pentium. Comenzando con el Pentium hay un registro de control adicional llamado CR4, que controla las extensiones a la arquitectura básica proporcionada en el Pentium o en los microprocesadores más recientes. Una de estas características es una página de 4 Mbytes que se habilita al establecer la posición del bit 4 del CR4. En los capítulos 18 y 19 podrá consultar más detalles sobre el sistema de paginación de memoria de 4 Mbytes.

Los registros que son importantes para la unidad de paginación son CR0 y CR3. La posición del bit más a la izquierda (PG) de CR0 selecciona la paginación cuando se coloca a un nivel lógico de 1. Si el bit PG está inactivo (0), la dirección lineal que genera el programa se convierte en la dirección física que se utilice para acceder a la memoria. Si el bit PG está activo (1), la dirección lineal se convierte en una dirección física a través del mecanismo de paginación. Este mecanismo funciona en los modos real y protegido.

El CR3 contiene la base del directorio de páginas, o dirección raíz, y los bits PCD y PWT. Estos bits controlan la operación de las terminales PCD y PWT en el microprocesador. Si PCD está activo (1), la terminal PCD se convierte en uno lógico durante los ciclos de bus que no se paginen. Esto permite al hardware externo controlar la memoria de caché de nivel 2. (Tenga en cuenta que esta memoria es una memoria interna [en versiones modernas del Pentium] de alta velocidad que funciona como un búfer

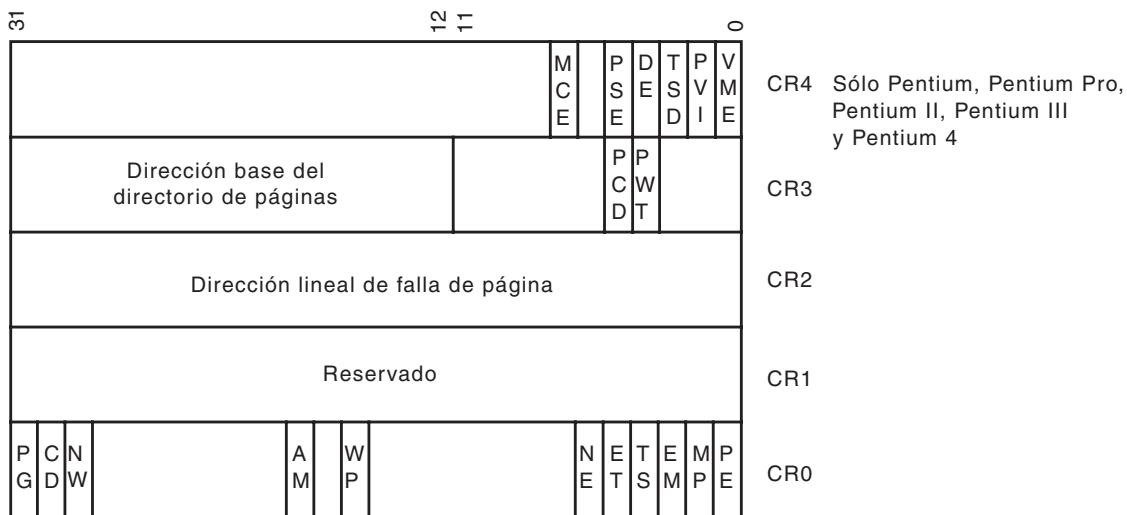


FIGURA 2-11 La estructura del registro de control del microprocesador.

entre el microprocesador y el sistema de memoria DRAM principal.) El bit PWT también aparece en la terminal PWT durante los ciclos de bus que no se paginan para controlar la escritura a través de la caché en el sistema. La dirección base del directorio de páginas localiza el directorio para la unidad de traducción de páginas. Esta dirección localiza el directorio de páginas en cualquier límite de 4 K en el sistema de memoria, ya que se adjunta en forma interna con 000H. El directorio de páginas contiene 1024 entradas de cuatro bytes cada una. Cada entrada en el directorio de páginas direcciona una tabla de páginas que contiene 1024 entradas.

A medida que el software genera la dirección lineal, ésta se descompone en tres secciones que se utilizan para acceder a la **entrada en el directorio de páginas**, la **entrada en la tabla de páginas** y la **dirección de desplazamiento de la página de memoria**. La figura 2-12 muestra la dirección lineal y su estructura para la paginación. Observe cómo los 10 bits de más a la izquierda direccionan una entrada en el directorio de páginas. Para la dirección lineal 00000000H-003FFFFH, se accede al primer directorio de páginas. Cada entrada en el directorio de páginas representa o repagina una

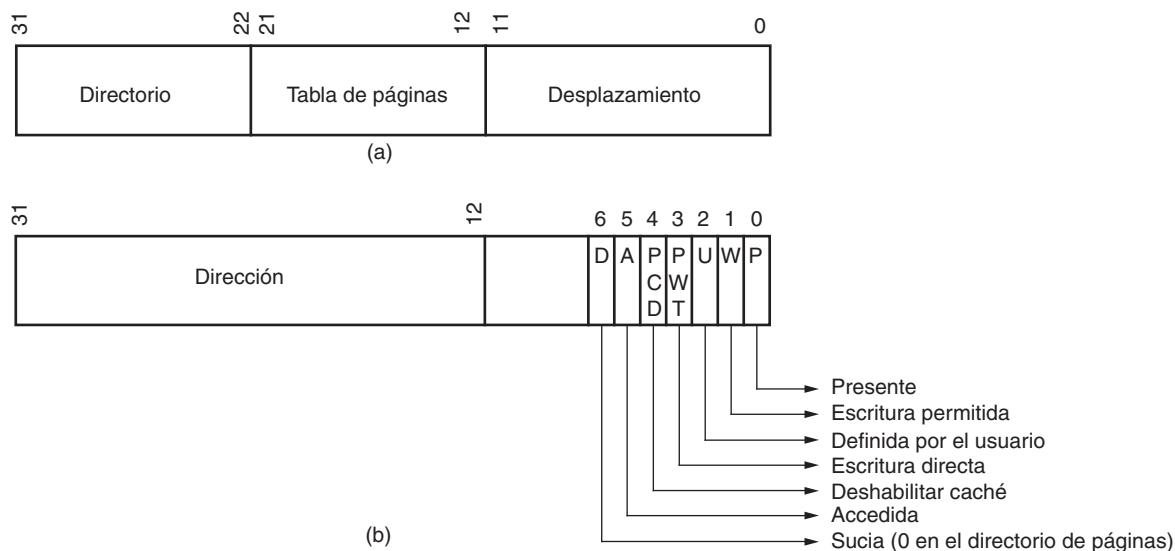


FIGURA 2-12 El formato para la dirección lineal (a) y una entrada en la tabla de páginas o en el directorio de páginas (b).

sección de 4 M del sistema de memoria. El contenido del directorio de páginas selecciona una tabla de páginas que está indizada por los siguientes 10 bits de la dirección lineal (posiciones de bit 12-21). Esto significa que la dirección 00000000H-00000FFFFH selecciona la entrada 0 en el directorio de páginas y la entrada 0 en la tabla de páginas. Este rango de direcciones es de 4 Kbytes. La parte de la dirección lineal correspondiente al desplazamiento (posiciones de bits 0-11) selecciona a continuación un byte en la página de memoria de 4 Kbytes. En la figura 2-12, si la entrada 0 de la tabla de páginas contiene la dirección 00100000H, entonces la dirección física es 00100000H-00100FFFF para la dirección lineal 00000000H-00000FFFFH. Esto significa que cuando el programa accede a una posición entre 00000000H y 00000FFFFH, el microprocesador dirige físicamente la posición 00100000H-00100FFFFH.

Como el acto de repaginar una sección de memoria de 4 Kbytes requiere del acceso al directorio de páginas y a una tabla de páginas, las cuales se encuentran en memoria, Intel ha incorporado un tipo especial de caché llamado TLB (**búfer de traducción adelantada**). En el microprocesador 80486 la caché guarda las 32 direcciones de traducción de página más recientes. Esto significa que las últimas 32 traducciones de la tabla de páginas se almacenan en el TLB, por lo que si se accede a la misma área de memoria, la dirección ya está presente en el TLB y no se requiere el acceso al directorio de páginas ni a las tablas de páginas. Esto agiliza la ejecución del programa. Si una traducción no está en el TLB hay que acceder al directorio de páginas y a la tabla de páginas, lo cual requiere de un tiempo de ejecución adicional. Los microprocesadores del Pentium al Pentium 4 contienen TLBs separados para cada una de sus cachés de instrucciones y de datos.

El directorio de páginas y la tabla de páginas

La figura 2-13 muestra el directorio de páginas, unas cuantas tablas de páginas y algunas páginas de memoria. Sólo hay un directorio de páginas en el sistema. Éste contiene 1024 direcciones con tamaño de doble palabra que localizan hasta 1024 tablas de páginas. El directorio de páginas y cada tabla de páginas son de 4 Kbytes de longitud. Si se paginan todos los 4 Gbytes de memoria, el sistema debe asignar 4 Kbytes de memoria para el directorio de páginas y 4 K por 1024, o 4 Mbytes para las 1024 tablas de páginas. Esto representa una inversión considerable en recursos de memoria.

El sistema DOS y el programa EMM386.EXE utilizan tablas de páginas para redefinir el área de memoria entre las posiciones C8000H-EFFFFH como bloques de memoria superior. Esto se hace mediante una repaginación de la memoria extendida para llenar esta parte del sistema de memoria.

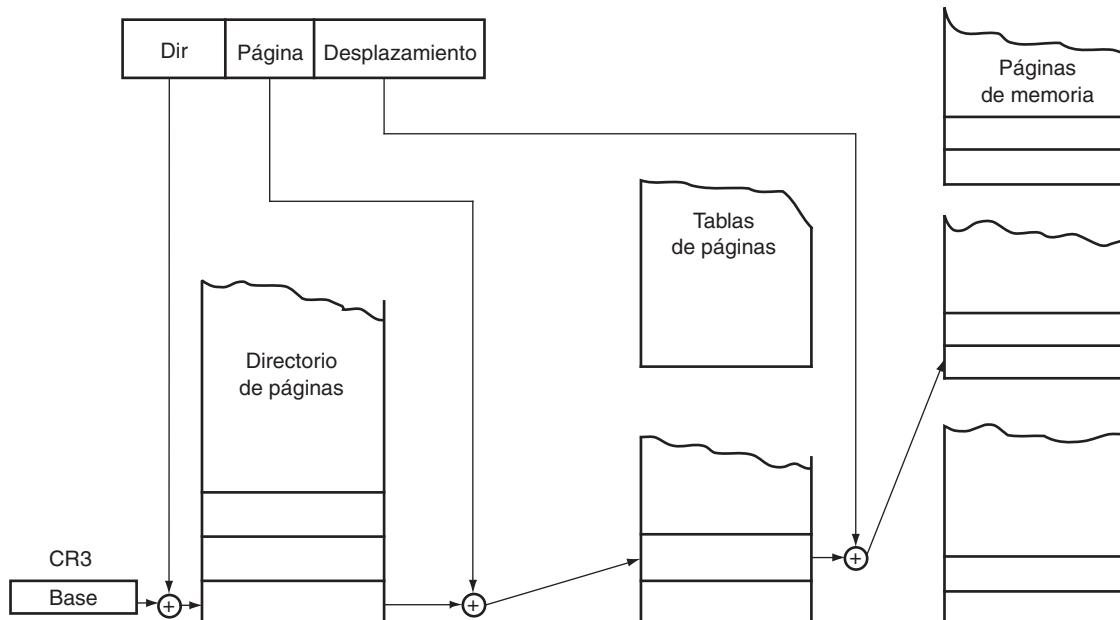
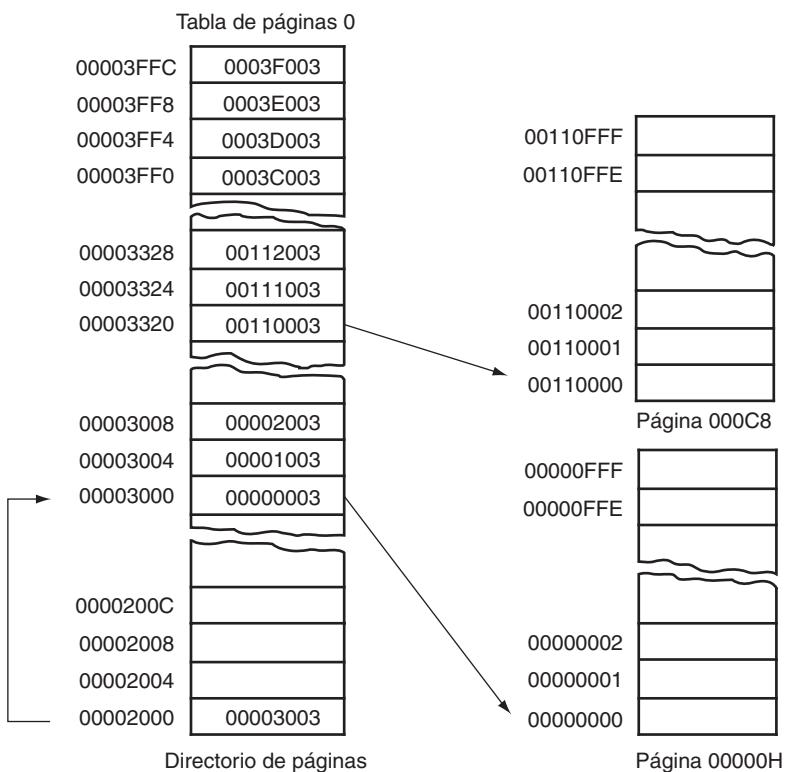


FIGURA 2-13 El mecanismo de paginación en los microprocesadores del 80386 al Pentium 4.

FIGURA 2-14 El directorio de páginas, la tabla de páginas 0 y dos páginas de memoria. Observe cómo la dirección de la página 000C8000-000C9000 se ha desplazado a 00110000-00110FFF.



convencional y permitir que el DOS tenga acceso a la memoria adicional. Suponga que el programa EMM386.EXE permite el acceso a 16 Mbytes de memoria extendida y convencional a través de la paginación, y que las posiciones C8000H-EFFFFH deben repaginarse a las posiciones 110000H-138000H, en donde las demás áreas de memoria se paginan a sus posiciones normales. Dicho esquema se describe en la figura 2-14.

Aquí el directorio de páginas contiene cuatro entradas. Recuerde que cada entrada en el directorio de páginas corresponde a 4 Mbytes de memoria física. El sistema también contiene cuatro tablas de páginas con 1024 entradas cada una. Recuerde que cada entrada en la tabla de páginas repagina 4 Kbytes de memoria física. Este esquema requiere un total de 16 K de memoria para las cuatro tablas de páginas y 16 bytes de memoria para el directorio de páginas.

Al igual que el DOS, el programa Windows también repagina el sistema de memoria. La versión 3.11 de Windows soporta la paginación sólo para 16 Mbytes de memoria, debido a la cantidad de memoria requerida para almacenar las tablas de páginas. Las versiones más recientes de Windows repagan todo el sistema de memoria. En los microprocesadores Pentium-Pentium 4 las páginas pueden ser de 4 Kbytes de longitud o de 4 Mbytes. Aunque no hay software en la actualidad que soporte las páginas de 4 Mbytes, a medida que el Pentium 4 y versiones más avanzadas se difundan en el área de las computadoras personales, sin duda los sistemas operativos del futuro empezarán a soportar páginas de memoria de 4 Mbytes.

2-5

RESUMEN

1. El modelo de programación de los microprocesadores del 8086 al 80286 contiene registros de 8 y de 16 bits. El modelo de programación de los microprocesadores 80386 y superiores contiene registros extendidos de 8, 16 y 32 bits, así como dos registros de segmento de 16 bits adicionales: FS y GS.

2. Los registros de 8 bits son AH, AL, BH, BL, CH, CL, DH y DL. Los registros de 16 bits son AX, BX, CX, DX, SP, BP, DI y SI. Los registros de segmento son CS, DS, ES, SS, FS y GS. Los registros extendidos de 32 bits son EAX, EBX, ECX, EDX, ESP, EBP, EDI y ESI. Además el microprocesador contiene un apuntador de instrucciones (IP/EIP) y un registro de bandera (FLAGS o EFLAGS).
3. Todas las direcciones de memoria en modo real son una combinación de una dirección de segmento más una dirección de desplazamiento. La posición inicial de un segmento se define mediante el número de 16 bits en el registro de segmento al que se adjunta un cero hexadecimal en el extremo derecho. La dirección de desplazamiento es un número de 16 bits que se suma a la dirección de segmento de 20 bits para formar la dirección de memoria en modo real.
4. Todas las instrucciones (código) se acceden mediante la combinación de CS (dirección de segmento) más IP o EIP (dirección de desplazamiento).
5. Por lo general se hace referencia a los datos a través de una combinación del DS (segmento de datos) y una dirección de desplazamiento o el contenido de un registro que contenga la dirección de desplazamiento. Los microprocesadores del 8086 al Pentium 4 utilizan BX, DI y SI como registros de desplazamiento predeterminados para los datos si se seleccionan registros de 16 bits. El 80386 y superiores pueden usar los registros de 32 bits EAX, EBX, ECX, EDX, EDI y ESI como registros de desplazamiento predeterminados para los datos.
6. La operación en modo protegido permite acceder a la memoria que está por encima del primer 1 Mbyte en los microprocesadores del 80286 al Pentium 4. Este sistema de memoria extendida (XMS) se accede a través de una dirección de segmento más una dirección de desplazamiento, justo igual que en el modo real. La diferencia es que la dirección de segmento no se guarda en el registro de segmento. En el modo protegido, la dirección de inicio de segmento se almacena en un descriptor que se selecciona mediante el registro de segmento.
7. Un descriptor de modo protegido contiene una dirección base, un límite y un byte de derechos de acceso. La dirección base localiza la dirección inicial del segmento de memoria; el límite define la última posición del segmento. El byte de derechos de acceso define cómo se accede al segmento de memoria mediante un programa. El microprocesador 80286 permite que un segmento de memoria empiece en cualquiera de sus 16 Mbytes de memoria mediante el uso de una dirección base de 24 bits. El 80386 y superiores permiten que un segmento de memoria empiece en cualquiera de sus 4 Gbytes de memoria, mediante el uso de una dirección base de 32 bits. El límite es un número de 16 bits en el 80286 y un número de 20 bits en el 80386 y superiores. Esto permite al 80286 un límite de 64 Kbytes en el segmento de memoria, y al 80386 y superiores un límite de 1 Mbyte ($G = 0$) o de 4 Gbytes ($G = 1$) en el segmento de memoria.
8. El registro de segmento contiene tres campos de información en el modo protegido. Los 13 bits de más a la izquierda del registro de segmento direccionan uno de 8192 descriptores de una tabla de descriptores. El bit TI accede a la tabla de descriptores globales ($TI = 0$) o a la tabla de descriptores locales ($TI = 1$). Los 2 bits de más a la derecha del registro de segmento seleccionan el nivel de prioridad solicitado para el acceso al segmento de memoria.
9. Los microprocesadores 80286 y superiores utilizan los registros invisibles para los programas para acceder a las tablas de descriptores. Cada registro de segmento contiene una porción de caché que se utiliza en modo protegido para guardar la dirección base, el límite y los derechos de acceso adquiridos de un descriptor. La caché permite al microprocesador acceder al segmento de memoria sin tener que volver a hacer referencia a la tabla de descriptores, hasta que se cambie el contenido del registro de segmento.
10. Una página de memoria tiene 4 Kbytes de longitud. La dirección lineal, como la genera un programa, puede asignarse a cualquier dirección física a través del mecanismo de paginación que se encuentra dentro de los microprocesadores del 80386 al Pentium 4.
11. La paginación de memoria se lleva a cabo a través de los registros de control CR0 y CR3. El bit PG de CR0 habilita la paginación y el contenido de CR3 direcciona el directorio de páginas. Este directorio contiene hasta 1024 direcciones de tablas de páginas que se utilizan para acceder a las tablas de páginas. La tabla de páginas contiene 1024 entradas que localizan la dirección física de una página de memoria de 4 Kbytes.
12. El TLB (búfer de traducción adelantada) guarda en la caché las 32 traducciones más recientes de tablas de páginas. Esto evita la traducción de tablas de páginas si ésta reside en el TLB, con lo cual se agiliza la ejecución del software.

2-6**PREGUNTAS Y PROBLEMAS**

1. ¿Qué son los registros visibles para los programas?
2. El 80286 direcciona registros que son de 8 y de _____ bits.
3. ¿Cuáles microprocesadores pueden direccionar los registros extendidos?
4. El registro extendido BX se direcciona como _____.
5. ¿Cuál registro guarda la cuenta para algunas instrucciones?
6. ¿Cuál es el propósito del registro IP/EIP?
7. ¿Cuáles son las operaciones aritméticas que no modifican el bit de bandera de acarreo?
8. ¿Ocurrirá un desbordamiento si se suma un FFH con signo a un 01H con signo?
9. Se dice que un número que contiene 3 bits uno tiene paridad _____.
10. ¿Cuál bit de bandera controla la terminal INTR en el microprocesador?
11. ¿Cuáles microprocesadores contienen un registro de segmento FS?
12. ¿Cuál es el propósito de un registro de segmento en la operación en modo real del microprocesador?
13. En el modo real, muestre las direcciones inicial y final de cada segmento localizado por los siguientes valores del registro de segmento:
 - (a) 1000H
 - (b) 1234H
 - (c) 2300H
 - (d) E000H
 - (e) AB00H
14. Encuentre la dirección en memoria de la siguiente instrucción ejecutada por el microprocesador, cuando opera en el modo real, para las siguientes combinaciones de CS:IP:
 - (a) CS = 1000H e IP = 2000H
 - (b) CS = 2000H e IP = 1000H
 - (c) CS = 2300H e IP = 1A00H
 - (d) CS = 1A00H e IP = B000H
 - (e) CS = 3456H e IP = ABCDH
15. ¿Las direcciones de memoria en modo real permiten el acceso a la memoria que está debajo de cuál dirección de memoria?
16. ¿Qué registro o registros se utiliza(n) como una dirección de desplazamiento para el destino de la instrucción de cadena en el microprocesador?
17. ¿Qué registro o registros de 32 bits se utiliza(n) para guardar una dirección de desplazamiento para los datos del segmento de datos en el microprocesador Pentium 4?
18. La memoria de la pila se direcciona mediante una combinación del segmento _____ más el desplazamiento _____.
19. Si el apuntador base (BP) direcciona memoria, el segmento _____ contiene los datos.
20. Determine la posición de memoria que se direcciona mediante las siguientes combinaciones de registros del 80286 en modo real:
 - (a) DS = 1000H y DI = 2000H
 - (b) DS = 2000H y SI = 1002H
 - (c) SS = 2300H y BP = 3200H
 - (d) DS = A000H y BX = 1000H
 - (e) SS = 2900H y SP = 3A00H
21. Determine la posición de memoria que se direcciona mediante las siguientes combinaciones de registros del Pentium 4 en modo real:
 - (a) DS = 2000H y EAX = 00003000H
 - (b) DS = 1A00H y ECX = 00002000H
 - (c) DS = C000H y ESI = 0000A000H
 - (d) SS = 8000H y ESP = 00009000H
 - (e) DS = 1239H y EDX = 0000A900H
22. ¿A cuál área de la memoria permite el acceso el direccionamiento de memoria en modo protegido en el microprocesador 80286?

23. ¿A cuál área de la memoria permite el acceso el direccionamiento de memoria en modo protegido en el microprocesador Pentium 4?
24. ¿Cuál es el propósito del registro de segmento en un direccionamiento de memoria en modo protegido?
25. ¿Cuántos descriptores son accesibles en la tabla de descriptores globales en modo protegido?
26. ¿Cuáles son las posiciones inicial y final que se direccionan sobre la base de un descriptor del 80286 que contiene una dirección base de A00000H y un límite de 1000H?
27. ¿Cuáles son las posiciones inicial y final que se direccionan sobre la base de un descriptor del Pentium 4 que contiene una dirección base de 01000000H, un límite de 0FFFFH y G = 0?
28. ¿Cuáles son las posiciones inicial y final que se direccionan sobre la base de un descriptor del Pentium 4 que contiene una dirección base de 00280000H, un límite de 00010H y G = 1?
29. Si el registro DS contiene 0020H en un sistema en modo protegido, ¿qué tabla de descriptores globales es la que se accesa?
30. Si DS = 0103H en un sistema en modo protegido, el nivel de privilegio solicitado es _____.
31. Si DS = 0105H en un sistema en modo protegido, ¿qué entrada, tabla y nivel de privilegio solicitado se seleccionan?
32. ¿Cuál es la máxima longitud de la tabla de descriptores globales en el microprocesador Pentium 4?
33. Codifique un descriptor que describa un segmento de memoria que empiece en la posición 210000H y termine en la posición 21001FH. Este segmento de memoria es un segmento de código que puede leerse. El descriptor es para un microprocesador 80286.
34. Codifique un descriptor que describa un segmento de memoria que empiece en la posición 03000000H y termine en la posición 05FFFFFFH. Este segmento de memoria es un segmento de datos que crece hacia arriba en el sistema de memoria y puede escribirse en él. El descriptor es para un microprocesador Pentium 4.
35. ¿Qué registro localiza la tabla de descriptores globales?
36. ¿Cómo se direcciona la tabla de descriptores globales en el sistema de memoria?
37. Describa lo que ocurre cuando se carga un nuevo número en un registro de segmento cuando el microprocesador se opera en el modo protegido.
38. ¿Qué son los registros invisibles para los programas?
39. ¿Cuál es el propósito del GDTR?
40. ¿Cuántos bytes hay en una página de memoria?
41. ¿Qué registro se utiliza para habilitar el mecanismo de paginación en los microprocesadores 80386, 80486, Pentium, Pentium Pro y Pentium 4?
42. ¿Cuántas direcciones de 32 bits se almacenan en el directorio de páginas?
43. ¿Cuánta memoria lineal traduce cada entrada en el directorio de páginas en memoria física?
44. Si el microprocesador envía la dirección lineal 00200000H al mecanismo de paginación, ¿qué entrada de directorio de paginación se accesa y qué entrada de tabla de páginas se accesa?
45. ¿Qué valor se coloca en la tabla de páginas para redirigir la dirección lineal 20000000H a la dirección física 30000000H?
46. ¿Cuál es el propósito del TLB ubicado dentro del microprocesador de la clase Pentium?
47. Mediante el uso de Internet, escriba un informe breve en el que se detalle el TLB. Sugerencia: tal vez sea conveniente que visite el sitio Web de Intel y busque información.
48. Localice artículos sobre paginación en Internet y escriba un informe en el que detalle cómo se utiliza la paginación en una variedad de sistemas.

CAPÍTULO 3

Modos de direccionamiento

INTRODUCCIÓN

Para desarrollar software eficiente para el microprocesador se requiere de una completa familiaridad con los modos de direccionamiento empleados por cada instrucción. En este capítulo se utiliza la instrucción MOV (**mover datos**) para describir los modos de direccionamiento de datos. La instrucción MOV transfiere bytes o palabras de datos entre registros, o entre registros y la memoria en los microprocesadores del 8086 al 80286. Los bytes, las palabras y las dobles palabras se transfieren en el 80386 y superiores mediante un MOV. Al describir los modos de direccionamiento de memoria de un programa, las instrucciones CALL y JUMP muestran cómo modificar el flujo del programa.

Los modos de direccionamiento de datos son: de registro, inmediato, directo, de registro indirecto, de base más índice, de registro relativo y de base relativa más índice en los microprocesadores del 8086 al 80286. Los microprocesadores 80386 y superiores también incluyen un modo de índice escalado para direccionar los datos de la memoria. Los modos de direccionamiento de memoria de programa son: relativo al programa, directo e indirecto. En este capítulo explicaremos la operación de la memoria de la pila para que puedan comprenderse las instrucciones PUSH y POP, junto con otras operaciones de la pila.

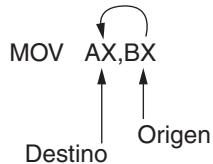
OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

1. Explicar la operación de cada uno de los modos de direccionamiento de datos.
2. Utilizar los modos de direccionamiento de datos para formar instrucciones en lenguaje ensamblador.
3. Explicar la operación de cada uno de los modos de direccionamiento de memoria del programa.
4. Usar los modos de direccionamiento de memoria del programa para formar instrucciones en lenguaje ensamblador y lenguaje máquina.
5. Seleccionar el modo de direccionamiento apropiado para realizar una tarea dada.
6. Detallar la diferencia entre el direccionamiento de los datos de la memoria mediante el uso de la operación en modo real y en modo protegido.
7. Describir la secuencia de eventos que colocan datos en la pila o quitan datos de la pila.
8. Explicar cómo se coloca una estructura de datos en memoria y cómo se utiliza con el software.

Como la instrucción MOV es muy común y flexible, proporciona una base para la explicación de los modos de direccionamiento de datos. La figura 3-1 ilustra la instrucción MOV y define la dirección

FIGURA 3-1 La instrucción MOV muestra el origen, el destino y la dirección del flujo de datos.



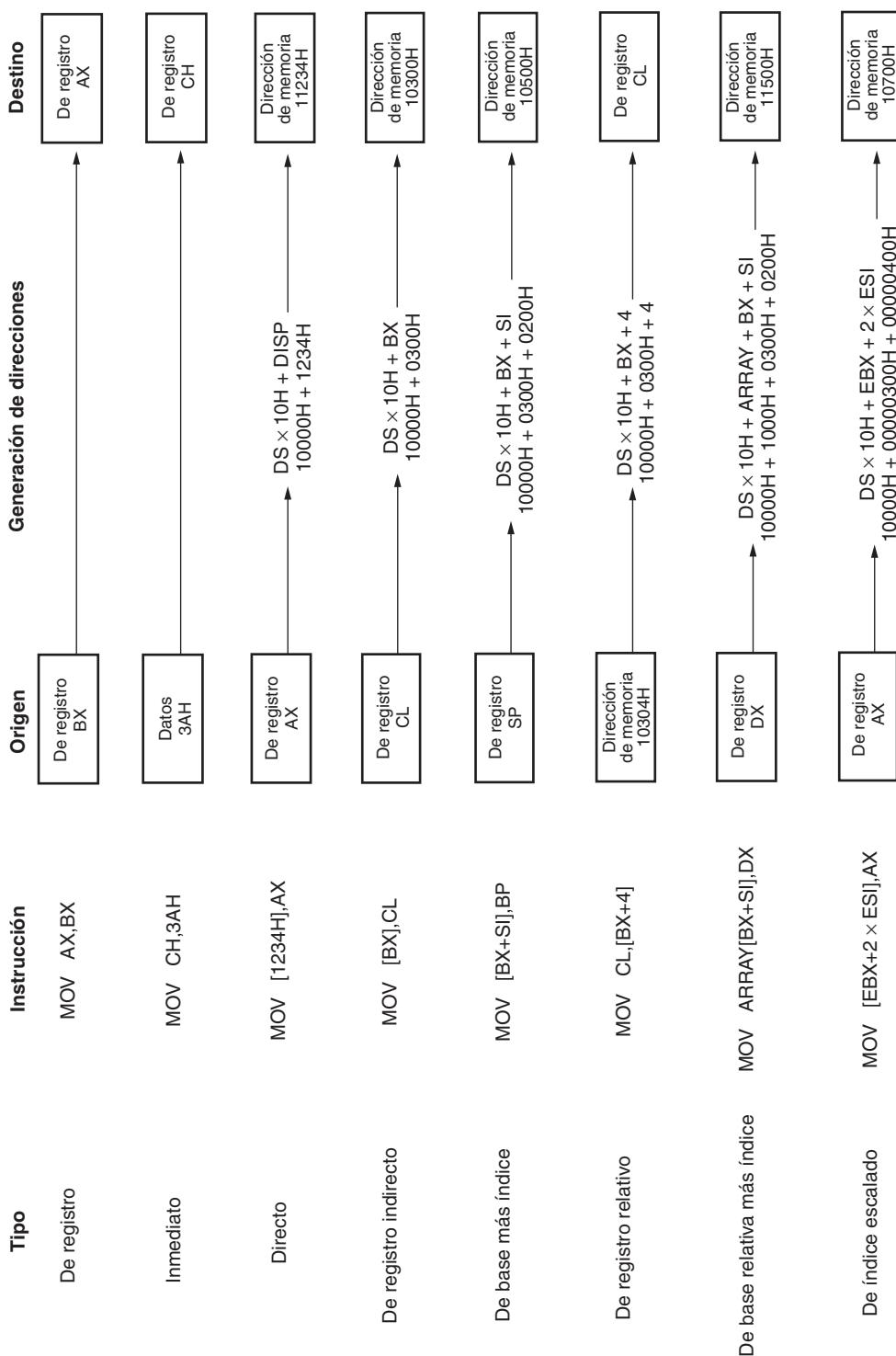
del flujo de datos. El **origen** está a la derecha y el **destino** está a la izquierda, enseguida del código de operación MOV. (Un **opcode**, o código de operación, indica al microprocesador qué operación realizar.) Esta dirección del flujo, que se aplica a todas las instrucciones, es molesta al principio. De manera natural suponemos que las cosas se mueven de izquierda a derecha, mientras que aquí se mueven de derecha a izquierda. Observe que siempre hay una coma que separa el destino del origen en una instrucción. Además, *ninguna* instrucción tiene permitido realizar transferencias de memoria a memoria, más que la instrucción MOVS.

En la figura 3-1, la instrucción MOV AX,BX transfiere el contenido de la palabra del registro de origen (BX) al registro de destino (AX). El origen nunca cambia, pero el destino siempre lo hace.¹ Es imprescindible recordar que una instrucción MOV siempre *copia* los datos del origen al destino. La operación MOV nunca recolecta los datos y los mueve. Además, el registro de banderas no se ve afectado por la mayoría de las instrucciones de transferencia de datos. Al origen y al destino se les llama por lo general **operандos**.

La figura 3-2 muestra las posibles variaciones de los modos de direccionamiento de datos mediante el uso de la instrucción MOV. Esta ilustración muestra cómo se formula cada uno de los modos de direccionamiento de datos con la instrucción MOV, y también sirve como referencia en relación con los modos de direccionamiento de datos. Observe que éstos son los mismos modos de direccionamiento de datos que se incluyen en todas las versiones del microprocesador Intel, excepto para el modo de direccionamiento de índice escalado, el cual se incluye sólo en los microprocesadores del 80386 al Pentium 4. Los modos de direccionamiento de datos son los siguientes:

Direccionamiento de registro	El direccionamiento de registro transfiere una copia de un byte o palabra del registro de origen, o el contenido de una posición de memoria al registro de destino o posición de memoria. (Ejemplo: la instrucción MOV CX,DX copia el contenido del tamaño de una palabra del registro DX al registro CX.) En los microprocesadores 80386 y superiores, una doble palabra puede transferirse desde el registro o posición de memoria de origen hasta el registro o posición de memoria de destino. (Ejemplo: la instrucción MOV ECX,EDX copia el contenido del tamaño de una doble palabra del registro EDX al registro ECX.)
Direccionamiento inmediato	Este modo de direccionamiento transfiere el origen (datos inmediatos tipo byte o palabra) al registro o posición de memoria de destino. (Ejemplo: la instrucción MOV AL,22H copia el número 22H del tamaño de un byte al registro AL.) En los microprocesadores 80386 y superiores puede transferirse una doble palabra de datos inmediatos hacia un registro o una posición de memoria. (Ejemplo: la instrucción MOV EBX, 12345678H copia el número 12345678H del tamaño de una doble palabra al registro EBX de 32 bits.)
Direccionamiento directo	Este modo de direccionamiento mueve un byte o palabra entre una posición de memoria y un registro. El conjunto de instrucciones no soporta una transferencia de memoria a memoria, excepto con la instrucción MOVS. (Ejemplo: la instrucción MOV CX,LISTA copia el contenido del tamaño de una palabra de la posición de memoria LISTA al registro CX.) En los microprocesadores 80386 y superiores también puede direccionarse una posición de memoria del tamaño de una doble palabra. (Ejemplo: la instrucción MOV ESI,LISTA copia un número de 32 bits, almacenado en cuatro bytes consecutivos de memoria, de la posición LIST al registro ESI.)

¹Las excepciones son las instrucciones CMP y TEST, que nunca cambian el destino. Describiremos estas instrucciones en capítulos posteriores.



Observaciones: EBX = 00000300H, ESI = 00000200H, ARRAY = 1000H, y DS = 1000H

FIGURA 3-2 Modos de direccionamiento de datos de los microprocesadores 8086-Pentium 4.

Direccionamiento de registro indirecto	Este modo de direccionamiento transfiere un byte o una palabra entre un registro y una posición de memoria direccionados por un registro índice o base. Los registros índice y base son BP, BX, DI y SI. (Ejemplo: la instrucción MOV AX,[BX] copia los datos del tamaño de una palabra desde la dirección de desplazamiento del segmento de datos indizado por BX, hacia el registro AX.) En los microprocesadores 80386 y superiores se transfiere un byte, una palabra o una doble palabra entre un registro y una posición de memoria direccionada por cualquier registro: EAX, EBX, ECX, EDX, EBP, EDI o ESI. (Ejemplo: la instrucción MOV AL,[ECX] carga el registro AL de la dirección de desplazamiento del segmento de datos seleccionada por el contenido de ECX.)
Direccionamiento de base más índice	Este modo de direccionamiento transfiere un byte o una palabra entre un registro y la posición de memoria direccionada por un registro base (BP o BX) más un registro índice (DI o SI). (Ejemplo: la instrucción MOV [BX+DI],CL copia el contenido del tamaño de un byte del registro CL en la posición de memoria del segmento de datos direccionada por BX más DL.) En los microprocesadores 80386 y superiores pueden combinarse dos registros cualesquiera (EAX, EBX, ECX, EDX, EBP, EDI o ESI) para generar la dirección de memoria. (Ejemplo: la instrucción MOV [EAX+EBX],CL copia el contenido del tamaño de un byte del registro CL en la posición de memoria del segmento de datos direccionada por EAX más EBX.)
Direccionamiento de registro relativo	Este modo de direccionamiento mueve un byte o una palabra entre un registro y la posición de memoria direccionada por un registro índice o base más un desplazamiento. (Ejemplo: MOV AX,[BX+4] o MOV AX,ARRAY[BX]. La primera instrucción carga AX en base a la dirección del segmento de datos formada por BX más 4. La segunda instrucción carga AX en base a la posición de memoria del segmento de datos en ARRAY más el contenido de BX.) Para direccionar memoria, los microprocesadores 80386 y superiores utilizan cualquier registro de 32 bits excepto ESP. (Ejemplo: MOV AX,[ECX+4] o MOV AX,ARRAY[EBX]. La primera instrucción carga AX en base a la dirección del segmento de datos formada por ECX más 4. La segunda instrucción carga AX en base a la posición de memoria ARRAY del segmento de datos más el contenido de EBX.)
Direccionamiento de base relativa más índice	Este modo de direccionamiento transfiere un byte o una palabra entre un registro y la posición de memoria direccionada por un registro base y un registro índice más un desplazamiento. (Ejemplo: MOV AX,ARRAY[BX+DI] o MOV AX,[BX+DI+4]. Estas instrucciones cargan AX en base a la posición de memoria de un segmento de datos. La primera instrucción utiliza una dirección formada mediante la suma de ARRAY, BX y DI, y la segunda mediante la suma de BX, DI y 4.) En los microprocesadores 80386 y superiores, MOV EAX,ARRAY[EBX+ECX] carga EAX en base a la posición de memoria del segmento de datos a la que se accede mediante la suma de ARRAY, EBX y ECX.
Direccionamiento de índice escalado	Este modo de direccionamiento sólo está disponible en los microprocesadores del 80386 al Pentium 4. El segundo registro de un par de registros se modifica mediante el factor de escala de 2x, 4x u 8x para generar la dirección de memoria del operando. (Ejemplo: una instrucción MOV EDX,[EAX+4*EBX] carga EDX en base a la posición de memoria del segmento de datos direccionada por EAX más cuatro veces EBX.) El uso de una escala permite el acceso a los datos de un arreglo de memoria de tipo palabra (2x), doble palabra (4x) o palabra cuádruple (8x). También existe un factor de escala de 1x, pero por lo general es implícito y no aparece explícitamente en la instrucción. La instrucción MOV AL,[EBX+ECX] es un ejemplo en el que el factor de escala es uno. Esta instrucción puede escribirse de manera alterna como MOV AL,[EBX+1*ECX]. Otro ejemplo es la instrucción MOV AL,[2*EBX], que utiliza sólo un registro escalado para direccionar memoria.

Direccionamiento de registros

Este modo de direccionamiento es el más común para direccionar datos y es el más fácil de aplicar una vez que se aprenden los nombres de los registros. El microprocesador contiene los siguientes nombres de registros de 8 bits que se utilizan en el direccionamiento de registros: AH, AL, BH, BL, CH, CL, DH y DL. También están presentes los siguientes nombres de registros de 16 bits: AX, BX, CX, DX, SP, BP, SI y DI. En los microprocesadores 80386 y superiores, los nombres de los registros extendidos de 32 bits son: EAX, EBX, ECX, EDX, ESP, EBP, EDI y ESI. Con el direccionamiento de registros, algunas instrucciones MOV y las instrucciones PUSH y POP también utilizan los nombres de los registros de segmento de 16 bits (CS, ES, DS, SS, FS y GS). Es importante que las instrucciones utilicen registros del mismo tamaño. *Nunca* se debe mezclar un registro de 8 bits con uno de 16 bits, uno de 8 bits con uno de 32 bits o uno de 16 bits con uno de 32 bits, ya que el microprocesador no lo permite y se produce un error al ensamblar. Esto es cierto incluso aunque parezca que una instrucción MOV AX,AL (MOV EAX, AL) tenga sentido. Desde luego que *no* se permiten las instrucciones MOV AX,AL o MOV EAX,AL, ya que estos registros son de distintos tamaños. Hay que tener en cuenta que unas cuantas instrucciones (como SHL DX,CL) son la excepción a esta regla, como veremos en capítulos posteriores. También es importante considerar que *ninguna* de las instrucciones MOV afecta a los bits de bandera.

La tabla 3-1 muestra muchas variaciones de las instrucciones para mover registros. Es imposible mostrar todas las combinaciones ya que hay demasiadas. Por ejemplo, tan sólo el subconjunto de 8 bits de la instrucción MOV tiene 64 variaciones distintas. Una instrucción MOV para mover registros de segmento a segmento es el único tipo de instrucción MOV que *no* se permite. Por lo general, el registro del segmento de código *no* cambia debido a una instrucción MOV ya que la dirección de la siguiente instrucción se encuentra mediante el uso de IP/EIP y de CS. Si sólo cambiara CS, la dirección de la siguiente instrucción sería impredecible. Por lo tanto, no se permite cambiar el registro CS con una instrucción MOV.

La figura 3-3 muestra la operación de la instrucción MOV BX,CX. Observe que el contenido del registro de origen no cambia, pero el contenido del registro de destino sí. Esta instrucción *mueve* (*copia*) un 1234H del registro CX al registro BX. Esto *borra* el contenido anterior (76AFH) del registro BX, pero el contenido de CX permanece sin cambio. El contenido del registro de destino o de la posición de memoria de destino cambian para todas las instrucciones, excepto para CMP y TEST. La instrucción MOV BX,CX no afecta a los 16 bits más a la izquierda del registro EBX.

El ejemplo 3-1 muestra una secuencia de instrucciones ensambladas que copian varios datos entre los registros de 8, 16 y 32 bits. Como se explicó anteriormente, el acto de mover datos de un registro a otro modifica sólo el contenido del registro de destino, nunca modifica el contenido del registro

TABLA 3-1 Ejemplos de instrucciones de direccionamiento de registros.

Lenguaje ensamblador	Tamaño	Operación
MOV AL,BL	8 bits	Copia BL en AL.
MOV CH,CL	8 bits	Copia CL en CH.
MOV AX,CX	16 bits	Copia CX en AX.
MOV SP,BP	16 bits	Copia BP en SP.
MOV DS,AX	16 bits	Copia AX en DS.
MOV SI,DI	16 bits	Copia DI en SI.
MOV BX,ES	16 bits	Copia ES en BX.
MOV ECX,EBX	32 bits	Copia EBX en ECX.
MOV ESP,EDX	32 bits	Copia EDX en ESP.
MOV DS,CX	16 bits	Copia CX en DS.
MOV ES,DS	—	No se permite (de segmento a segmento).
MOV BL,DX	—	No se permite (mezcla de tamaños).
MOV CS,AX	—	No se permite (el registro de segmento de código no puede ser el registro de destino).

FIGURA 3-3 El efecto de ejecutar la instrucción MOV BX,CX en el punto justo antes de que cambie el registro BX. Observe que sólo cambian los 16 dígitos más a la derecha del registro EBX.



de origen. La última instrucción en este ejemplo (MOV CS,AX) se ensambla sin error, pero ocasiona problemas si se ejecuta. Si sólo cambia el contenido de CS sin cambiar IP, el siguiente paso en el programa se desconoce y, por consecuencia, el programa empieza a fallar.

EJEMPLO 3-1

```

0000 8B C3      MOV AX,BX    ; copia el contenido de BX en AX
0002 8A CE      MOV CL,DH    ; copia el contenido de DH en CL
0004 8A CD      MOV CL,CH    ; copia el contenido de CH en CL
0006 66|8B C3    MOV EAX,EBX ; copia el contenido de EBX en EAX
0009 66|8B D8    MOV EBX,EAX ; copia el contenido de EAX en EBX
000C 66|8B C8    MOV ECX,EAX ; copia el contenido de EAX en ECX
000F 66|8B D0    MOV EDX,EAX ; copia el contenido de EAX en EDX
0012 8C C8      MOV AX,CS    ; copia CS en DS (dos pasos)
0014 8E D8      MOV DS,AX    ; copia AX en DS
0016 8E C8      MOV CS,AX    ; copia AX en CS (ocasiona problemas)

```

Direccionamiento inmediato

Otro de los modos de direccionamiento de datos es el direccionamiento inmediato. El término *inmediato* implica que los datos siguen inmediatamente del código de operación (opcode) hexadecimal en la memoria. Los datos inmediatos son **datos constantes**, mientras que los datos que se transfieren de un registro o una posición de memoria son **datos variables**. El direccionamiento inmediato opera en base a un byte o una palabra de datos. En los microprocesadores del 80386 al Pentium 4 el direccionamiento inmediato también opera sobre datos tipo doble palabra. La instrucción MOV inmediata transfiere una copia de los datos inmediatos hacia un registro o una posición de memoria. La figura 3-4 muestra la operación de una instrucción MOV EAX,13456H. Esta instrucción copia el número 13456H de la instrucción (ubicado en la posición de memoria inmediata al código de operación hexadecimal) en el registro EAX. Al igual que con la instrucción MOV ilustrada en la figura 3-3, los datos de origen sobreescriben a los datos de destino.

En un lenguaje ensamblador simbólico, el símbolo # se coloca antes de los datos inmediatos en algunos ensambladores. La instrucción MOV AX,#3456H es un ejemplo. La mayoría de los ensambladores no utilizan el símbolo #, sino que representan los datos inmediatos como en la instrucción MOV AX,3456H. En este libro no utilizaremos el símbolo # para los datos inmediatos. Los ensambladores

FIGURA 3-4 La operación de la instrucción MOV EAX,3456H. Esta instrucción copia los datos inmediatos (13456H) en EAX.

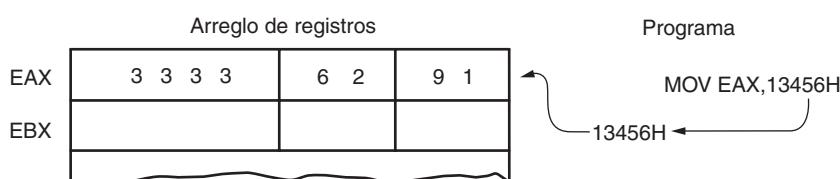


TABLA 3-2 Ejemplos de direccionamiento inmediato mediante el uso de la instrucción MOV.

Lenguaje ensamblador	Tamaño	Operación
MOV BL,44	8 bits	Copia el 44 decimal (2CH) en BL.
MOV AX,44H	16 bits	Copia el 0044H en AX.
MOV SI,0	16 bits	Copia el 0000H en SI.
MOV CH,100	8 bits	Copia el 100 decimal (64H) en CH.
MOV AL,'A'	8 bits	Copia el carácter ASCII A en AL.
MOV AX,'AB'	16 bits	Copia los caracteres ASCII BA* en AX.
MOV CL,11001110B	8 bits	Copia el 11001110 binario en CL.
MOV EBX,12340000H	32 bits	Copia el 12340000H en EBX.
MOV ESI,12	32 bits	Copia el 12 decimal en ESI.
MOV EAX,100B	32 bits	Copia el 100 binario en EAX.

*Éste no es un error. Los caracteres ASCII se almacenan como BA, por lo que hay que tener cuidado al utilizar pares de caracteres ASCII del tamaño de una palabra.

más comunes (Intel ASM, Microsoft MASM² y Borland TASM³) no utilizan el símbolo # para los datos inmediatos, pero un ensamblador antiguo que se utiliza con el sistema de desarrollo lógico de Hewlett-Packard sí lo utiliza, al igual que otros.

El ensamblador simbólico representa los datos inmediatos de muchas formas. La letra H se agrega después de los datos hexadecimales. Si estos datos empiezan con una letra, el ensamblador requiere que empiecen con un 0. Por ejemplo, para representar el número F2 hexadecimal, se utiliza 0F2H en el lenguaje ensamblador. En algunos ensambladores (aunque no en MASM, TASM o en este libro) los datos hexadecimales se representan con 'h, como en MOV AX,#h1234. Los datos decimales se representan como están y no requieren códigos especiales ni ajustes. (Un ejemplo es el 100 decimal en la instrucción MOV AL,100.) Uno o varios caracteres en código ASCII pueden representarse en el formato inmediato si los datos ASCII van encerrados entre apóstrofos. (Un ejemplo es la instrucción MOV BH,'A', que mueve una letra A en código ASCII [41H] al registro BH.) Tenga cuidado de utilizar el apóstrofo (') para los datos ASCII y no la comilla ('). Los datos binarios se representan si el número binario va seguido de la letra B o, en algunos ensambladores, la letra Y. La tabla 3-2 muestra muchas variaciones distintas de instrucciones MOV que aplican datos inmediatos.

El ejemplo 3-2 muestra varias instrucciones inmediatas en un programa corto en lenguaje ensamblador que coloca el número 0000H en los registros de 16 bits AX, BX y CX. Esto va seguido de instrucciones que utilizan direccionamiento de registros para copiar el contenido de AX en los registros SI, DI y BP. Éste es un programa completo que utiliza modelos de programación para su ensamblaje y ejecución con MASM. La instrucción .MODEL TINY hace que el ensamblador ensamble el programa en un solo segmento de código. La instrucción o directiva .CODE indica el inicio del segmento de código; la instrucción .STARTUP indica la instrucción inicial en el programa; y la instrucción .EXIT hace que el programa salga a DOS. La instrucción END indica el final del archivo de programa. Este programa está ensamblado con MASM y se ejecuta con CodeView⁴ (CV) para ver su ejecución. Para almacenar el programa en el sistema, utilice el programa EDIT de DOS, Windows NotePad⁵ o Programmer's Work-Bench⁶ (PWB). Un programa TINY siempre se ensambla como un programa de comandos (.COM).

EJEMPLO 3-2

```
.MODEL TINY ;selecciona el modelo de un solo segmento
0000      ;inicio del segmento de código
.CODE      ;inicio del programa
.STARTUP
```

²MASM (MACRO ensamblador) es una marca registrada de Microsoft Corporation.

³TASM (Turbo ensamblador) es una marca registrada de Borland Corporation.

⁴CodeView es una marca registrada de Microsoft Corporation.

⁵NotePad es una marca registrada de Microsoft Corporation.

⁶Programmer's WorkBench es una marca registrada de Microsoft Corporation.

```

0100 B8 0000      MOV AX,0          ;coloca el 0000H en AX
0103 BB 0000      MOV BX,0          ;coloca el 0000H en BX
0106 B9 0000      MOV CX,0          ;coloca el 0000H en CX

0109 8B F0        MOV SI,AX       ;copia AX en SI
010B 8B F8        MOV DI,AX       ;copia AX en DI
010D 8B E8        MOV BP,AX       ;copia AX en BP

.EXIT
END               ;salida al DOS
                  ;fin del programa

```

Cada instrucción en un programa en lenguaje ensamblador consiste de cuatro partes de campos, como se muestra en el ejemplo 3-3. El campo de más a la izquierda se llama *etiqueta*. Se utiliza para almacenar un nombre simbólico para la posición de memoria que representa. Todas las etiquetas deben empezar con una letra o con uno de los siguientes caracteres especiales: @, \$, - o ?. Una etiqueta puede tener cualquier longitud de 1 a 35 caracteres. La etiqueta aparece en un programa para identificar el nombre de una posición de memoria para almacenar datos y para otros fines que se explicarán según se vayan utilizando. El siguiente campo a la derecha se llama *campo de código de operación*; está diseñado para guardar la instrucción, o código de operación (opcode). La parte representada por MOV de la instrucción para mover datos es un ejemplo de un código de operación. A la derecha del campo del código de operación está el *campo del operando*, el cual contiene información utilizada por el código de operación. Por ejemplo, la instrucción MOV AL,BL tiene el código de operación MOV y los operandos AL y BL. Algunas instrucciones contienen entre cero y tres operandos. El campo final, llamado *campo de comentarios*, contiene un comentario sobre una instrucción o un grupo de instrucciones. Un comentario siempre debe empezar con un signo de punto y coma (;).

EJEMPLO 3-3

ETIQUETA	CÓDIGO DE OPERACIÓN	OPERANDO	COMENTARIO
DATOS1	DB	23H	;define DATOS1 como un byte de valor 23H
DATOS2	DW	1000H	;define DATOS2 como una palabra de valor 1000H
INICIO:	MOV	AL, BL	;copia BL en AL
	MOV	BH, AL	;copia AL en BH
	MOV	CX, 200	;copia 200 en CX

Cuando se ensambla el programa y se ve el archivo lista (.LST), aparece como el programa que se muestra en el ejemplo 3-2. El número hexadecimal en el extremo izquierdo es la dirección de desplazamiento de la instrucción o de los datos. El ensamblador genera este número. El número o números a la derecha de la dirección de desplazamiento son las instrucciones o los datos en código máquina que también genera el ensamblador. Por ejemplo, si la instrucción MOV AX,0 aparece en un archivo y se ensambla, aparece en la posición de memoria de desplazamiento 0100 en el ejemplo 3-2. Su formato en lenguaje máquina hexadecimal es B8 0000. El B8 es el código de operación en lenguaje máquina y el 0000 corresponde a los datos de 16 bits con un valor de cero. Cuando se escribió el programa, sólo la instrucción MOV AX,0 se escribió en el editor; el ensamblador generó el código máquina y las direcciones, y guardó el programa en un archivo con la extensión .LST. Todos los programas que se muestran en este libro están en el formato generado por el ensamblador.

EJEMPLO 3-4

```

int miFuncion(int temp)
{
    __asm
    {
        mov eax,temp
        add eax,20h
        mov temp,eax
    }
    return temp;
}

```

Los programas también pueden escribirse mediante el uso del ensamblador en línea en algunos programas en Visual C++. El ejemplo 3-4 muestra una función en un programa de Visual C++ que incluye algo de código escrito con el ensamblador en línea. Esta función suma 20H al número devuelto por la función. Observe que el código ensamblador accede a la variable temp de C++ y que todo el código ensamblador está colocado en un bloque de código `_asm`. En este libro hay muchos ejemplos escritos con el ensamblador en línea dentro de un programa en C++.

Direccionamiento directo de datos

La mayoría de las instrucciones pueden usar el modo de direccionamiento directo de datos. De hecho, este modo se aplica a muchas instrucciones en un programa común. Hay dos formas básicas de direccionamiento directo de datos: (1) **direccionamiento directo**, que se aplica a una instrucción MOV entre una posición de memoria y AL, AX o EAX, y (2) **direccionamiento por desplazamiento**, que se aplica a casi cualquier instrucción en el conjunto de instrucciones. En cualquier caso, la dirección se forma mediante la suma del desplazamiento a la dirección de segmento de datos predeterminada o a una dirección de segmento alternativa.

Direccionamiento directo. El direccionamiento directo con una instrucción MOV transfiere datos entre una posición de memoria (ubicada dentro del segmento de datos) y el registro AL (8 bits), AX (16 bits) o EAX (32 bits). Una instrucción MOV que utiliza este tipo de direccionamiento es por lo general de 3 bytes. (En los microprocesadores 80386 y superiores puede aparecer un prefijo de tamaño del registro antes de la instrucción, haciendo que se exceda de tres bytes en longitud.)

La instrucción MOV AL,DATOS que representan la mayoría de los ensambladores carga AL en base a la posición de memoria del segmento de datos DATOS (1234H). La posición de memoria DATOS es una posición de memoria simbólica, y 1234H es la posición hexadecimal actual. La posición [1234H] es una posición absoluta de memoria que no se permite en todos los programas ensambladores⁷. Algunos ensambladores pueden requerir que se forme como MOV AL,DS:[1234H] para mostrar que la dirección está en el segmento de datos. La figura 3-5 muestra cómo esta instrucción transfiere una copia del contenido del tamaño de un byte de la posición de memoria 11234H hacia AL. La dirección efectiva se forma sumando 1234H (la dirección de desplazamiento) y 10000H (la dirección de segmento de datos 1000H por 10H) en un sistema que opere en modo real.

La tabla 3-3 lista las tres instrucciones direccionalas en forma directa. Estas instrucciones aparecen a menudo en los programas, por lo que Intel decidió hacerlas instrucciones especiales de tres bytes para reducir la longitud de los programas. Todas las demás instrucciones que mueven datos de una posición de memoria a un registro (llamadas **instrucciones con direccionamiento por desplazamiento**) requieren de cuatro o más bytes de memoria para almacenarse en un programa.

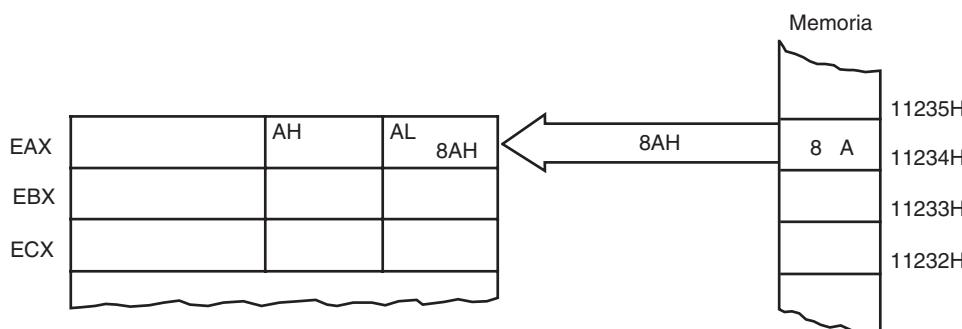


FIGURA 3-5 La operación de la instrucción MOV AL,[1234H] cuando DS = 1000H.

⁷Esta forma puede escribirse en un programa en MASM, pero aparece con mayor frecuencia cuando se ejecuta la herramienta de depuración.

TABLA 3-3 Instrucciones con direccionamiento directo que utilizan EAX, AX y AL.

Lenguaje ensamblador	Tamaño	Operación
MOV AL,NUMERO	8 bits	Copia en AL el contenido tipo byte de la posición de memoria NUMERO del segmento de datos.
MOV AX,VACA	16 bits	Copia en AX el contenido tipo palabra de la posición de memoria VACA del segmento de datos.
MOV EAX,AGUA*	32 bits	Copia en EAX el contenido tipo doble palabra de la posición AGUA del segmento de datos.
MOV NOTICIAS,AL	8 bits	Copia AL en la posición de memoria NOTICIAS tipo byte.
MOV AHI, AX	16 bits	Copia AX en la posición de memoria AHI tipo palabra.
MOV HOGAR,EAX*	32 bits	Copia EAX en la posición de memoria HOGAR tipo doble palabra.
MOV ES:[2000H],AL	8 bits	Copia AL en la memoria de segmento extra en la dirección de desplazamiento 2000H.

*Los microprocesadores del 80386 al Pentium 4 utilizan más de tres bytes de memoria para las instrucciones de 32 bits.

Direccionamiento por desplazamiento. Este modo de direccionamiento es casi idéntico al direccionamiento directo, excepto que la instrucción es de cuatro bytes en vez de tres. En los microprocesadores del 80386 al Pentium 4, esta instrucción puede ser de hasta siete bytes si se especifica tanto un registro de 32 bits como un desplazamiento de 32 bits. Este tipo de direccionamiento directo de datos es mucho más flexible, ya que la mayoría de las instrucciones lo utilizan.

Si la operación de la instrucción MOV CL,DS:[1234H] se compara con la operación de la instrucción MOV AL,DS:[1234] de la figura 3-5, podemos ver que ambas realizan básicamente la misma operación con la excepción del registro de destino (CL en vez de AL). Otra diferencia sólo se vuelve aparente al examinar las versiones ensambladas de estas dos instrucciones. La instrucción MOV AL,DS:[1234H] es de tres bytes y la instrucción MOV CL,DS:[1234H] es de cuatro bytes, como se muestra en el ejemplo 3-5. Este ejemplo muestra cómo el ensamblador convierte estas dos instrucciones en lenguaje máquina hexadecimal. Se debe incluir el registro de segmento DS: en este ejemplo, antes de la parte [desplazamiento] de la instrucción. Se puede utilizar cualquier registro de segmento pero, en la mayoría de los casos los datos se almacenan en el segmento de datos, por lo que en este ejemplo utilizamos DS:[1234H].

EJEMPLO 3-5

0000 A0 1234 R	MOV AL, DS : [1234H]
0003 BA 0E 1234 R	MOV CL, DS : [1234H]

La tabla 3-4 lista algunas instrucciones MOV que utilizan el modo de direccionamiento directo. No se listan todas las variaciones ya que hay muchas instrucciones MOV de este tipo. Los registros de segmento pueden almacenarse o cargarse desde la memoria.

El ejemplo 3-6 muestra un programa corto en el que se utilizan modelos que direccionan información en el segmento de datos. Observe que el **segmento de datos** empieza con una instrucción .DATA para informar al ensamblador en dónde inicia. El tamaño del modelo se ajusta de TINY, como se muestra en el ejemplo 3-3, a SMALL para que pueda incluirse un segmento de datos. El **modelo SMALL** permite un segmento de datos y un segmento de código. El modelo SMALL se utiliza por lo general siempre que se requieren datos de la memoria para un programa. Un programa con el modelo SMALL se ensambla como archivo de programa ejecutable (.EXE). Observe cómo este ejemplo asigna las posiciones de memoria en el segmento de datos mediante el uso de las directivas DB y DW. Aquí la instrucción .STARTUP no sólo indica el inicio del código, sino que también carga el registro del segmento de datos con la dirección del segmento de datos. Si este programa se ensambla y se ejecuta con CodeView,

TABLA 3-4 Ejemplos de direccionamiento directo de datos mediante el uso de un desplazamiento.

Lenguaje ensamblador	Tamaño	Operación
MOV CH,PERRO	8 bits	Copia en CH el contenido tipo byte de la posición de memoria PERRO del segmento de datos.
MOV CH,DS:[1000H]*	8 bits	Copia en CH el contenido tipo byte de la dirección de memoria 1000H del segmento de datos.
MOV ES,DATOS6	16 bits	Copia en ES el contenido tipo palabra de la posición de memoria DATOS6 del segmento de datos.
MOV DATOS7,BP	16 bits	Copia BP en la posición de memoria DATOS7 del segmento de datos.
MOV NUMERO,SP	16 bits	Copia SP en la posición de memoria NUMERO del segmento de datos.
MOV DATOS1,EAX	32 bits	Copia EAX en la posición de memoria DATOS1 del segmento de datos.
MOV EDI,SUMA1	32 bits	Copia en EDI el contenido tipo doble palabra de la posición de memoria SUMA1 del segmento de datos.

*Esta forma de direccionamiento se utiliza raras veces con la mayoría de los ensambladores, ya que es muy raro acceder a una dirección de desplazamiento numérica.

las instrucciones pueden verse a medida que se ejecutan y modifican los registros y las posiciones de memoria.

EJEMPLO 3-6

```

        .MODEL SMALL      ;selecciona el modelo pequeño (SMALL)
0000          .DATA      ;inicio del segmento de datos
                    DATOS1 DB 10H    ;coloca un 10H en DATOS1
0001 00        DATOS2 DB 0     ;coloca un 00H en DATOS2
0002 0000      DATOS3 DW 0     ;coloca un 0000H en DATOS3
0004 AAAA      DATOS4 DW 0AAAHH ;coloca un AAAAH en DATOS4

0000          .CODE      ;inicio del segmento de código
                    .STARTUP   ;inicio del programa

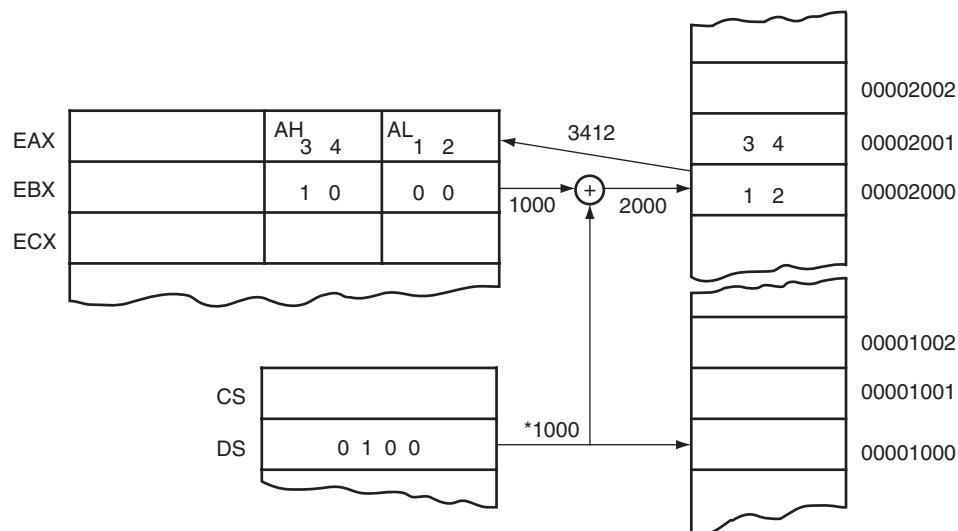
0017 A0 0000 R  MOV AL,DATA1  ;copia DATOS1 en AL
001A 8A 26 0001 R  MOV AH,DATA2  ;copia DATOS2 en AH
001E A3 0002 R  MOV DATA3,AX   ;copia AX en DATOS3
0021 8B 1E 0004 R  MOV BX,DATA4  ;copia DATOS4 en BX

                    .EXIT      ;salida al DOS
END             ;fin del listado del programa

```

Direccionamiento indirecto de registros

Este modo de direccionamiento permite que se direccionen datos en cualquier posición de memoria, a través de una dirección de desplazamiento que se guarde en cualquiera de los siguientes registros: BP, BX, DI y SI. Por ejemplo, si el registro BX contiene 1000H y se ejecuta la instrucción MOV AX,[BX], el contenido tipo palabra de la dirección de desplazamiento 1000H del segmento de datos se copia en el registro AX. Si el microprocesador se opera en el modo real y DS = 0100H, esta instrucción direcciona una palabra almacenada en los bytes de memoria 2000H y 2001H, y la transfiere hacia el registro AX (vea la figura 3-6). El contenido de la memoria 2000H se mueve hacia AL y el contenido de la memoria 2001H se mueve hacia AH. Los símbolos [] denotan un direccionamiento indirecto en lenguaje ensamblador. Además de utilizar los registros BP, BX, DI y SI para direccionar la memoria en forma indirecta,



*Después de adjuntar un 0 a DS.

FIGURA 3-6 La operación de la instrucción `MOV AX,[BX]` cuando `BX = 1000H` y `DS = 0100H`. Observe que la instrucción se muestra una vez que se ha transferido el contenido de la memoria hacia AX.

los microprocesadores 80386 y superiores permiten el direccionamiento indirecto de registros con cualquier registro extendido, excepto ESP. En la tabla 3-5 se muestran algunas instrucciones comunes en las que se utiliza el direccionamiento indirecto.

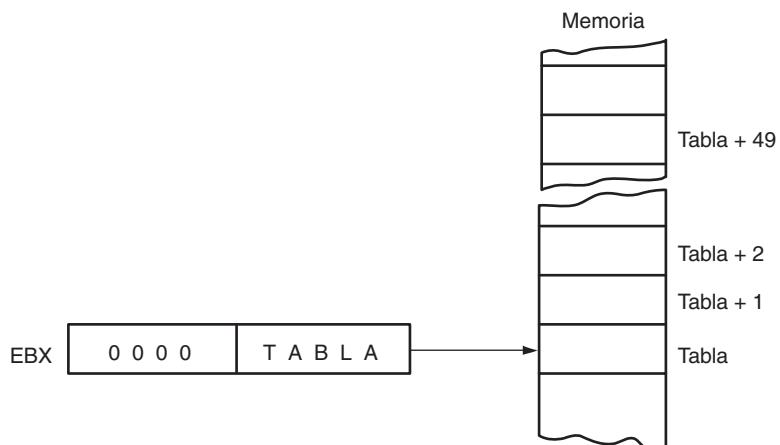
El **segmento de datos** se utiliza de manera predeterminada con el direccionamiento indirecto de registros o con cualquier otro modo de direccionamiento que utilice a BX, DI o SI para direccionar memoria. Si el registro BP direcciona memoria, se utiliza el **segmento de pila** de manera predeterminada. Estos ajustes se consideran como predeterminados para estos cuatro registros índice y base. Para los microprocesadores 80386 y superiores, EBP direcciona la memoria en el segmento de pila de manera predeterminada; EAX, EBX, ECX, EDX, EDI y ESI direccionan memoria en el segmento de datos de manera predeterminada. Al utilizar un registro de 32 bits para direccionar memoria en modo real, el

TABLA 3-5 Ejemplos de direccionamiento indirecto de registros.

Lenguaje ensamblador	Tamaño	Operación
<code>MOV CX,[BX]</code>	16 bits	Copia en CX el contenido tipo palabra de la posición de memoria del segmento de datos direccionado por BX.
<code>MOV [BP],DL*</code>	8 bits	Copia DL en la posición de memoria del segmento de pila direccionado por BP.
<code>MOV [DI],BH</code>	8 bits	Copia BH en la posición de memoria del segmento de datos direccionado por DI.
<code>MOV [DI],[BX]</code>	—	La transferencia de memoria a memoria no se permite excepto con instrucciones tipo cadena.
<code>MOV AL,[EDX]</code>	8 bits	Copia en AL el contenido tipo byte de la posición de memoria del segmento de datos direccioando por EDX.
<code>MOV ECX,[EBX]</code>	32 bits	Copia en ECX el contenido tipo doble palabra de la posición de memoria del segmento de datos diseccionado por EBX.

*Los datos direccionados por BP o EBP están en el segmento de pila forzosamente, mientras otras instrucciones direccionadas de forma indirecta usan el segmento de datos obligatoriamente.

FIGURA 3–7 Arreglo (TABLA) y registro BX utilizado para direccionamiento indirecto de 50 muestras.



contenido del registro de 32 bits no debe exceder de 0000FFFFH. En el modo protegido puede usarse cualquier valor en un registro de 32 bits que se utilice para direccionar memoria en forma indirecta, siempre y cuando no acceda a una posición fuera del segmento, como según lo indicado por el byte de derechos de acceso. Un ejemplo de una instrucción para 80386-Pentium 4 es MOV EAX,[EBX]. Esta instrucción carga EAX con el número de tamaño de doble palabra que se almacena en la dirección de desplazamiento del segmento de datos indicada por EBX.

En algunos casos el direccionamiento indirecto requiere que se especifique el tamaño de los datos. Este tamaño se especifica mediante la **directiva especial del ensamblador** BYTE PTR, WORD PTR o DWORD PTR. Estas directivas indican el tamaño de los datos de la memoria direccionados por el apuntador de memoria (PTR). Por ejemplo, la instrucción MOV AL,[DI] es evidentemente una instrucción para mover datos del tamaño de un byte, pero la instrucción MOV [DI],10H es ambigua, ya que no se sabe si direcciona una posición de memoria tipo byte, palabra o doble palabra. El ensamblador no puede determinar el tamaño del número 10H. La instrucción MOV BYTE PTR [DI],10H designa con claridad la posición direccionada por DI como una posición de memoria del tamaño de un byte. De igual forma, la instrucción MOV DWORD PTR [DI],10H identifica claramente la posición de memoria como del tipo doble palabra. Las directivas BYTE PTR, WORD PTR y DWORD PTR se utilizan solamente en instrucciones que direccionan una posición de memoria a través de un apuntador o un registro índice con datos inmediatos, y para otras instrucciones que describiremos en los siguientes capítulos. Otra directiva que se usa en ocasiones es QWORD PTR, en donde QWORD es una palabra cuádruple (64 bits). Si los programas utilizan las instrucciones SIMD, la directiva QWORD PTR se utiliza también para representar un número de 128 bits.

A menudo el direccionamiento indirecto permite que un programa haga referencia a datos tabulares ubicados en el sistema de memoria. Por ejemplo, suponga que debe crear una tabla de información que contenga 50 muestras tomadas de la posición de memoria 0000:046C. Esta posición de memoria contiene un contador en el DOS que se mantiene en base al reloj en tiempo real de la computadora. La figura 3-7 muestra la tabla y el registro BX utilizado para direccionar en forma secuencial cada posición de la tabla. Para lograr esta tarea se carga la posición inicial de la tabla en el registro BX con una instrucción MOV inmediata. Después de inicializar la dirección inicial de la tabla, se utiliza el direccionamiento indirecto de registros para almacenar las 50 muestras en forma secuencial.

La secuencia mostrada en el ejemplo 3-7 carga el registro BX con la dirección inicial de la tabla e inicializa el contador, ubicado en el registro CX, con 50. La directiva **OFFSET** indica al ensamblador que debe cargar el registro BX con la dirección de desplazamiento de la posición de memoria TABLA, no con el contenido de TABLA. Por ejemplo, la instrucción MOV BX,DATOS copia el contenido de la posición de memoria DATOS en BX, mientras que la instrucción MOV BX,OFFSET DATOS copia la dirección de desplazamiento DATOS en BX. Cuando se utiliza la directiva OFFSET con la instrucción MOV, el ensamblador calcula la dirección de desplazamiento y después utiliza una instrucción MOV inmediata para cargar la dirección en el registro de 16 bits especificado.

EJEMPLO 3-7

```

        .MODEL SMALL
0000          .DATA      ;selecciona el modelo pequeño
                      ;inicio del segmento de datos

0000 0032 [      DATOS DW    50 DUP(?) ;establece un arreglo de 50 palabras
              0000
              ]
0000          .CODE      ;inicio del segmento de código
                      .STARTUP ;inicio del programa
0017 B8 0000    MOV AX, 0
001A 8E C0      MOV ES, AX ;direcciona el segmento 0000 con ES
001C B8 0000 R   MOV BX,OFFSET DATOS ;direcciona el arreglo DATOS con BX
001F B9 0032    MOV CX,50  ;carga el contador con 50
0022          DENUEVO:
0022 26:A1 046C  MOV AX,ES:[046CH] ;obtiene el valor del reloj
0026 89 07      MOV [BX],AX  ;guarda el valor del reloj en DATAS
0028 43         INC BX   ;incrementa BX al siguiente elemento
0029 43         INC BX
002A E2 F6      LOOP AGAIN ;se repite 50 veces

                      .EXIT      ;salida al DOS
END                 ;fin del listado del programa

```

Una vez que se inicializan el contador y el apuntador, se ejecuta un ciclo repetir-hasta que CX = 0. Aquí los datos se leen de la posición de memoria 46CH del segmento extra con la instrucción MOV AX,ES:[046CH] y se almacenan en la memoria que está direccionada en forma indirecta por la dirección de desplazamiento ubicada en el registro BX. Después se incrementa BX (se le suma 1) dos veces para direccionar la siguiente palabra en la tabla. Por último, la instrucción LOOP repite el ciclo 50 veces. La instrucción LOOP decrementa (resta 1 de) el contador (CX); si CX no es igual a cero, LOOP provoca un salto a la posición de memoria DENUEVO. Si CX se hace cero no se produce ningún salto y termina esta secuencia de instrucciones. Este ejemplo copia los 50 valores más recientes del reloj en el arreglo de memoria DATOS. Este programa mostrará con frecuencia los mismos datos en cada posición, ya que el contenido del reloj se cambia sólo 18.2 veces por segundo. Para ver el programa y su ejecución, utilice el programa CodeView. Para usar CodeView, escriba CV XXXX.EXE, en donde XXXX.EXE es el nombre del programa que se va a depurar. También puede utilizarlo como DEBUG desde el programa Programmer's WorkBench, bajo el menú RUN. Tenga en cuenta que CodeView sólo funciona con archivos .COM o .EXE. Algunos modificadores útiles para CodeView son /50 para una pantalla de 50 líneas y /S para el uso de pantallas de vídeo de alta resolución en una aplicación. Para depurar el archivo PRUEBA.COM con 50 líneas, escriba CV /50 /S PRUEBA.COM en el símbolo del sistema.

Direccionamiento de base más índice

Este modo de direccionamiento es similar al direccionamiento indirecto, ya que direcciona los datos de la memoria en forma indirecta. En los microprocesadores del 8086 al 80286 este tipo de direccionamiento utiliza un registro base (BP o BX) y un registro índice (DI o SI) para direccionar la memoria de forma indirecta. A menudo el registro base guarda la posición inicial de un arreglo de memoria, mientras que el registro índice guarda la posición relativa de un elemento en el arreglo. Recuerde que siempre que BP direcciona datos de la memoria, tanto el registro de segmento como BP generan la dirección efectiva.

En los microprocesadores 80386 y superiores, este tipo de direccionamiento permite la combinación de cualesquiera dos registros extendidos de 32 bits, excepto ESP. Por ejemplo, la instrucción MOV DL,[EAX+EBX] es un ejemplo que utiliza EAX (como la base) más EBX (como el índice). Si se utiliza el registro EBP, los datos se localizan en el segmento de pila en vez de estar en el segmento de datos.

Localización de datos con el direccionamiento de base más índice. La figura 3-8 muestra cómo la instrucción MOV DX,[BX+DI] direcciona los datos cuando el microprocesador opera en el modo real. En este ejemplo, BX = 1000H, DI = 0010H y DS = 0100H, lo cual se traduce en la posición de memoria 02010H. Esta instrucción transfiere una copia de la palabra de la posición 02010H hacia el registro DX.

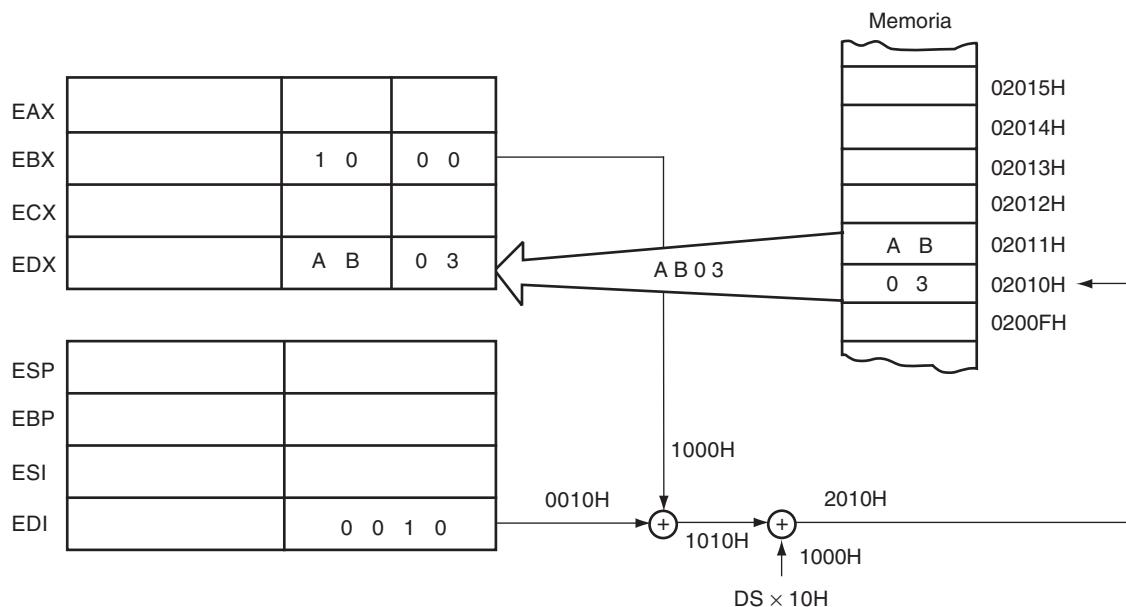


FIGURA 3-8 Un ejemplo que muestra cómo funciona el direccionamiento de base más índice para la instrucción `MOV DX,[BX+DI]`. Observe que se accede a la dirección de memoria 02010H, ya que DS = 0100H, BX = 100H y DI = 0010H.

La tabla 3-6 lista algunas instrucciones que se utilizan para el direccionamiento de base más índice. El ensamblador Intel requiere que este modo de direccionamiento aparezca como `[BX][DI]` en vez de `[BX+DI]`. La instrucción `MOV DX,[BX+DI]` es `MOV DX,[BX][DI]` para un programa escrito para el ensamblador Intel ASM. En este libro utilizamos la primera forma en todos los programas de ejemplo, pero la segunda forma puede utilizarse en muchos ensambladores, incluyendo MASM de Microsoft. Las instrucciones como `MOD DI,[BX+DI]` se ensamblarán, pero no se ejecutarán correctamente.

TABLA 3-6 Ejemplos de direccionamiento de base más índice.

Lenguaje ensamblador	Tamaño	Operación
<code>MOV CX,[BX+DI]</code>	16 bits	Copia en CX el contenido tipo palabra de la posición de memoria del segmento de datos direccionado por BX más DI.
<code>MOV CH,[BP+SI]</code>	8 bits	Copia en CH el contenido tipo byte de la posición de memoria del segmento de datos direccionado por BP más SI.
<code>MOV [BX+SI],SP</code>	16 bits	Copia SP en la posición de memoria del segmento de datos direccionado por BX más SI.
<code>MOV [BP+DI],AH</code>	8 bits	Copia AH en la posición de memoria del segmento de datos direccionado por BP más DI.
<code>MOV CL,[EDX+EDI]</code>	8 bits	Copia en CL el contenido tipo byte de la posición de memoria del segmento de datos direccionado por EDX más EDI.
<code>MOV [EAX+EBX],ECX</code>	32 bits	Copia ECX en la posición de memoria del segmento de datos direccionado por EAX más EBX.

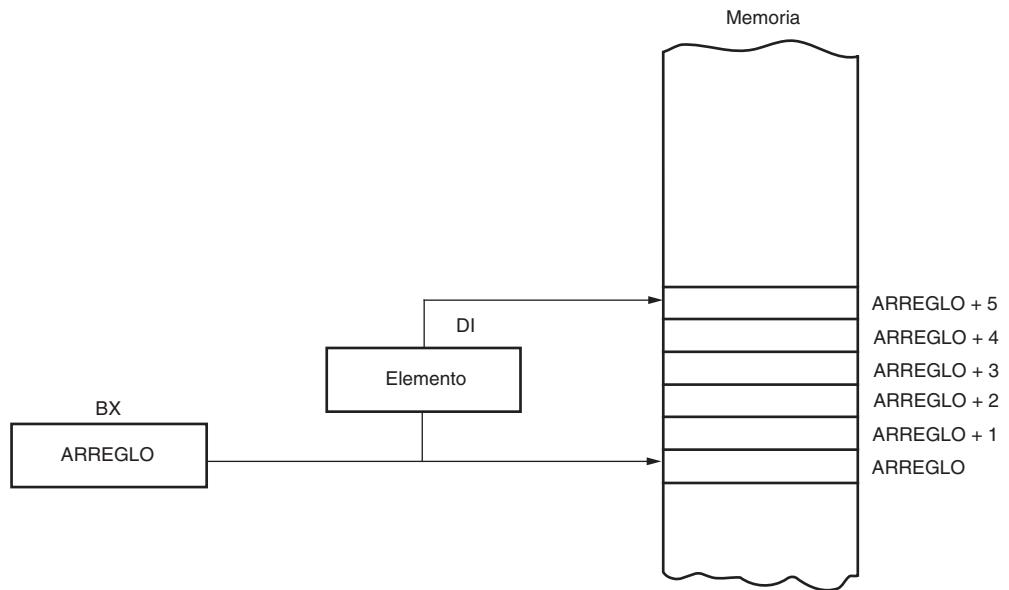


FIGURA 3-9 Un ejemplo del modo de direccionamiento de base más índice. Aquí se direcciona un elemento (DI) de un ARREGLO (BX).

Localización de datos de un arreglo mediante el direccionamiento de base más índice. Un uso importante del modo de direccionamiento de base más índice es para direccionar elementos en un arreglo de memoria. Suponga que se debe tener acceso a los elementos en un arreglo ubicado en el segmento de datos de la posición de memoria ARREGLO. Para lograr esto se carga el registro BX base con la dirección inicial del arreglo, y el registro DI (índice) con el número de elemento al que se va a tener acceso. La figura 3-9 muestra el uso de BX y DI para acceder a un elemento en un arreglo de datos. Un programa corto (listado en el ejemplo 3-8) mueve el elemento 10H del arreglo hacia el elemento 20H del arreglo. Observe que el número de elemento del arreglo, que se carga en el registro DI, direcciona el elemento del arreglo. Además observe cómo se ha inicializado el contenido del ARREGLO de forma que el elemento 10H contenga 29H.

EJEMPLO 3-8

```

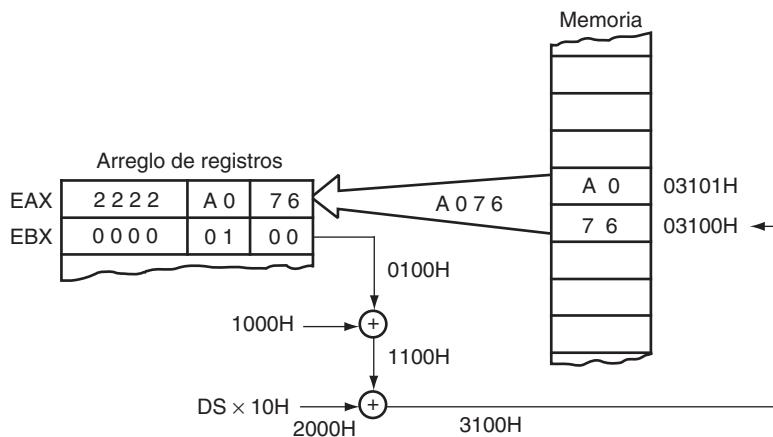
0000          .MODEL SMALL           ;selecciona el modelo pequeño
0000 0010 [     .DATA             ;inicio del segmento de datos
    ARREGLO DB 16 DUP(?)         ;establece un arreglo de 16 bytes
    00
]
0010 29         DB 29H            ;elemento 10H
0011 001E [     DB 20 dup(?)      ;elemento 20H
    00
]
0000          .CODE              ;inicio del segmento de datos
          .STARTUP

0017 B8 0000 R  MOV  BX,OFFSET ARRAY ;direcciona ARREGLO
001A BF 0010    MOV  DI,10H        ;direcciona el elemento 10H
001D 8A 01      MOV  AL,[BX+DI]   ;obtiene el elemento 10H
001F BF 0020    MOV  DI,20H        ;direcciona el elemento 20H
0022 88 01      MOV  [BX+DI],AL    ;almacena en el elemento 20H

.EXIT          ;salida al DOS
END            ;fin del programa

```

FIGURA 3-10 La operación de la instrucción MOV AX,[BX+1000H] cuando BX = 0100H y DS = 0200H.



Direcccionamiento relativo de registros

Este modo de direcccionamiento es similar al direcccionamiento de base más índice y al direcccionamiento por desplazamiento. En el direcccionamiento relativo de registros, los datos en un segmento de memoria se direccionan mediante la suma del desplazamiento al contenido de un registro base o índice (BP, BX, DI o SI). La figura 3-10 muestra la operación de la instrucción `MOV AX,[BX+1000H]`. En este ejemplo, BX = 0100H y DS = 0200H, por lo que la dirección generada es la suma de $DS \times 0H$, BX y el desplazamiento de 1000H, con lo cual se direcciona la posición 03100H. Recuerde que BX, DI o SI direccionan el segmento de datos y que BP direcciona el segmento de pila. En los microprocesadores 80386 y superiores el desplazamiento puede ser un número de 32 bits y el registro puede ser cualquier registro de 32 bits, excepto ESP. Recuerde que el tamaño de un segmento en modo real es de 64 Kbytes. La tabla 3-7 lista unas cuantas instrucciones que utilizan el direcccionamiento relativo de registros.

El desplazamiento es un número que se suma al registro dentro de los corchetes [], como en la instrucción `AL,[DI+2]`, o puede ser un desplazamiento que se sustraen del registro, como en `MOV AL,[SI-1]`. Un desplazamiento también puede ser una dirección de desplazamiento que se anexa al frente de los corchetes [], como en `MOV AL,DATA[DI]`. Ambas formas de desplazamientos también pueden aparecer de manera simultánea, como en la instrucción `MOV AL,DATA[DI+3]`. En todos los casos, ambas formas de desplazamiento se suman al registro base o al registro base más índice dentro

TABLA 3-7 Ejemplos de direcccionamiento relativo de registros.

Lenguaje ensamblador	Tamaño	Operación
<code>MOV AX,[DI+100H]</code>	16 bits	Copia en AX el contenido tipo palabra de la posición de memoria del segmento de datos direcccionado por DI más 100H.
<code>MOV ARREGLO[SI],BL</code>	8 bits	Copia BL en la posición de memoria del segmento de datos direcccionado por el ARREGLO SI.
<code>MOV LISTA[SI+2],CL</code>	8 bits	Copia CL en la posición de memoria del segmento de datos direcccionado por la suma de LISTA SI y 2.
<code>MOV DI,SET_IT[BX]</code>	16 bits	Copia en DI el contenido tipo palabra en la posición de memoria del segmento de datos direcccionado por SET_IT de BX.
<code>MOV DI,[EAX+10H]</code>	16 bits	Copia en DI el contenido tipo palabra de la posición del segmento de datos direcccionado por EAX más 10H.
<code>MOV ARREGLO[EBX],EAX</code>	32 bits	Copia EAX en la posición de memoria del segmento de datos direcccionado por ARREGLO EBX.

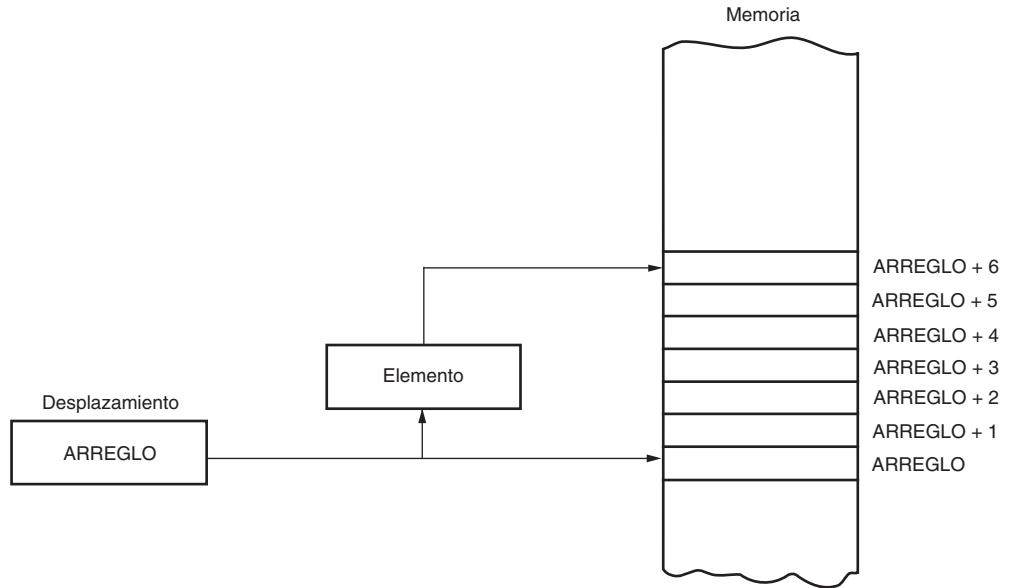


FIGURA 3-11 El direccionamiento relativo de registros usado para direccionar un elemento del ARREGLO. El desplazamiento de las direcciones de inicio del ARREGLO, y el acceso de un elemento a DI.

de los corchetes []. En los microprocesadores del 8086 al 80286, el valor de desplazamiento está limitado a un número de 16 bits con signo, con un valor entre +32,767 (7FFFH) y -32,768 (8000H); en los microprocesadores 80386 y superiores se permite un desplazamiento de 32 bits con un valor entre +2,147,483,647 (7FFFFFFFH) y -2,147,483,648 (8000000H).

Direccionamiento de un arreglo con el modo relativo de registros. Es posible direccionar datos de un arreglo con el direccionamiento relativo de registros, tal como se hace con el direccionamiento de base más índice. En la figura 3-11 se muestra el direccionamiento relativo de registros con el mismo ejemplo que para el direccionamiento de base más índice. Esto muestra cómo se suma el desplazamiento ARREGLO al registro índice DI para generar una referencia a un elemento del arreglo.

El ejemplo 3-9 muestra cómo este nuevo modo de direccionamiento puede transferir el contenido del elemento 10H del arreglo hacia el elemento 20H del mismo arreglo. Observe la similitud entre este ejemplo y el ejemplo 3-8. La principal diferencia es que en el ejemplo 3-9 no se utiliza el registro BX para direccionar la memoria ARREGLO; en vez de ello, ARREGLO se utiliza como desplazamiento para realizar la misma tarea.

EJEMPLO 3-9

```

        .MODEL SMALL           ;seleccionar modelo pequeño
0000          .DATA          ;inicio del segmento de datos
0000 0010 [      ARREGLO  DB 16 dup(?) ;establece el ARREGLO
                00
                ]
0010 29          DB 29          ;elemento 10H
0011 001E [      DB 30 dup(?) ;elemento 20H
                00
                ]
0000          .CODE          ;inicio del segmento de código
                .STARTUP       ;inicio del programa
0017 BF 0010      MOV DI,10H    ;direcciona el elemento 10H
001A 8A 85 0000 R MOV AL,ARRAY[DI] ;obtiene el elemento 10H del ARREGLO
001E BF 0020      MOV DI,20H    ;obtiene el elemento 20H
0021 88 85 0000 R MOV ARRAY[DI],AL ;se guarda en el elemento 20H
                .EXIT          ;salida al DOS
                END            ;fin del programa

```

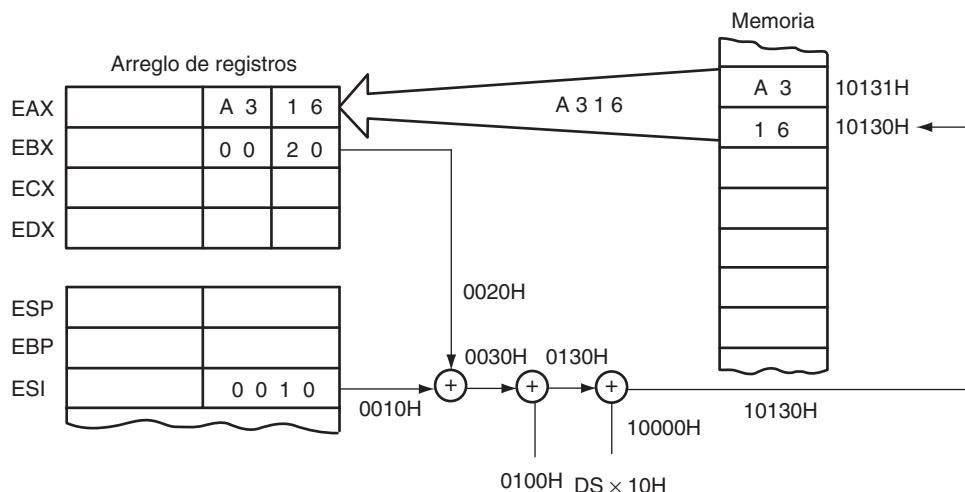


FIGURA 3-12 Un ejemplo del direccionamiento de base relativa más índice, utilizando una instrucción MOV AX,[BX+SI+100H]. Nota: DS = 1000H.

Direcciónamiento de base relativa más índice

Este modo de direcciónamiento es similar al direcciónamiento de base más índice, sólo que suma un desplazamiento además de usar un registro base y un registro índice para formar la dirección de memoria. Este tipo de modo de direcciónamiento a menudo dirige a un arreglo bidimensional de datos en la memoria.

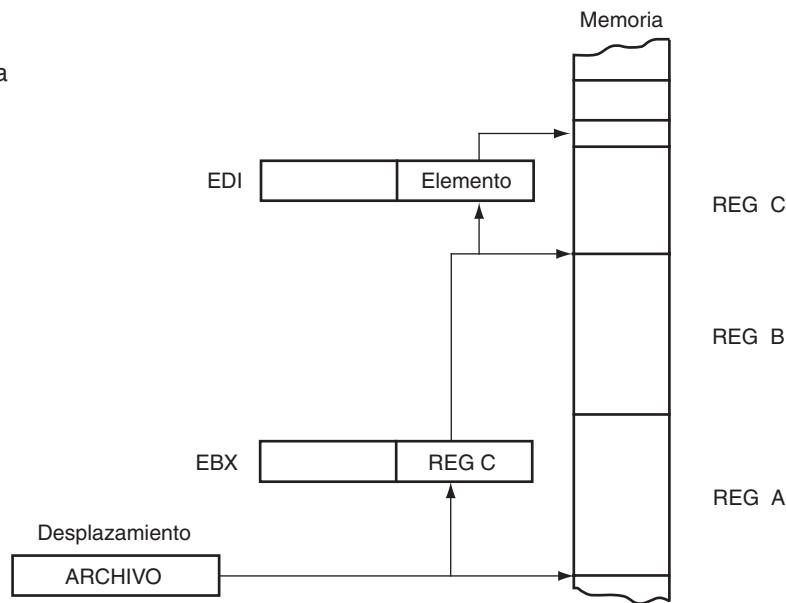
Direcciónamiento de datos con base relativa más índice. El direcciónamiento de base relativa más índice es el modo de direcciónamiento menos utilizado. La figura 3-12 muestra cómo se hace referencia a los datos si la instrucción que ejecuta el microprocesador es MOV AX,[BX+SI+100H]. El desplazamiento de 100H se suma a BX y SI para formar la dirección de desplazamiento dentro del segmento de datos. Los valores de los registros son BX = 0020H, SI = 0100H y DS = 1000H, de manera que la dirección efectiva para esta instrucción sea 10130H; la suma de estos registros más un desplazamiento de 100H. Este modo de direcciónamiento es demasiado complejo como para utilizarlo con frecuencia en la programación. En la tabla 3-8 aparecen algunas instrucciones típicas que utilizan el direcciónamiento de base relativa más índice. Con los microprocesadores 80386 y superiores, la dirección efectiva se genera mediante la suma de dos registros de 32 bits más un desplazamiento de 32 bits.

TABLA 3-8 Instrucciones de ejemplo de direcciónamiento con base relativa más índice.

Lenguaje ensamblador	Tamaño	Operación
MOV DH,[BX+DI+20H]	8 bits	Copia en DH el contenido tipo byte de la posición de memoria del segmento de datos direccionada por la suma de BX, DI y 20H.
MOV AX,ARCHIVO[BX+DI]	16 bits	Copia en AX el contenido tipo palabra de la posición de memoria del segmento de datos, direccionada por la suma de ARCHIVO, BX y DI.
MOV LISTA[BP+DI],CL	8 bits	Copia CL en la posición de memoria del segmento de pila direccionada por la suma de LISTA, BP y DI.
MOV LISTA[BP+SI+4],DH	8 bits	Copia DH en la posición de memoria del segmento de pila direccionada por la suma de LISTA, BP, SI y 4.
MOV EAX,ARCHIVO[EBX+ECX+2]	32 bits	Copia en EAX el contenido tipo doble palabra de la posición de memoria direccionada por la suma de ARCHIVO, EBX, ECX y 2.

FIGURA 3-13

Direccionamiento de base relativa más índice que se utiliza para acceder a un ARCHIVO que contiene varios registros (REG).



Direccionamiento de arreglos con el modo de base relativa más índice. Suponga que existe un archivo de muchos registros en memoria y que cada registro contiene muchos elementos. El desplazamiento direcciona el archivo, el registro base direcciona un registro y el registro índice direcciona un elemento de un registro. La figura 3-13 ilustra esta forma bastante compleja de direccionamiento.

El ejemplo 3-10 proporciona un programa que copia el elemento 0 del registro A en el elemento 2 del registro C, mediante el uso del modo de direccionamiento de registro relativo más índice. Este ARCHIVO de ejemplo contiene cuatro registros y cada registro contiene 10 elementos. Observe cómo se utiliza la instrucción THIS BYTE para definir las etiquetas ARCHIVO y REGA como la misma posición de memoria.

EJEMPLO 3-10

```

0000          .MODEL SMALL           ;selecciona el modelo pequeño
0000 = 0000      .DATA              ;inicio del segmento de datos
0000 000A [     ARCHIVO EQU THIS BYTE ;asignar ARCHIVO a este byte
                REGA   DB 10 dup(?)  ;10 bytes para el registro A
0000          00
                ]
000A 000A [     REGB   DB 10 dup(?)  ;10 bytes para el registro B
0000          00
                ]
0014 000A [     REGC   DB 10 dup(?)  ;10 bytes para el registro C
0000          00
                ]
001E 000A [     REGD   DB 10 dup(?)  ;10 bytes para el registro D
0000          00
                ]
0000          .CODE              ;inicio del segmento de código
0000          .STARTUP           ;inicio del programa
0017 BB 0000 R  MOV  BX,OFFSET REGA ;direcciona el registro A
001A BF 0000      MOV  DI,0        ;direcciona el elemento 0
001D 8A 81 0000 R MOV  AL,FILE[BX+DI] ;obtiene los datos
0021 BB 0014 R    MOV  BX,OFFSET REGC ;direcciona el registro C
0024 BF 0002      MOV  DI,2        ;direcciona el elemento 2
0027 88 81 0000 R MOV  FILE[BX+DI],AL ;almacena los datos
0000          .exit              ;salida al DOS
0000          end                ;fin del programa

```

Direccionamiento de índice escalado

Este modo de direccionamiento es el último modo de direccionamiento de datos que veremos. Este modo es único para los microprocesadores del 80386 al Pentium 4. Utiliza dos registros de 32 bits (un registro base y un registro índice) para acceder a la memoria. El segundo registro (índice) se multiplica por un factor de escala. Este factor puede ser 1×, 2×, 4× u 8×. El factor de escala 1× está implícito y no necesita incluirse en la instrucción de lenguaje ensamblador (MOV AL,[EBX+ECX]). El factor de escala 2× se utiliza para direccionar arreglos de memoria del tamaño de una palabra, el factor de escala 4× se utiliza con los arreglos de memoria del tamaño de una doble palabra y el factor de escala 8× se utiliza con los arreglos de memoria del tamaño de una palabra cuádruple.

La instrucción MOV AX,[EDI+2*ECX] es un ejemplo. Esta instrucción utiliza un factor de escala de 2×, el cual multiplica el contenido de ECX por 2 antes de sumarlo al registro EDI para formar la dirección de memoria. Si ECX contiene 00000000H, se dirige al elemento 0 del tamaño de una palabra; si ECX contiene 00000001H, se dirige al elemento 1 del tamaño de una palabra, y así sucesivamente. Esto escala el índice (ECX) por un factor de 2 para un arreglo de memoria del tamaño de una palabra. En la tabla 3-9 podrá consultar algunos ejemplos de direccionamiento de índice escalado. Como puede imaginar, hay una variedad extremadamente amplia de combinaciones de registros de direccionamiento de índice escalado. La escala también se aplica a las instrucciones que utilizan un solo registro indirecto para acceder a la memoria. La instrucción MOV EAX,[4*EDI] es una instrucción de índice escalado que utiliza un registro para acceder a la memoria de forma indirecta.

El ejemplo 3-11 muestra una secuencia de instrucciones que utilizan el direccionamiento de índice escalado para acceder a un arreglo de datos del tamaño de una palabra llamada LISTA. Observe que la dirección de desplazamiento de LISTA se carga en el registro EBX con la instrucción MOV EBX,OFFSET LISTA. Una vez que EBX dirige al arreglo LISTA, se suman los elementos (ubicados en ECX) 2, 4 y 7 de este arreglo de datos tipo palabra, utilizando un factor de 2 para acceder a los elementos. Este programa almacena el 2 (que se encuentra en el elemento 2) en los elementos 4 y 7. Observe además la directiva .386 para seleccionar el microprocesador 80386. Esta directiva debe ir después de la instrucción .MODEL para que el ensamblador procese instrucciones de 80386 para el DOS. Si se está utilizando el 80486, la directiva .486 aparece después de la instrucción .MODEL; si se utiliza el Pentium se aplica la directiva .586; y si se utiliza el Pentium Pro, el Pentium II, el Pentium III o el Pentium 4 se aplica la directiva .686. Si la directiva de selección de microprocesador aparece antes de la instrucción .MODEL, el microprocesador ejecuta instrucciones en el modo protegido de 32 bits, que debe ejecutarse en Windows.

EJEMPLO 3-11

```
.MODEL SMALL ;selecciona el modelo pequeño
.386 ;selecciona el microprocesador
80386
```

TABLA 3-9 Ejemplos de direccionamiento de índice escalado.

Lenguaje ensamblador	Tamaño	Operación
MOV EAX,[EBX+4*ECX]	32 bits	Copia en EAX el contenido tipo doble palabra de la posición de memoria del segmento de datos direccionado por la suma de 4 por ECX más EBX.
MOV [EAX+2*EDI+100H],CX	16 bits	Copia CX en la posición de memoria del segmento de datos direccionado por la suma de EAX, 100H y 2 por EDI.
MOV AL,[EBP+2*EDI+2]	8 bits	Copia en AL el contenido tipo byte de la posición de memoria del segmento de datos direccionado por la suma de EBP, 2 y 2 por EDI.
MOV EAX,ARREGLO[4*ECX]	32 bits	Copia en EAX el contenido tipo doble palabra de la posición de memoria del segmento de datos direccionado por la suma de ARREGLO y 4 por ECX.

```

0000 .DATA ;inicio del segmento de datos
0000 0000 0001 0002 LISTA DW 0,1,2,3,4 ;define el arreglo LISTA
      0003 0004
000A 0005 0006 0007 DW 5,6,7,8,9
      0008 0009

0000 .CODE ;inicio del segmento de código
0010 66|BB 00000000 R MOV EBX,OFFSET LIST ;direcciona el arreglo LISTA
0016 66|B9 00000002 MOV ECX,2 ;direcciona el elemento 2
001C 67&8B 04 4B MOV AX,EBX+2*ECX] ;obtiene el elemento 2
0020 66|B9 00000004 MOV ECX,4 ;direcciona el elemento 4
0026 67&89 04 4B MOV [EBX+2*ECX],AX ;almacena en el elemento 4
002A 66|B9 00000007 MOV ECX,7 ;direcciona el elemento 7
0030 67&89 04 4B MOV [EBX+2*ECX],AX ;almacena en el elemento 7
      .exit ;salida al DOS
      end

```

Estructuras de datos

Una estructura de datos se utiliza para especificar cómo se almacena la información en un arreglo de memoria, lo cual puede ser bastante útil con las aplicaciones que utilizan arreglos. Es mejor pensar en una estructura de datos como si fuera una plantilla para los datos. El inicio de una estructura se identifica con la directiva STRUC del lenguaje ensamblador y el final con la instrucción ENDS. En el ejemplo 3-12 se define y utiliza una estructura de datos típica. Observe que el nombre de la estructura aparece con las instrucciones STRUC y ENDS. El ejemplo muestra la estructura de datos en la forma en que se escribió, sin la versión ensamblada.

EJEMPLO 3-12

```

;definir la estructura de datos INFO
;
INFO     STRUC
NOMBRES   DB      32 dup(?)    ;reserva 32 bytes para un nombre
CALLE     DB      32 dup(?)    ;reserva 32 bytes para la dirección de la calle
CIUDAD    DB      16 dup(?)    ;reserva 16 bytes para la ciudad
ESTADO    DB      2  dup(?)    ;reserva 2 bytes para el estado
CP        DB      5  dup(?)    ;reserva 5 bytes para el código postal

INFO     ENDS

NOMBRE1   INFO <'Bob Smith',  '123 Main Street',  'Wanda',  'OH',  '44444'>
NOMBRE2   INFO <'Steve Doe',  '222 Moose Lane',  'Miller',  'PA',  '18100'>
NOMBRE3   INFO <'Jim Dover',  '303 Main Street',  'Orender', 'CA',  '90000'>

```

La estructura de datos en el ejemplo 3-12 define cinco campos de información. El primero es de 32 bytes y guarda un nombre; el segundo es de 32 bytes y guarda una dirección; el tercero es de 16 bytes para la ciudad; el cuarto es de 2 bytes para el estado; el quinto es de 5 bytes para el código postal. Una vez definida la estructura (INFO) puede llenarse, como se muestra, con nombres y direcciones. Se muestran tres ejemplos para INFO. Observe que las literales van rodeadas de apóstrofos y que todo el campo va rodeado de los símbolos < > cuando la estructura de datos se utiliza para definir datos.

Cuando se direccionan los datos en una estructura, se utiliza el nombre de la estructura y el nombre del campo para seleccionar un campo de la estructura. Por ejemplo, para direccionar la CALLE en NOMBRE2 se utiliza el operando NOMBRE2.CALLE, en donde el nombre de la estructura va primero, seguido de un punto y luego del nombre del campo. De igual forma se utiliza NOMBRE3.CIUDAD para hacer referencia a la ciudad en la estructura NOMBRE3.

En el ejemplo 3-13 aparece una secuencia corta de instrucciones que borra el campo del nombre en la estructura NOMBRE1, el campo de dirección en la estructura NOMBRE2 y el campo del código postal en la estructura NOMBRE3. En capítulos posteriores se definirá la función y operación de las instrucciones en este programa. Tal vez quiera hacer referencia a este ejemplo una vez que haya visto estas instrucciones.

EJEMPLO 3-13

```

;borrar NOMBRES en el arreglo NOMBRE1
;
0000 B9 0020      MOV CX,32
0003 B0 00      MOV AL,0
0005 BE 0000 R    MOV DI,OFFSET NOMBRE1.NOMBRES
0008 F3/AA      REP STOSB
;
;borrar CALLE en el arreglo NOMBRE2
;
000A B9 0020      MOV CX,32
000D B0 00      MOV AL,0
000F BE 0077 R    MOV DI,OFFSET NAME2.CALLE
0012 F3/AA      REP STOSB
;
;borrar CP en NOMBRE3
;
0014 B9 0005      MOV CX,5
0017 B0 00      MOV AL,0
0019 BE 0100 R    MOV DI,OFFSET NAME3.CP
001C F3/AA      REP STOSB

```

3-2**MODOS DE DIRECCIONAMIENTO DE MEMORIA DE UN PROGRAMA**

Estos modos de direccionamiento, si se utilizan con las instrucciones JMP (salto) y CALL (llamada), consisten de tres formas distintas: directo, relativo e indirecto. Esta sección presenta estas tres formas de direccionamiento y utiliza la instrucción JMP para ilustrar su operación.

Direccionamiento directo de memoria de programa

Este modo de direccionamiento es el que utilizaban muchos de los primeros microprocesadores para todos los saltos y llamadas. El direccionamiento directo de memoria de programa también se utiliza en los lenguajes de alto nivel, como las instrucciones GOTO y GOSUB del lenguaje BASIC. El microprocesador utiliza esta forma de direccionamiento, pero no con tanta frecuencia como el direccionamiento relativo e indirecto de memoria de programa.

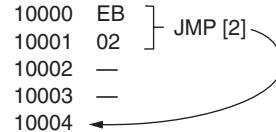
Las instrucciones para el direccionamiento directo de memoria de programa almacenan la dirección con el código de operación. Por ejemplo, si un programa salta a la posición de memoria 10000H para la siguiente instrucción, la dirección (1000H) se almacena después del código de operación en la memoria. La figura 3-14 muestra la instrucción JMP directa intersegmentos y los cuatro bytes requeridos para almacenar la dirección 1000H. Esta instrucción JMP carga CS con 1000H e IP con 0000H para saltar a la posición de memoria 10000H para la siguiente instrucción. (Un **salto intersegmentos** es un salto a cualquier posición de memoria dentro de todo el sistema de memoria completo.) El salto directo se llama a menudo *salto lejano* (*far jump*) porque puede saltar a cualquier posición de memoria para la siguiente instrucción. En el modo real, un salto lejano accede a cualquier posición dentro del primer 1 Mbyte de memoria, ya que se cambia tanto CS como IP. En la operación en modo protegido, el salto lejano accede a un nuevo descriptor del segmento de código proveniente de la tabla de descriptores, lo que le permite saltar a cualquier posición de memoria dentro de todo el rango de direcciones de 4 Gbytes, en los microprocesadores del 80386 al Pentium 4.

La otra instrucción que utiliza el direccionamiento directo de programa es la instrucción CALL entre segmentos o lejana. Por lo general, el nombre de una dirección de memoria (a la que se le llama

FIGURA 3-14 Los 5 byte de la versión del lenguaje de máquina de una instrucción JMP [10000H].

Código de operación	Compensación (baja)	Compensación (alta)	Segmento (bajo)	Segmento (alto)
E A	0 0	0 0	0 0	1 0

FIGURA 3-15 Una instrucción JMP [2]. Esta instrucción salta dos bytes de memoria que siguen a la instrucción JMP.



etiqueta) hace referencia a la posición que se llama o a la que se salta, en vez de la dirección numérica actual. Al utilizar una etiqueta con la instrucción CALL o JMP, la mayoría de los ensambladores seleccionan la mejor forma de direccionamiento de programa.

Direccionamiento relativo de memoria de programa

Este modo de direccionamiento no está disponible en todos los primeros microprocesadores, pero sí está disponible en esta familia de microprocesadores. El término **relativo** significa “relativo al apuntador de instrucciones (IP)”. Por ejemplo, si una instrucción JMP omite los siguientes dos bytes de memoria, la dirección en relación con el apuntador de instrucciones es un 2 que se suma al apuntador de instrucciones. Con esto se desarrolla la dirección de la siguiente instrucción del programa. En la figura 3-15 se muestra un ejemplo de la instrucción JMP relativa. Observe que esta instrucción es de un byte, con un desplazamiento de uno o dos bytes que se suma al apuntador de instrucciones. El desplazamiento de un byte se utiliza en saltos **short** y el desplazamiento de dos bytes se utiliza en saltos y en llamadas **near**. Ambos tipos se consideran como saltos intrasegmentos. (Un **salto intrasegmentos** es un salto hacia cualquier parte dentro del segmento de código actual.) En los microprocesadores 80386 y superiores el desplazamiento también puede ser un valor de 32 bits, lo que les permite utilizar el direccionamiento relativo en cualquier posición dentro de sus segmentos de código de 4 Gbytes.

Las instrucciones JMP y CALL relativas contienen un desplazamiento de 8 bits o uno de 16 bits, el cual permite una referencia de memoria hacia delante o una referencia de memoria hacia atrás. (Los microprocesadores 80386 y superiores pueden tener un desplazamiento de 8 bits o de 16 bits.) Todos los ensambladores calculan automáticamente la distancia para el desplazamiento y seleccionan el formato adecuado de uno, dos o cuatro bytes. Si la distancia es demasiada para un desplazamiento de dos bytes en los microprocesadores del 8086 al 80286, algunos ensambladores utilizan el salto directo. Un desplazamiento de 8 bits (*short*) tiene un rango de salto de entre +127 y -128 bytes a partir de la siguiente instrucción; un desplazamiento de 16 bits (*near*) tiene un rango de ±32K bytes. En los microprocesadores 80386 y superiores, un desplazamiento de 32 bits permite un rango de ±2 Gbytes. El desplazamiento de 32 bits sólo puede utilizarse en el modo protegido.

Direccionamiento indirecto de memoria de programa

El microprocesador permite varias formas de direccionamiento indirecto de memoria de programa para las instrucciones JMP y CALL. La tabla 3-10 lista algunas instrucciones de salto indirecto aceptables

TABLA 3-10 Ejemplos de direccionamiento indirecto de memoria de programa.

Lenguaje ensamblador	Operación
JMP AX	Salta a la posición del segmento de código actual direccionada por el contenido de AX.
JMP CX	Salta a la posición del segmento de código actual direccionada por el contenido de CX.
JMP NEAR PTR[BX]	Salta a la posición del segmento de código actual direccionada por el contenido de la posición del segmento de datos direccionada por BX.
JMP NEAR PTR[DI+2]	Salta a la posición del segmento de código actual direccionada por el contenido de la posición de memoria del segmento de datos direccionada por DI más 2.
JMP TABLA[BX]	Salta a la posición del segmento de código actual direccionada por el contenido de la posición de memoria del segmento de datos direccionada por TABLA más BX.
JMP ECX	Salta a la posición del segmento de código actual direccionada por el contenido de ECX.

FIGURA 3-16 Una tabla de saltos que almacena las direcciones de varios programas. La dirección exacta seleccionada de la TABLA se determina en base a un índice almacenado con la instrucción de salto.

TABLA	DW	LOC0
	DW	LOC1
	DW	LOC2
	DW	LOC3

en los programas, en las que se puede utilizar cualquier registro de 16 bits (AX, BX, CX, DX, SP, BP, DI o SI); cualquier registro relativo ([BP], [BX], [DI] o [SI]); y también cualquier registro relativo con un desplazamiento. En los microprocesadores 80386 y superiores también puede usarse un registro extendido para guardar la dirección o la dirección indirecta de una instrucción JMP o CALL relativa. Por ejemplo, JMP EAX salta a la posición direccionada por el registro EAX.

Si un registro de 16 bits guarda la dirección de una instrucción JMP, el salto es cercano (near). Por ejemplo, si el registro BX contiene 1000H y se ejecuta una instrucción JMP BX, el microprocesador salta a la dirección de desplazamiento 1000H en el segmento de código actual.

Si un registro relativo guarda la dirección, el salto también se considera como indirecto. Por ejemplo, JMP [BX] hace referencia a la posición de memoria dentro del segmento de datos en la dirección de desplazamiento contenida en BX. En esta dirección de desplazamiento hay un número de 16 bits que se utiliza como la dirección de desplazamiento en el salto intrasegmentos. A este tipo de salto se le denomina algunas veces como *salto indirecto-indirecto o doble-indirecto*.

La figura 3-16 muestra una tabla de saltos que se almacena empezando en la posición de memoria TABLA. El programa corto del ejemplo 3-14 hace referencia a esta tabla de saltos. En este ejemplo, el registro BX se carga con un 4 que, al combinarse en la instrucción JMP TABLA[BX] con TABLA, ocasiona que la dirección efectiva sea el contenido de la segunda entrada en la tabla de saltos de 16 bits.

EJEMPLO 3-14

```
;Uso del direccionamiento indirecto para un salto
;
0000 BB 0004      MOV  BX,4          ;direcciona LOC2
0003 FF A7 23A1 R  JMP  TABLA[BX]   ;salto a LOC2
```

3-3

MODOS DE DIRECCIONAMIENTO DE MEMORIA DE LA PILA

La pila juega un papel importante en todos los microprocesadores. Guarda los datos de manera temporal y almacena las direcciones de retorno que utilizan los procedimientos. La memoria de la pila es una memoria LIFO (**último en entrar, primero en salir**) que describe la forma en que los datos se almacenan y se sacan de la pila. Los datos se colocan en la pila con una **instrucción PUSH** y se sacan con una **instrucción POP**. La instrucción CALL también utiliza la pila para guardar la dirección de retorno para los procedimientos y una instrucción RET (retorno) para sacar la dirección de retorno de la pila.

La memoria de la pila se mantiene mediante dos registros: el apuntador a la pila (SP o ESP) y el registro de segmento de pila (SS). Siempre que se mete una palabra de datos a la pila [vea la figura 3-17(a)] se colocan los 8 bits de mayor orden en la posición direccionada por SP – 1. Los 8 bits de menor orden se colocan en la posición direccionada por SP – 2. Después SP se decrementa en 2, para que la siguiente palabra de datos se almacene en la siguiente posición de memoria de la pila que esté disponible. El registro SP/ESP siempre apunta a un área de memoria ubicada dentro del segmento de pila. El registro SP/ESP se suma a SS × 10H para formar la dirección de memoria de la pila en el modo real. En la operación en modo protegido, el registro SS guarda un selector que accede a un descriptor para la dirección base del segmento de pila.

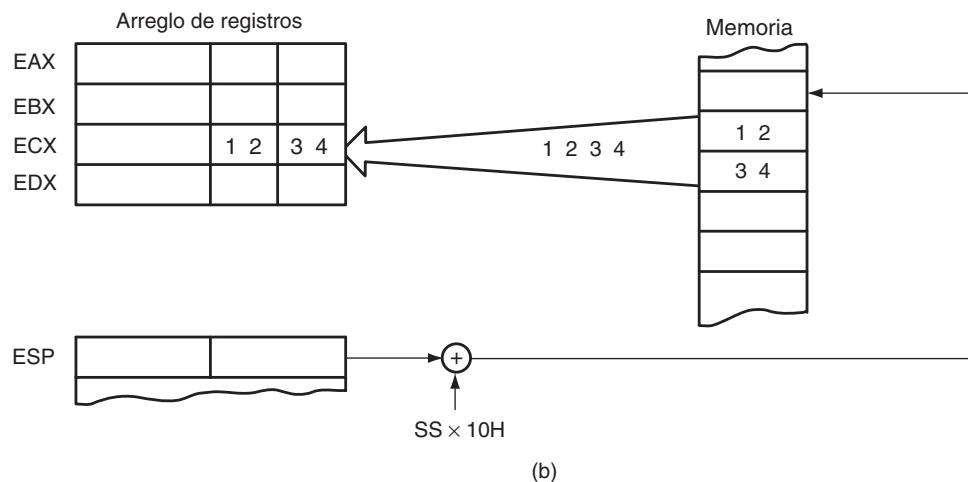
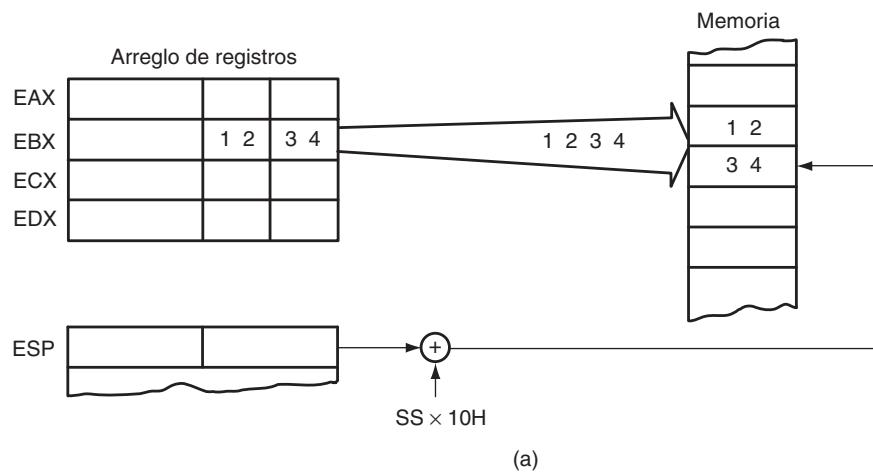


FIGURA 3-17 Las instrucciones PUSH y POP: (a) PUSH BX coloca el contenido de BX en la pila; (b) POP CX saca los datos de la pila y los coloca en CX. Ambas instrucciones se muestran después de su ejecución.

Siempre que se sacan datos de la pila [vea la figura 3-17(b)] se sacan los 8 bits de menor orden de la posición direccionada por SP. Los 8 bits de mayor orden se sacan de la posición direccionada por SP + 1. Después el registro SP se incrementa en 2. La tabla 3-11 lista algunas de las instrucciones PUSH y POP disponibles para el microprocesador. Observe que PUSH y POP almacenan o recuperan palabras de datos (nunca bytes) en los microprocesadores del 8086 al 80286. Los microprocesadores 80386 y superiores permiten la transferencia de palabras o dobles palabras hacia y desde la pila. Los datos pueden meterse en la pila desde cualquier registro de 16 bits o registro de segmento; en los microprocesadores 80386 y superiores, los datos pueden meterse en la pila desde cualquier registro extendido de 32 bits. Los datos pueden sacarse de la pila y meterse en cualquier registro o cualquier registro de segmento, excepto CS. La razón por la cual no pueden sacarse datos de la pila y colocarse en CS es que sólo se cambia parte de la dirección de la siguiente instrucción.

Las instrucciones PUSHA o POPA meten o sacan todos los registros (excepto los registros de segmento) en la pila. Estas instrucciones no están disponibles en los primeros procesadores 8086/8088. La instrucción PUSH inmediata también es nueva en los microprocesadores del 80286 al Pentium 4.

TABLA 3-11 Instrucciones PUSH y POP de ejemplo.

<i>Lenguaje ensamblador</i>	<i>Operación</i>
POPF	Saca una palabra de la pila y la coloca en el registro de banderas.
POPFD	Saca una doble palabra de la pila y la coloca en el registro EFLAG.
PUSHF	Copia el registro de banderas en la pila.
PUSHFD	Copia el registro EFLAG en la pila.
PUSH AX	Copia el registro AX en la pila.
POP BX	Saca una palabra de la pila y la coloca en el registro BX.
PUSH DS	Copia el registro DS en la pila.
PUSH 1234H	Copia el número 1234H (tipo doble palabra) en la pila.
POP CS	Esta instrucción es ilegal.
PUSH WORD PTR[BX]	Copia en la pila el contenido tipo palabra de la posición de memoria del segmento de datos direccionada por BX.
PUSHA	Copia AX, CX, DX, BX, SP, BP, DI y SI en la pila.
POPA	Saca de la pila el contenido tipo palabra para los siguientes registros: SI, DI, BP, SP, BX, DX, CX y AX.
PUSHAD	Copia EAX, ECX, EDX, EBX, ESP, EBP, EDI y ESI en la pila.
POPAD	Saca de la pila el contenido tipo doble palabra para los siguientes registros: ESI, EDI, EBP, ESP, EBX, EDX, ECX y EAX.
POP EAX	Saca una doble palabra de la pila y la coloca en el registro EAX.
PUSH EDI	Copia EDI en la pila.

Observe los ejemplos en la tabla 3-11, que muestran el orden de los registros transferidos por las instrucciones PUSHA y POPA. Los microprocesadores 80386 y superiores también permiten que se saquen o se metan registros extendidos.

El ejemplo 3-15 muestra un programa corto que mete el contenido de AX, BX y CX en la pila. La primera instrucción POP recupera el valor que se metió en la pila desde CX y lo coloca en AX. La segunda instrucción POP coloca el valor original de BX en CX. La última instrucción POP coloca el valor de AX en BX.

EJEMPLO 3-15

```

        .MODEL TINY      ;selecciona el modelo diminuto (tiny)
0000          .CODE      ;inicio del segmento de código
                  .STARTUP   ;inicio del programa
0100 B8 1000  MOV  AX,1000H ;carga los datos de prueba
0103 BB 2000  MOV  BX,2000H
0106 B9 3000  MOV  CX,3000H

0109 50          PUSH  AX      ;mete el 1000H en la pila
010A 53          PUSH  BX      ;mete el 2000H en la pila
010B 51          PUSH  CX      ;mete el 3000H en la pila

010C 58          POP   AX      ;mete el 3000H en AX
010D 59          POP   CX      ;mete el 2000H en CBX
010E 5B          POP   BX      ;mete el 1000H en BX
                  .exit      ;salida al DOS
                  end       ;fin del programa

```

3-4**RESUMEN**

- Los modos de direccionamiento de datos son: de registro, inmediato, directo, de registro indirecto, de base más índice, de registro relativo y de base relativa más índice. Los microprocesadores del 80386 al Pentium 4 tienen un modo de direccionamiento adicional, llamado direccionamiento de índice escalado.
- Los modos de direccionamiento de memoria de un programa son: directo, relativo e indirecto.

TABLA 3-12

Ejemplos de modos de direccionamiento de datos en modo real.

<i>Lenguaje ensamblador</i>	<i>Generación de direcciones</i>
MOV AL,BL	Direccionamiento de registros de 8 bits
MOV AX,BX	Direccionamiento de registros de 16 bits
MOV EAX,ECX	Direccionamiento de registros de 32 bits
MOV DS,DX	Direccionamiento de registros de segmento
MOV AL,LISTA	(DS × 10H) + LISTA
MOV CH,DATOS1	(DS × 10H) + DATOS1
MOV ES,DATOS2	(DS × 10H) + DATOS2
MOV AL,12	Datos inmediatos de 12
MOV AL,[BP]	(SS × 10H) + BP
MOV AL,[BX]	(DS × 10H) + BX
MOV AL,[DI]	(DS × 10H) + DI
MOV AL,[SI]	(DS × 10H) + SI
MOV AL,[BP+2]	(DS × 10H) + BP + 2
MOV AL,[BX-4]	(DS × 10H) + BX - 4
MOV AL,[DI+1000H]	(DS × 10H) + DI + 1000H
MOV AL,[SI+300H]	(DS × 10H) + SI + 300H
MOV AL,LISTA[BP]	(SS × 10H) + LISTA + BP
MOV AL,LISTA[BX]	(DS × 10H) + LISTA + BX
MOV AL,LISTA[DI]	(DS × 10H) + LISTA + DI
MOV AL,LISTA[SI]	(DS × 10H) + LISTA + SI
MOV AL,LISTA[BP+2]	(SS × 10H) + LISTA + BP + 2
MOV AL,LISTA[BX-6]	(DS × 10H) + LISTA + BX - 6
MOV AL,LISTA[DI+100H]	(DS × 10H) + LISTA + DI + 100H
MOV AL,LISTA[SI+200H]	(DS × 10H) + LISTA + SI + 200H
MOV AL,[BP+DI]	(SS × 10H) + BP + DI
MOV AL,[BP+SI]	(SS × 10H) + BP + SI
MOV AL,[BX+DI]	(DS × 10H) + BX + DI
MOV AL,[BX+SI]	(DS × 10H) + BX + SI
MOV AL,[BP+DI+8]	(SS × 10H) + BP + DI + 8
MOV AL,[BP+SI-8]	(SS × 10H) + BP + SI - 8
MOV AL,[BX+DI+10H]	(DS × 10H) + BX + DI + 10H
MOV AL,[BX+SI-10H]	(DS × 10H) + BX + SI - 10H
MOV AL,LISTA[BP+DI]	(SS × 10H) + LISTA + BP + DI
MOV AL,LISTA[BP+SI]	(SS × 10H) + LISTA + BP + SI
MOV AL,LISTA[BX+DI]	(DS × 10H) + LISTA + BX + DI
MOV AL,LISTA[BX+SI]	(DS × 10H) + LISTA + BX + SI
MOV AL,LISTA[BP+DI+2]	(SS × 10H) + LISTA + BP + DI + 2
MOV AL,LISTA[BP+SI-7]	(SS × 10H) + LISTA + BP + SI - 7
MOV AL,LISTA[BX+DI+3]	(DS × 10H) + LISTA + BX + DI + 3
MOV AL,LISTA[BX+SI-2]	(DS × 10H) + LISTA + BX + SI - 2

3. La tabla 3-12 lista todos los modos de direccionamiento de datos en modo real disponibles para los microprocesadores del 8086 al 80286. Hay que tener en cuenta que los microprocesadores 80386 y superiores también utilizan estos modos, además de los otros que definimos en este capítulo. En el modo protegido, la función del registro de segmento es direccionar un descriptor que contenga la dirección base del segmento de memoria.
4. Los microprocesadores del 80386 al Pentium 4 tienen modos de direccionamiento adicionales que permiten a los registros EAX, EBX, ECX, EDX, EBP, EDI y ESI direccionar memoria. Aunque estos modos de direccionamiento son demasiado numerosos como para listarlos en formato tabular, en general cualquiera de estos registros funciona de la misma forma que los que se listan en la tabla 3-12. Por ejemplo, MOV AL,TABLA[EBX+2*ECX+10H] es un modo de direccionamiento válido para los microprocesadores del 80386 al Pentium 4.
5. La instrucción MOV copia el contenido del operando de origen en el operando de destino. El origen nunca cambia para ninguna instrucción.
6. El direccionamiento de registros especifica a cualquier registro de 8 bits (AH, AL, BH, BL, CH, CL, DH o DL) o a cualquier registro de 16 bits (AX, BX, CX, DX, SP, BP, SI o DI). Los registros de segmento (CS, DS, ES o SS) también pueden direccionarse para mover datos entre un registro de segmento y un registro/posición de memoria de 16 bits, o para PUSH y POP. En los microprocesadores del 80386 al Pentium 4 también se utilizan los registros extendidos para el direccionamiento de registros; estos registros extendidos son: EAX, EBX, ECX, EDX, ESP, EBP, EDI y ESI. También están disponibles los registros de segmento FS y GS en los microprocesadores 80386 y superiores.
7. La instrucción MOV inmediata transfiere hacia un registro o posición de memoria el byte o palabra que sigue inmediatamente después del código de operación. El direccionamiento inmediato manipula datos constantes en un programa. En los microprocesadores 80386 y superiores también pueden cargarse datos inmediatos tipo doble palabra en un registro o en una posición de memoria de 32 bits.
8. La instrucción .MODEL se utiliza en el lenguaje ensamblador para identificar el inicio de un archivo y el tipo de modelo de memoria que se va a utilizar. Si el tamaño es TINY, el programa existe en un segmento (el segmento de código) y se ensambla como un programa de comandos (.COM). Si se utiliza el modelo SMALL, el programa usa un segmento de código y un segmento de datos, y se ensambla como programa ejecutable (.EXE). En el apéndice A se listan otros tamaños de modelos y sus atributos.
9. En el microprocesador, el direccionamiento directo ocurre en dos formas: (1) direccionamiento directo y (2) direccionamiento por desplazamiento. Ambas formas de direccionamiento son idénticas, excepto que el direccionamiento directo se utiliza para transferir datos entre EAX, AX o AL y la memoria; el direccionamiento por desplazamiento se utiliza con cualquier transferencia entre la memoria y los registros. El direccionamiento directo requiere tres bytes de memoria, mientras que el direccionamiento por desplazamiento requiere cuatro bytes. Algunas de estas instrucciones en los microprocesadores 80386 y superiores pueden requerir bytes adicionales en la forma de prefijos para los tamaños de los registros y los operandos.
10. El direccionamiento indirecto de registros permite que los datos se direccionen en la posición de memoria a la que apunta un registro base (BP y BX) o un registro índice (DI y SI). En los microprocesadores 80386 y superiores se utilizan los registros extendidos EAX, EBX, ECX, EDX, EBP, EDI y ESI para direccionar los datos de la memoria.
11. A menudo, el direccionamiento de base más índice dirige los datos en un arreglo. La dirección de memoria para este modo se forma mediante la suma de un registro base, un registro índice y el contenido de un registro de segmento multiplicado por 10H. En los microprocesadores 80386 y superiores, los registros base e índice pueden ser cualquier registro de 32 bits, excepto EIP y ESP.
12. El direccionamiento relativo de registros utiliza un registro base o índice más un desplazamiento para acceder a los datos en la memoria.
13. El direccionamiento de base relativa más índice es útil para direccionar un arreglo de memoria bidimensional. La dirección se forma mediante la suma de un registro base, un registro índice, un desplazamiento y el contenido de un registro de segmento multiplicado por 10H.

14. El direccionamiento de índice escalado es único para los microprocesadores del 80386 al Pentium 4. El segundo de dos registros (índice) se escala basándose en un factor de $2\times$, $4\times$ u $8\times$ para acceder a las palabras, dobles palabras o palabras cuádruples en los arreglos de memoria. Las instrucciones `MOV AX,[EBX+2*ECX]` y `MOV [4*ECX],EDX` son ejemplos de direccionamiento de índice escalado.
15. Las estructuras de datos son plantillas para almacenar arreglos de datos, y se direccionan basándose en el nombre y el campo del arreglo. Por ejemplo, el arreglo `NUMERO` y el campo `DIEZ` del arreglo `NUMERO` se direccionan como `NUMERO.DIEZ`.
16. El direccionamiento directo de memoria de un programa se permite con las instrucciones `JMP` y `CALL` en cualquier posición del sistema de memoria. Con este modo de direccionamiento, la dirección de desplazamiento y la dirección de segmento se almacenan con la instrucción.
17. El direccionamiento relativo de memoria de un programa permite que una instrucción `JMP` o `CALL` se ramifique hacia delante o hacia atrás en el segmento de código actual por ± 32 Kbytes. En los microprocesadores 80386 y superiores, el desplazamiento de 32 bits permite una ramificación hacia cualquier posición en el segmento de código actual mediante el uso de un valor de desplazamiento de ± 2 Gbytes. El desplazamiento de 32 bits sólo puede usarse en modo protegido.
18. El direccionamiento indirecto de memoria de un programa permite que las instrucciones `JMP` o `CALL` direccionen otra porción de un programa o de una subrutina en forma indirecta, a través de una posición de memoria o un registro.
19. Las instrucciones `PUSH` y `POP` transfieren una palabra entre la pila y un registro o una posición de memoria. Hay una instrucción `PUSH` inmediata disponible para colocar datos inmediatos en la pila. Las instrucciones `PUSHA` y `POPA` transfieren los registros `AX`, `CX`, `, BX, BP, SP, SI y DI entre la pila y estos registros. En los microprocesadores 80386 y superiores también pueden transferirse los registros extendidos y las banderas extendidas entre los registros y la pila. Una instrucción PUSHFD almacena el registro EFLAGS, mientras que una instrucción PUSHF almacena el registro FLAGS.`

3-5**PREGUNTAS Y PROBLEMAS**

1. ¿Qué hacen las siguientes instrucciones `MOV`?
 - (a) `MOV AX,BX`.
 - (b) `MOV BX,AX`.
 - (c) `MOV BL,CH`.
 - (d) `MOV ESP,EBP`.
 - (e) `MOV AX,CS`.
2. Liste los registros de 8 bits que se utilizan para el direccionamiento de registros.
3. Liste los registros de 16 bits que se utilizan para el direccionamiento de registros.
4. Liste los registros de 32 bits que se utilizan para el direccionamiento de registros en los microprocesadores del 80386 al Pentium 4.
5. Liste los registros de segmento de 16 bits que se utilizan para el direccionamiento de registros mediante las instrucciones `MOV`, `PUSH` y `POP`.
6. ¿Qué error tiene la instrucción `MOV BL,CX`?
7. ¿Qué error tiene la instrucción `MOV DS,SS`?
8. Seleccione una instrucción para cada una de las siguientes tareas:
 - (a) copiar `EBX` en `EDX`.
 - (b) copiar `BL` en `CL`.
 - (c) copiar `SI` en `BX`.
 - (d) copiar `DS` en `AX`.
 - (e) copiar `AL` en `AH`.

9. Seleccione una instrucción para cada una de las siguientes tareas:
 - (a) mover 12H hacia AL.
 - (b) mover 123AH hacia AX.
 - (c) mover 0CDH hacia CL.
 - (d) mover 1000H hacia SI.
 - (e) mover 1200A2H hacia EBX.
10. ¿Qué símbolo especial se utiliza algunas veces para denotar datos inmediatos?
11. ¿Cuál es el propósito de la instrucción .MODEL TINY?
12. ¿Qué directiva en lenguaje ensamblador indica el inicio del segmento de código?
13. ¿Qué es una etiqueta?
14. ¿En qué campo de una instrucción se coloca la instrucción MOV?
15. ¿Con qué caracteres puede empezar una etiqueta?
16. ¿Cuál es el propósito de la directiva .EXIT?
17. ¿La instrucción .MODEL TINY provoca que un programa se ensamble como ejecutable (.EXE)?
18. ¿Qué tareas realiza la directiva .STARTUP en el modelo pequeño de memoria?
19. ¿Qué es un desplazamiento? ¿Cómo determina la dirección de memoria en una instrucción MOV DS:[2000H],AL?
20. ¿Qué indican los símbolos []?
21. Suponga que DS = 0200H, BX = 0300H y DI = 400H. Determine la dirección de memoria a la que accede cada una de las siguientes instrucciones, suponiendo la operación en modo real:
 - (a) MOV AL,[1234H].
 - (b) MOV EAX,[BX].
 - (c) MOV [DI],AL.
22. ¿Qué error tiene la instrucción MOV [BX],[DI]?
23. Seleccione una instrucción que requiere BYTE PTR.
24. Seleccione una instrucción que requiere WORD PTR.
25. Seleccione una instrucción que requiere DWORD PTR.
26. Explique la diferencia entre la instrucción MOV BX,DATOS y la instrucción MOV BX,OFFSET DATOS.
27. Suponga que DS = 1000H, SS = 2000H, BP = 1000H y DI = 0100H. Determine la dirección de memoria a la que accede cada una de las siguientes instrucciones, suponiendo la operación en modo real:
 - (a) MOV AL,[BP+DI].
 - (b) MOV CX,[DI].
 - (c) MOV EDX,[BP].
28. ¿Qué error tiene la instrucción MOV AL,[BX][SI], si acaso hay uno?
29. Suponga que DS = 1200H, BX = 0100H y SI = 0250H. Determine la dirección a la que accede cada una de las siguientes instrucciones, suponiendo la operación en modo real:
 - (a) MOV [100H],DL.
 - (b) MOV [SI+100H],EAX.
 - (c) MOV DL,[BX+100H].
30. Suponga que DS = 1100H, BX = 0200H, LISTA = 0250H y SI = 0500H. Determine la dirección a la que accede cada una de las siguientes instrucciones, suponiendo la operación en modo real:
 - (a) MOV LISTA[SI],EDX.
 - (b) MOV CL,LISTA[BX+SI].
 - (c) MOV CH,[BX+SI].
31. Suponga que DS = 1300H, SS = 1400H, BP = 1500H y SI = 0100H. Determine la dirección a la que accede cada una de las siguientes instrucciones, suponiendo la operación en modo real:
 - (a) MOV EAX,[BP+200H].
 - (b) MOV AL,[BP+SI-200H].
 - (c) MOV AL,[SI-0100H].
32. ¿Qué registro base direcciona los datos en el segmento de pila?

33. Suponga que EAX = 00001000H, EBX = 00002000H y DS = 0010H. Determine la dirección a la que accede cada una de las siguientes instrucciones, suponiendo la operación en modo real:
 - (a) MOV ECX,[EAX+EBX].
 - (b) MOV [EAX+2*EBX],CL.
 - (c) MOV DH,[EBX+4*EAX+1000H].
34. Desarrolle una estructura de datos llamada CAMPOS que tenga cinco campos de una palabra cada uno, llamados F1, F2, F3, F4 y F5.
35. Muestre cómo el campo F3 de la estructura de datos construida en la pregunta 34 se direcciona en un programa.
36. ¿Cuáles son los tres modos de direccionamiento de memoria de un programa?
37. ¿Cuántos bytes de memoria almacenan una instrucción de salto directo lejana (far)? ¿Qué se almacena en cada uno de esos bytes?
38. ¿Cuál es la diferencia entre un salto intersegmentos y un salto intrasegmentos?
39. Si un salto cercano (near) utiliza un desplazamiento de 16 bits con signo, ¿cómo puede saltar a cualquier posición de memoria dentro del segmento de código actual?
40. Los microprocesadores 80386 y superiores utilizan un desplazamiento de _____ bits para saltar a cualquier posición dentro del segmento de código de 4 Gbytes.
41. ¿Qué es un salto lejano (far)?
42. Si una instrucción JMP se almacena en la posición de memoria 100H dentro del segmento de código actual, no puede ser un salto _____ si salta a la posición de memoria 200H dentro del segmento de código actual.
43. Muestre qué instrucción JMP se ensambla (short, near o far) si la instrucción JMP AHI se almacena en la dirección de memoria 10000H y la dirección de AHI es:
 - (a) 10020H
 - (b) 11000H
 - (c) 0FFEHE
 - (d) 30000H
44. Forme una instrucción JMP que salte a la dirección a la que apunta el registro BX.
45. Seleccione una instrucción JMP que salte a la posición almacenada en memoria, en la posición TABLA. Suponga que es un salto cercano (near).
46. ¿Cuántos bytes se almacenan en la pila mediante una instrucción PUSH AX?
47. Explique cómo funciona la instrucción PUSH [DI].
48. ¿Qué registros se colocan en la pila mediante la instrucción PUSHA? ¿En qué orden?
49. ¿Qué hace la instrucción PUSHAD?
50. ¿Qué instrucción coloca el registro EFLAGS en la pila, en el microprocesador Pentium 4?

CAPÍTULO 4

Instrucciones para mover datos

INTRODUCCIÓN

Este capítulo se concentra en las instrucciones para mover datos, que son: MOV, MOVSX, MOVZX, PUSH, POP, BSWAP, XCHG, XLAT, IN, OUT, LEA, LDS, LES, LFS, LGS, LSS, LAHF, SAHF y las instrucciones de cadena MOVS, LODS, STOS, INS y OUTS. La instrucción de transferencia de datos implementada de manera más reciente en los microprocesadores del Pentium Pro al Pentium 4 es CMOV (MOV condicional). Este capítulo presenta primero las instrucciones para mover datos ya que se utilizan con más frecuencia en los programas y son fáciles de entender.

El microprocesador requiere un programa ensamblador que genere lenguaje máquina, ya que las instrucciones en lenguaje máquina son demasiado complejas para generarlas de forma manual con eficiencia. Este capítulo describe la sintaxis del lenguaje ensamblador y algunas de sus directivas. [En este libro asumiremos que el usuario va a desarrollar software en una computadora personal IBM o un clon. Se recomienda utilizar el programa Microsoft MACRO Assembler (MASM) como herramienta de desarrollo, aunque el Intel Assembler (ASM), el Borland Turbo Assembler (TASM) o el software similar funcionan de manera similar. La versión más reciente de TASM emula completamente el programa MASM. Este libro presenta información que funciona con el ensamblador MASM de Microsoft, pero la mayoría de los programas se ensamblan con otros ensambladores sin necesidad de modificarse. En el apéndice A explicaremos el funcionamiento del ensamblador de Microsoft y proporcionaremos los detalles sobre el programa enlazador.] Como una alternativa más moderna también puede utilizarse el compilador Visual C++ y su programa ensamblador en línea como sistema de desarrollo. Este libro explicará el funcionamiento de ambos programas con más detalle.

OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

1. Explicar la operación de cada instrucción para mover datos con los modos de direccionamiento que se apliquen.
2. Explicar los propósitos de las seudooperaciones y las palabras clave del lenguaje ensamblador, tales como ALIGN, ASSUME, DB, DD, DW, END, ENDS, ENDP, EQU, .MODEL, OFFSET, ORG, PROC, PTR, SEGMENT, USE16, USE32 y USES.
3. Seleccionar la instrucción apropiada en lenguaje ensamblador para realizar una tarea de movimiento de datos específica.
4. Determinar el código de operación simbólico, el origen, el destino y el modo de direccionamiento para una instrucción en lenguaje máquina hexadecimal.
5. Utilizar el ensamblador para establecer un segmento de datos, un segmento de pila y un segmento de código.

6. Mostrar cómo establecer un procedimiento utilizando PROC y ENDP.
7. Explicar la diferencia entre los modelos de memoria y las definiciones de segmento completo para el ensamblador MASM.
8. Utilizar el ensamblador en línea de Visual C++ para realizar tareas de movimiento de datos.

4-1**RECAPITULACIÓN DE MOV**

La instrucción MOV que se presentó en el capítulo 3 explica la diversidad de los modos de direccionamiento de los microprocesadores del 8086 al Pentium 4. En este capítulo, la instrucción MOV presenta las instrucciones en lenguaje máquina disponibles con varios modos de direccionamiento e instrucciones. Se introducirá el código máquina porque en algunas ocasiones tal vez sea necesario interpretar los programas en lenguaje máquina generados por un ensamblador o por el ensamblador en línea de Visual C++. La interpretación del lenguaje nativo de la máquina (**lenguaje máquina**) permite la depuración o la modificación a nivel del lenguaje máquina. Algunas veces se realizan parches en lenguaje máquina mediante el uso del programa DEBUG disponible en DOS y también en Visual C++ para Windows, para lo cual se requiere cierto conocimiento del lenguaje máquina. En el apéndice B se muestra la conversión entre las instrucciones de lenguaje máquina y las de lenguaje ensamblador.

Lenguaje máquina

Este lenguaje es el código binario nativo que el microprocesador entiende y utiliza como sus instrucciones para controlar su operación. Las instrucciones en lenguaje máquina para los microprocesadores del 8086 al Pentium 4 varían en longitud, desde uno hasta 13 bytes. Aunque el lenguaje máquina parece complejo, hay un orden para el lenguaje máquina de estos microprocesadores. Hay más de 100,000 variaciones de instrucciones en lenguaje máquina, lo que significa que no existe una lista completa de estas variaciones. Debido a esto se proporcionan algunos bits binarios en una instrucción en lenguaje máquina, y los bits restantes se determinan para cada variación de la instrucción.

Las instrucciones para los microprocesadores del 8086 al 80286 son instrucciones en modo de 16 bits que toman la forma que se muestra en la figura 4-1(a). Las instrucciones en modo de 16 bits son compatibles con los microprocesadores 80386 y superiores si se programan para operar en el modo de instrucciones de 16 bits, aunque pueden ir con prefijos como se muestra en la figura 4-1(b). Los microprocesadores 80386 y superiores suponen que todas las instrucciones están en modo de 16 bits cuando el equipo se opera en el *modo real (DOS)*. En el *modo protegido (Windows)*, el byte superior del descriptor contiene el bit D que selecciona el modo de instrucciones de 16 o de 32 bits. Hasta ahora, sólo los sistemas operativos desde Windows 95 hasta Windows XP y Linux operan en el modo de instrucciones de 32 bits. Estas instrucciones se encuentran en la forma que se muestra en la figura 4-1(b).

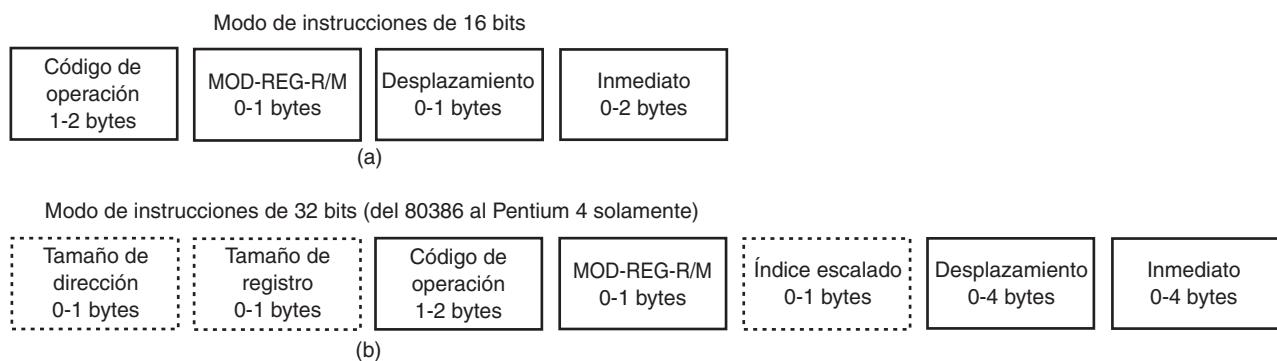


FIGURA 4-1 Los formatos de las instrucciones del 8086 al Pentium 4. (a) La forma de 16 bits y (b) la forma de 32 bits.

FIGURA 4-2 El byte 1 de muchas instrucciones en lenguaje máquina, en el que se muestra la posición de los bits D y W.



Estas instrucciones ocurren en el modo de instrucciones de 16 bits mediante el uso de prefijos, los cuales se explicarán más adelante.

Los primeros dos bytes del formato de modo de instrucciones de 32 bits se llaman **prefijos de sustitución**, ya que no siempre están presentes. El primer byte modifica el tamaño de la dirección del operando utilizado por la instrucción y el segundo modifica el tamaño del registro. Si los microprocesadores de 80386 al Pentium 4 operan como máquinas en modo de instrucciones de 16 bits (en modo real o protegido) y se utiliza un registro de 32 bits, el **prefijo de tamaño de registro** (66H) se anexa al frente de la instrucción. Si se opera en el modo de instrucciones de 32 bits (modo protegido solamente) y se utiliza un registro de 32 bits, no se utiliza el prefijo de tamaño de registro. Si aparece un registro de 16 bits en una instrucción en el modo de instrucciones de 32 bits, se utiliza el prefijo de tamaño de registro para seleccionar un registro de 16 bits. El **prefijo de tamaño de dirección** (67H) se utiliza en forma similar, como se explicará más adelante. Los prefijos cambian el tamaño del registro y la dirección del operando de 16 bits a 32 bits, o de 32 bits a 16 bits para la instrucción con el prefijo. El modo de instrucciones de 16 bits utiliza registros y modos de direccionamiento de 8 y de 16 bits, mientras que el modo de instrucciones de 32 bits utiliza registros y modos de direccionamiento de 8 y de 32 bits, de manera predeterminada. Los prefijos sustituyen estos valores predeterminados, de manera que pueda utilizarse un registro de 32 bits en el modo de 16 bits, o un registro de 16 bits en el modo de 32 bits. El **modo de operación** (16 o 32 bits) debe seleccionarse de manera que funcione con la aplicación actual. Si prevalecen los datos de 8 y de 32 bits en la aplicación se debe seleccionar el modo de 32 bits; de igual forma, si prevalecen los datos de 8 y de 16 bits debe seleccionarse el modo de 16 bits. Por lo general, la selección del modo es una función del sistema operativo. (Recuerde que el DOS puede operar solamente en el modo de 16 bits, pero Windows puede operar en ambos modos.)

El código de operación. El **código de operación** (opcode) selecciona la operación (suma, resta, movimiento, etc.) que va a realizar el microprocesador. El código de operación puede ser de uno o de dos bytes para la mayoría de las instrucciones en lenguaje máquina. La figura 4-2 ilustra la forma general del primer byte del código de operación de muchas instrucciones en lenguaje máquina (pero *no* de todas). Aquí, los primeros seis bits del primer byte son el código de operación binario. Los dos bits restantes indican la **dirección** (D) (que no debe confundirse con el bit de modo de instrucciones [16/32] o con el bit de bandera de dirección [utilizado con las instrucciones de cadenas]) del flujo de los datos, y que indica si los datos son un byte o una palabra (W). En los microprocesadores 80386 y superiores se especifican tanto palabras como dobles palabras cuando W = 1. Los prefijos de modo de instrucciones y de tamaño de registro (66H) determinan si W representa una palabra o una doble palabra.

Si el bit de dirección (D) = 1, los datos fluyen *hacia* el campo REG del registro, desde el campo R/M ubicado en el segundo byte de una instrucción. Si el bit D = 0 en el código de operación, los datos fluyen hacia el campo R/M *desde* el campo REG. Si el bit W = 1, el tamaño de los datos es de una *palabra* o *doble palabra*; si el bit W = 0, el tamaño de los datos es siempre un *byte*. El bit W aparece en la mayoría de las instrucciones, mientras que el bit D aparece por lo general con MOV y algunas otras instrucciones. En la figura 4-3 podrá consultar el patrón de bits del segundo byte del código de operación (reg-mod-r/m) de muchas instrucciones. La figura 4-3 muestra la posición de los campos MOD (modo), REG (registro) y R/M (registro/memoria).

FIGURA 4-3 El byte 2 de muchas instrucciones en lenguaje máquina. Aquí se muestra la posición de los campos MOD, REG y R/M.

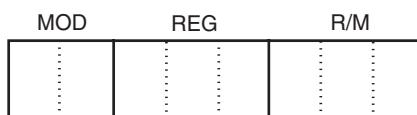


TABLA 4-1 El campo MOD para el modo de instrucciones de 16 bits.

MOD	Función
00	Sin desplazamiento.
01	Desplazamiento extendido de 8 bits.
10	Desplazamiento con signo de 16 bits.
11	R/M es un registro.

Campo MOD. El campo MOD especifica el modo de direccionamiento (MOD) para la instrucción seleccionada. El campo MOD selecciona el tipo de direccionamiento y si hay un desplazamiento presente con el tipo seleccionado. La tabla 4-1 lista las formas de operandos disponibles para el campo MOD en el modo de instrucciones de 16 bits, a menos que aparezca el prefijo de sustitución de tamaño de dirección del operando (67H). Si el campo MOD contiene 11, selecciona el modo de direccionamiento de registros. Este modo utiliza el campo R/M para especificar un registro en vez de una posición de memoria. Si el campo MOD contiene 00, 01 o 10, el campo R/M selecciona uno de los modos de direccionamiento de memoria de datos. Cuando MOD selecciona uno de estos modos, indica que el modo de direccionamiento no contiene desplazamiento (00), que contiene un desplazamiento extendido con signo de 8 bits (01) o que contiene un desplazamiento de 16 bits (10). La instrucción MOV AL,[DI] es un ejemplo que no contiene desplazamiento; la instrucción MOV AL,[DI+2] utiliza un desplazamiento de 8 bits (+2) y la instrucción MOV AL,[DI+1000H] utiliza un desplazamiento de 16 bits (+1000H).

Todos los desplazamientos de 8 bits se extienden con signo en desplazamientos de 16 bits cuando el microprocesador ejecuta la instrucción. Si el desplazamiento de 8 bits es 00H-7FH (positivo), se extiende con signo a 0000H-007FH antes de sumarse a la dirección de desplazamiento. Si el desplazamiento de 8 bits es 80H-FFH (negativo), se extiende con signo a FF80H-FFFFH. Para extender con signo un número, se copia su bit de signo al siguiente byte de mayor orden, el cual genera ya sea un 00H o un FFH en el siguiente byte de mayor orden. Algunos programas ensambladores no utilizan los desplazamientos de 8 bits, sino que se convierten todos en desplazamientos de 16 bits de manera pre-determinada.

En los microprocesadores del 80386 al Pentium 4 el campo MOD puede ser igual que el que se muestra en la tabla 4-1 para el modo de instrucciones de 16 bits; si el modo de instrucciones es de 32 bits, el campo MOD es como se muestra en la tabla 4-2. El campo MOD se interpreta según la selección del prefijo de sustitución de tamaño de dirección o del modo de operación del microprocesador. Este cambio en la interpretación del campo MOD y de la instrucción soporta muchos de los diversos modos de direccionamiento adicionales que se permiten en los microprocesadores del 80386 al Pentium 4. La principal diferencia es que cuando el campo MOD es 10, el desplazamiento de 16 bits se convierte en un desplazamiento de 32 bits para permitir el acceso a cualquier posición de memoria (4 Gbytes) en modo protegido. Los microprocesadores 80386 y superiores sólo permiten un desplazamiento de 8 o de 32 bits cuando se operan en el modo de instrucciones de 32 bits, a menos que aparezca el prefijo de sustitución de tamaño de dirección. Si se selecciona un desplazamiento de 8 bits, el microprocesador lo extiende con signo en un desplazamiento de 32 bits.

Asignaciones de registros. La tabla 4-3 lista las asignaciones de registros para el campo REG y el campo R/M (MOD = 11). Esta tabla contiene tres listas de asignaciones de registros. Uno se utiliza cuando el bit W = 0 (bytes) y los otros dos se utilizan cuando el bit W = 1 (palabras o dobles palabras). Tenga en cuenta que los registros tipo doble palabra sólo están disponibles en los microprocesadores del 80386 al Pentium 4.

TABLA 4-2 El campo MOD para el modo de instrucciones de 32 bits (sólo para los microprocesadores del 80386 al Pentium 4).

MOD	Función
00	Sin desplazamiento.
01	Desplazamiento extendido con signo de 8 bits.
10	Desplazamiento con signo de 32 bits.
11	R/M es un registro.

TABLA 4-3

Asignaciones de REG y R/M (cuando MOD = 11).

Código	<i>W = 0 (Byte)</i>	<i>W = 1 (Palabra)</i>	<i>W = 1 (Doble palabra)</i>
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

Suponga que una instrucción 8BECH de dos bytes aparece en un programa en lenguaje máquina. Como no aparece el 67H (prefijo de sustitución de tamaño de dirección del operando) o el 66H (prefijo de sustitución de tamaño de registro) como el primer byte, entonces se considera que el código de operación es el primer byte. Si el microprocesador se opera en el modo de instrucciones de 16 bits, esta instrucción se convierte en código binario y se coloca en el formato de instrucción de los bytes 1 y 2, como se muestra en la figura 4-4. El código de operación es 100010. Si hace referencia al apéndice B, que lista las instrucciones en lenguaje máquina, encontrará que éste es el código de operación para una instrucción MOV. Observe que los bits D y W son ambos un 1 lógico, lo que significa que se mueve una palabra hacia el registro de destino especificado en el campo REG. Este campo contiene 101, lo cual indica al registro BP, por lo que la instrucción MOV mueve los datos hacia el registro BP. Como el campo MOD contiene 11, el campo R/M también indica un registro. Aquí, R/M = 100 (SP); por lo tanto, esta instrucción mueve los datos de SP hacia BP y se escribe en forma simbólica como una instrucción MOV BP,SP.

Suponga que aparece una instrucción 668BE8H en un microprocesador 80386 o superior, que se opere en el modo de instrucciones de 16 bits. El primer byte (66H) es el prefijo de sustitución de tamaño de registro que selecciona como operandos registros de 32 bits para el modo de instrucción de 16 bits. El resto de la instrucción indica que el código de operación es MOV con un operando de origen EAX y un operando de destino EBP. Esta instrucción es MOV EBP,EAX. La misma instrucción se convierte en MOV BP,AX en los microprocesadores 80386 y superiores si se opera en el modo de instrucciones de 32 bits, ya que el prefijo de sustitución de tamaño de registro selecciona un registro de 16 bits. Por fortuna el programa ensamblador lleva la cuenta de los prefijos de tamaño de registro y de dirección, y del modo de operación. Recuerde que si el modificador .386 se coloca antes de la instrucción .MODEL, se selecciona el modo de 32 bits; si se coloca después de la instrucción .MODEL, se selecciona el modo de 16 bits. Todos los programas escritos mediante el ensamblador en línea de Visual C++ están siempre en el modo de 32 bits.

Direccionamiento de memoria R/M. Si el campo MOD contiene 00, 01 o 10, el campo R/M toma un nuevo significado. La tabla 4-4 muestra los modos de direccionamiento de memoria para el campo R/M, cuando MOD es 00, 01 o 10 para el modo de instrucciones de 16 bits.

Código de operación				D	W	MOD				REG	R/M	
1	0	0	0	1	0	1	1	1	0	1	1	
Código de operación = MOV												
D = Transferencia al registro (REG)												
W = Palabra												
MOD = R/M es un registro												
REG = BP												
R/M = SP												

FIGURA 4-4 La instrucción 8BEC se coloca en los formatos de los bytes 1 y 2 de las figuras 4-2 y 4-3. Esta instrucción es MOV BP,SP.

TABLA 4-4 Los modos de direccionamiento de memoria R/M de 16 bits.

Código R/M	Modo de direccionamiento
000	DS:[BX+SI]
001	DS:[BX+DI]
010	SS:[BP+SI]
011	SS:[BP+DI]
100	DS:[SI]
101	DS:[DI]
110	SS:[BP]*
111	DS:[BX]

*Vea la sección “Modo de direccionamiento especial”.

Todos los modos de direccionamiento de 16 bits que se presentaron en el capítulo 3 y aparecen en la tabla 4-4. El desplazamiento que vimos en el capítulo 3 se define mediante el campo MOD. Si MOD = 00 y R/M = 101, el modo de direccionamiento es [DI]. Si MOD = 01 o 10, el modo de direccionamiento es [DI+33H], o LISTA [DI+22H] para el modo de instrucciones de 16 bits. En este ejemplo utilizamos LISTA, 33H y 22H como valores arbitrarios para el desplazamiento.

La figura 4-5 ilustra la versión en lenguaje máquina de la instrucción de 16 bits MOV DL,[DI], o instrucción (8A15H). Esta instrucción es de dos bytes y tiene el código de operación 100010, D = 1 (hacia REG desde R/M), W = 0 (byte), MOD = 00 (sin desplazamiento), REG = 010 (DL) y R/M = 101 ([DI]). Si la instrucción cambia a MOV DL,[DI+1], el campo MOD cambia a 01 para un desplazamiento de 8 bits, pero los primeros dos bytes de la instrucción quedan sin cambio. La instrucción ahora se convierte en 8A5501H en vez de 8A15H. Observe que el desplazamiento de 8 bits se adjunta a los primeros dos bytes de la instrucción para formar una instrucción de tres bytes en vez de dos. Si la instrucción se cambia de nuevo a MOV DL,[DI+1000H], el formato en lenguaje máquina se convierte en 8A750010H. Aquí, el desplazamiento de 16 bits de 1000H (codificado como 0010H) adjunta el código de operación.

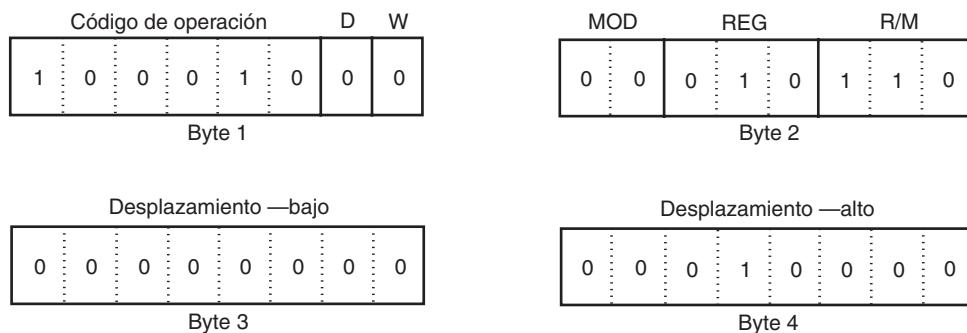
Modo de direccionamiento especial. Existe un modo de direccionamiento especial que no aparece en las tablas 4-2, 4-3 o 4-4. Ocurre siempre que se hace referencia a los datos de la memoria sólo mediante el modo de direccionamiento con desplazamiento para las instrucciones de 16 bits. Las instrucciones MOV [1000H],DL y MOV NUMERO,DL son dos ejemplos. La primera instrucción mueve el contenido del registro DL hacia la posición de memoria 1000H del segmento de datos. La segunda instrucción mueve el registro DL hacia la posición de memoria NUMERO del segmento de datos.

Cuando una instrucción tiene sólo un desplazamiento, el campo MOD es siempre 00 y el campo R/M es siempre 110. Como se muestra en las tablas, la instrucción no contiene desplazamiento y utiliza el modo de direccionamiento [BP]. No se puede utilizar el modo de direccionamiento [BP] sin un desplazamiento en lenguaje máquina. El ensamblador se encarga de esto mediante el uso de un desplazamiento de 8 bits (MOD = 01) de 00H siempre que aparece el modo de direccionamiento [BP] en

Código de operación	D	W	MOD	REG	R/M
1 0 0 0 1 0	1	0	0 0	1 0	1 0 1

Código de operación = MOV
D = Transferencia hacia el registro (REG)
W = Byte
MOD = Sin desplazamiento
REG = DL
R/M = DS:[DI]

FIGURA 4-5 Una instrucción MOV DL,[DI] convertida a su formato en lenguaje máquina.



Código de operación = MOV

D = Transferencia desde el registro (REG)

W = Byte

MOD = debido a R/M es [BP] (direcciónamiento especial)

REG = DL

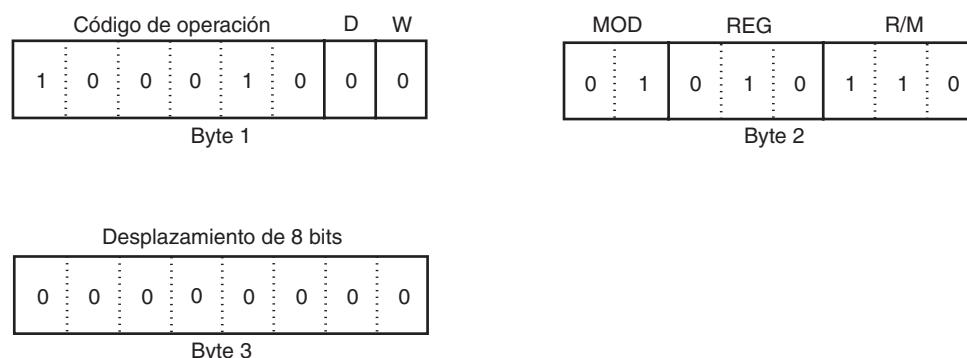
R/M = DS:[BP]

Desplazamiento = 1000H

FIGURA 4-6 La instrucción MOV [1000H],DL utiliza el modo de direcciónamiento especial.

una instrucción. Esto significa que el modo de direcciónamiento [BP] se ensambla como [BP+0], aún y cuando se utiliza [BP] en la instrucción. También está disponible el mismo modo de direcciónamiento especial para el modo de 32 bits.

La figura 4-6 muestra el patrón de bits binario requerido para codificar la instrucción MOV [1000H],DL en lenguaje máquina. Si quien traduce esta instrucción simbólica en lenguaje máquina no sabe acerca del modo de direcciónamiento especial, la instrucción se traduciría incorrectamente en una instrucción MOV [BP],DL. La figura 4-7 muestra el formato actual de la instrucción MOV [BP],DL. Observe que ésta es una instrucción de tres bytes con un desplazamiento de 00H.



Código de operación = MOV

D = Transferencia desde el registro (REG)

W = Byte

MOD = debido a R/M es [BP] (direcciónamiento especial)

REG = DL

R/M = DS:[BP]

Desplazamiento = 00H

FIGURA 4-7 La instrucción MOV [BP],DL convertida en lenguaje máquina binario.

TABLA 4-5 Los modos de direccionamiento de 32 bits seleccionados por R/M.

Código R/M	Función
000	DS:[EAX]
001	DS:[ECX]
010	DS:[EDX]
011	DS:[EBX]
100	Utiliza el byte de índice escalado
101	SS:[EBP]*
110	DS:[ESI]
111	DS:[EDI]

*Vea la sección “Modo de direccionamiento especial”.

Modos de direccionamiento de 32 bits. Los modos de direccionamiento de 32 bits incluidos en los microprocesadores 80386 y superiores se obtienen ya sea mediante la ejecución de estos equipos en el modo de instrucciones de 32 bits, o en el modo de instrucciones de 16 bits mediante el uso del prefijo 67H de tamaño de dirección. La tabla 4-5 muestra la codificación para R/M que se utiliza para especificar los modos de direccionamiento de 32 bits. Observe que cuando R/M = 100 aparece un byte adicional llamado **byte de índice escalado** en la instrucción. Este byte de índice escalado indica las formas adicionales de direccionamiento de índice escalado que no aparecen en la tabla 4-5. El byte de índice escalado se utiliza principalmente cuando se suman dos registros para especificar la dirección de memoria en una instrucción. Como el byte de índice escalado se suma a la instrucción, hay siete bits en el código de operación y ocho bits en el byte de índice escalado que deben definirse. Esto significa que una instrucción de índice escalado tiene 2^{15} (32K) posibles combinaciones. Existen tan sólo más de 32,000 variaciones distintas de la instrucción MOV en los microprocesadores del 80386 al Pentium 4.

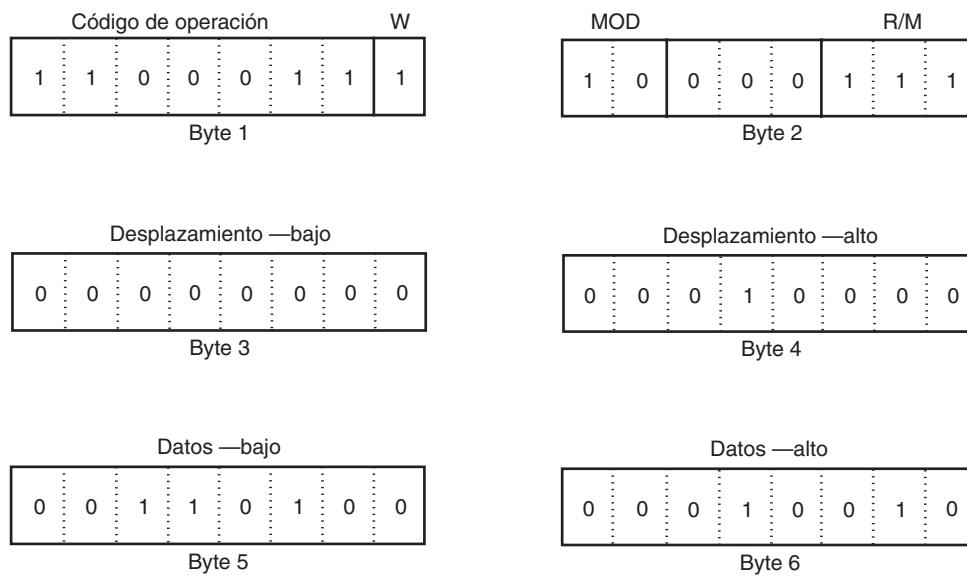
La figura 4-8 muestra el formato del byte de índice escalado según su selección mediante un valor de 100 en el campo R/M de una instrucción, cuando los microprocesadores 80386 y superiores utilizan una dirección de 32 bits. Los dos bits de más a la izquierda seleccionan un factor de escala (multiplicador) de $1\times$, $2\times$, $4\times$ u $8\times$. El factor de escala de $1\times$ está implícito si no se utiliza ningún factor en una instrucción que contenga dos registros de dirección indirectos de 32 bits. Los campos de índice y base contienen números de registros, como lo indica la tabla 4-3 para registros de 32 bits.

La instrucción MOV EAX,[EBX+4*ECX] se codifica como 67668B048BH. En esta instrucción aparecen los prefijos de *tamaño de dirección* (67H) y de *tamaño de registro* (66H). Esta codificación (67668B048BH) se utiliza cuando los microprocesadores 80386 y superiores se operan en el modo de instrucciones de 16 bits para esta instrucción. Si el microprocesador opera en el modo de instrucciones de 32 bits, ambos prefijos desaparecen y la instrucción se convierte en 8B048BH. El uso de los prefijos depende del modo de operación del microprocesador. En el direccionamiento de índice escalado también se puede utilizar un solo registro multiplicado por un factor de escala. La instrucción MOV AL,[2*ECX] es un ejemplo. El contenido de la posición del segmento de datos direccionada por dos multiplicado por ECX se copia en AL.

Una instrucción inmediata. Suponga que se elige la instrucción MOV WORD PTR [BX+1000H],1234H como ejemplo de una instrucción de 16 bits que utiliza direccionamiento inmediato. Esta instrucción

FIGURA 4-8 El byte de índice escalado.

s	s	Índice	Base
...
...
...
ss			
00	= $\times 1$		
01	= $\times 2$		
10	= $\times 4$		
11	= $\times 8$		



Código de operación = MOV (inmediato)

W = Palabra

MOD = Desplazamiento de 16 bits

REG = 000 (no se utiliza en el direccionamiento inmediato)

R/M = DS:[BX]

Desplazamiento = 1000H

Datos = 1234H

FIGURA 4-9 Una instrucción MOV WORD PTR [BX+1000H],1234H convertida en lenguaje máquina binario.

mueve el número 1234H hacia la posición de memoria tipo palabra direccionada por la suma de 1000H, BX y DS × 10H. Esta instrucción de seis bytes utiliza dos bytes para los campos código de operación, W, MOD y R/M. Dos de los seis bytes son los datos representados por 1234H; dos de los seis bytes son el desplazamiento de 1000H. La figura 4-9 muestra el patrón de bits binario para cada byte de esta instrucción.

Esta instrucción incluye WORD PTR en su forma simbólica. La directiva WORD PTR indica al ensamblador que la instrucción utiliza un apuntador a memoria tipo palabra. Si la instrucción mueve un byte de datos inmediatos, BYTE PTR sustituye a WORD PTR en la instrucción. De igual forma, si la instrucción utiliza una doble palabra de datos inmediatos, la directiva DWORD PTR sustituye a BYTE PTR. La mayoría de las instrucciones que hacen referencia a la memoria a través de un apuntador no necesitan las directivas BYTE PTR, WORD PTR o DWORD PTR. Estas directivas son necesarias solamente cuando no está claro si la operación es un byte, una palabra o una doble palabra. La instrucción MOV [BX],AL es evidentemente el movimiento de un byte; la instrucción MOV [BX],9 no es exacta, ya que podría ser el movimiento de un byte, una palabra o una doble palabra. Aquí la instrucción debe codificarse como MOV BYTE PTR [BX],9, MOV WORD PTR [BX],9 o MOV DWORD PTR [BX],9. Si no es así, el ensamblador lo marca como un error ya que no puede determinar la intención de esta instrucción.

Instrucciones MOV de segmentos. Si se mueve el contenido de un registro de segmento mediante las instrucciones MOV, PUSH o POP, un conjunto especial de bits de registro (campo REG) selecciona el registro de segmento (vea la tabla 4-6).

La figura 4-10 muestra una instrucción MOV BX,CS convertida en binario. El código de operación para este tipo de instrucción MOV es distinto de las instrucciones MOV anteriores. Los registros de segmento pueden moverse entre cualquier registro de 16 bits o cualquier posición de memoria de 16

TABLA 4-6 Selección del registro de segmento.

Código	Registro de segmento
000	ES
001	CS*
010	SS
011	DS
100	FS
101	GS

*No se permite el uso de MOV CS,R/M y POP CS.

Código de operación								MOD	REG	R/M	
1	0	0	0	1	1	0	0	1	0	1	1

Código de operación = MOV

MOD = R/M es un registro

REG = CS

R/M = BX

FIGURA 4-10 Una instrucción MOV BX,CS convertida en lenguaje máquina binario.

bits. Por ejemplo, la instrucción MOV [DI],DS almacena el contenido de DS en la posición de memoria direccionada por DI en el segmento de datos. En el conjunto de instrucciones no hay una instrucción MOV de registro de segmento inmediata. Para cargar un registro de segmento con datos inmediatos, primero se debe cargar otro registro con los datos y después se deben mover al registro de segmento.

Aunque esta discusión no ha sido una cobertura completa de la codificación en lenguaje máquina, proporciona suficiente información para la programación en lenguaje máquina. Recuerde que un programa escrito en lenguaje ensamblador simbólico (*lenguaje ensamblador*) raras veces se ensambla a mano para convertirlo en lenguaje máquina binario. Un programa ensamblador convierte el lenguaje ensamblador simbólico en lenguaje máquina. Como el microprocesador tiene más de 100,000 variaciones de instrucciones, esperemos que haya un ensamblador disponible para la conversión, ya que este proceso toma mucho tiempo, por no decir que es imposible.

4-2

PUSH/POP

PUSH y POP son instrucciones importantes que *almacenan y recuperan* datos de la memoria de pila LIFO (último en entrar, primero en salir). El microprocesador tiene seis formas de instrucciones PUSH y POP: de registro, de memoria, inmediata, de registro de segmento, de banderas y de todos los registros. Las formas PUSH y POP inmediatas y PUSHA y POPA (todos los registros) no están disponibles en los microprocesadores 8086/8088, pero sí pueden utilizarse en los microprocesadores del 80286 al Pentium 4.

El direccionamiento de registros permite transferir el contenido de cualquier registro de 16 bits hacia o desde la pila. En los microprocesadores 80386 y superiores también pueden meterse o sacarse de la pila los registros extendidos de 32 bits y las banderas (EFLAGS). Las instrucciones PUSH y POP de direccionamiento de memoria almacenan en una posición de memoria el contenido de una posición de memoria de 16 bits (o de 32 bits en los microprocesadores 80386 y superiores) en la pila o los datos de la pila. El direccionamiento inmediato permite meter datos en la pila, pero no sacarlos. El direccionamiento de registros de segmento permite meter en la pila o sacar de la pila el contenido de

cualquier registro de segmento (ES puede meterse, pero no pueden sacarse datos de la pila y colocarse en ES). Las banderas pueden meterse o sacarse de esa pila, y el contenido de todos los registros puede meterse o sacarse.

PUSH

La instrucción PUSH de los microprocesadores 8086-80286 siempre transfiere dos bytes de datos hacia la pila; los microprocesadores 80386 y superiores transfieren dos o cuatro bytes, dependiendo del registro o del tamaño de la posición de memoria. El origen de los datos puede ser cualquier registro interno de 16 o 32 bits, datos inmediatos, cualquier registro de segmento o cualesquiera dos bytes de datos de la memoria. También existe una instrucción PUSHA que copia el contenido del conjunto de registros interno (excepto los registros de segmento) hacia la pila. La instrucción PUSHA (**meter todos**) copia los registros en la pila, en el siguiente orden: AX, CX, DX, BX, SP, BP, SI y DI. El valor de SP que se mete en la pila es el que tenga antes de ejecutarse la instrucción PUSHA. La instrucción PUSHF (**meter banderas**) copia el contenido del registro de banderas en la pila. Las instrucciones PUSHAD y POPAD meten y sacan el contenido del conjunto de registros de 32 bits incluido en los microprocesadores del 80386 al Pentium 4.

Siempre que se meten datos en la pila, el primer byte de datos (el más significativo) se mueve a la posición de memoria del segmento de la pila direccionada por $SP - 1$. El segundo byte de datos (menos significativo) se mueve a la posición de memoria del segmento de la pila direccionada por $SP - 2$. Una vez que se almacenan los datos mediante una instrucción PUSH, el contenido del registro SP se decremente en 2. Lo mismo se aplica para una instrucción PUSH con datos tipo doble palabra, sólo que se mueven cuatro bytes a la memoria de la pila (el byte más significativo primero), después de lo cual el apuntador de la pila se decrementa en 4. La figura 4-11 muestra la operación de la instrucción PUSH AX. Esta instrucción copia el contenido de AX en la pila, en donde la dirección SS:[$SP - 1$] = AH, SS:[$SP - 2$] = AL, y después de eso $SP = SP - 2$.

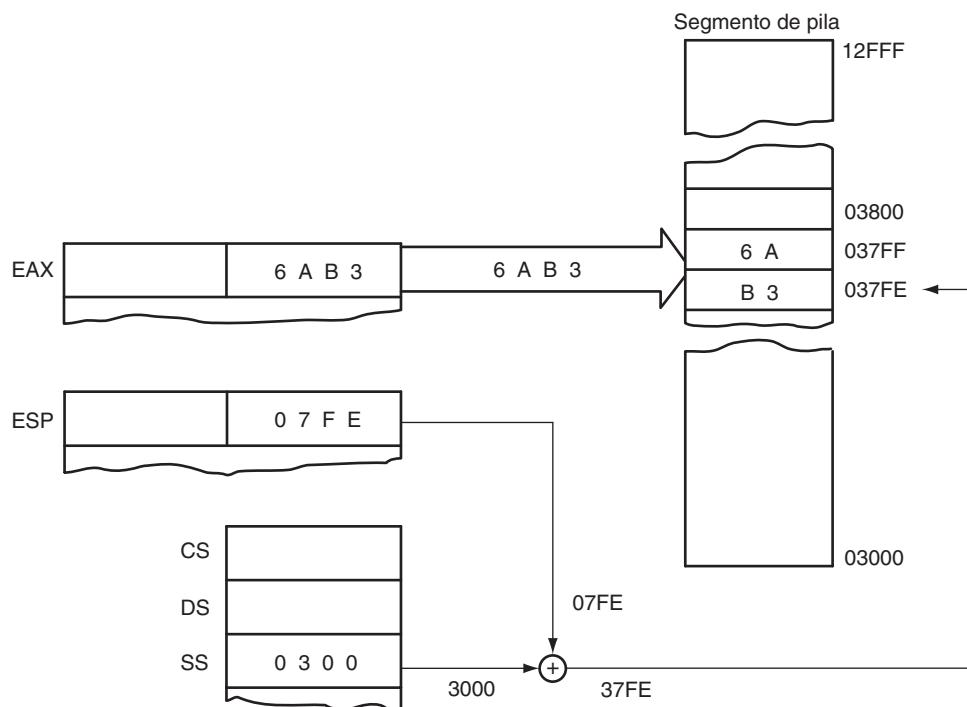
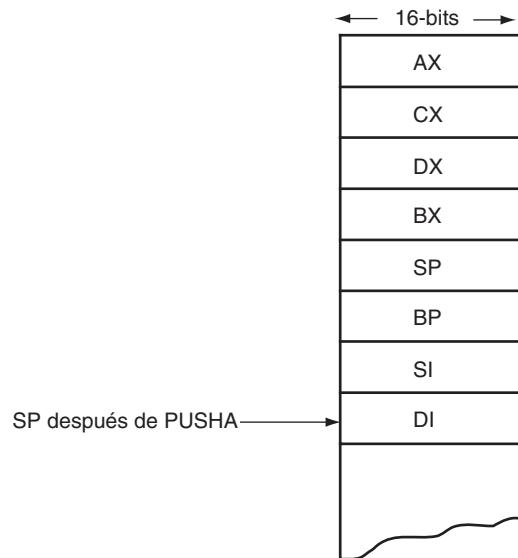


FIGURA 4-11 El efecto de la instrucción PUSH AX en ESP y las posiciones 37FFH y 37FEH de la memoria de la pila. Esta instrucción se muestra en el punto después de su ejecución.

FIGURA 4-12 La operación de la instrucción **PUSHA**, en donde se muestra la posición y el orden de los datos de la pila.



La instrucción **PUSHA** mete todos los registros internos de 16 bits en la pila, como se muestra en la figura 4-12. Esta instrucción requiere 16 bytes de espacio de memoria de la pila para almacenar los ocho registros de 16 bits. Una vez que se meten los registros, el contenido del registro **SP** se decrementa en 16. La instrucción **PUSHA** es muy útil cuando todo el conjunto de registros (entorno del microprocesador) de los microprocesadores 80286 y superiores debe guardarse durante una tarea. La instrucción **PUSHAD** coloca el conjunto de registros de 32 bits en la pila, en los microprocesadores del 80386 al Pentium 4. **PUSHAD** requiere 32 bytes de espacio de almacenamiento en la pila.

La instrucción **PUSH** de datos inmediatos tiene dos códigos de operación distintos, pero en ambos casos se mueve un número inmediato de 16 bits a la pila; si se utiliza **PUSHD**, se mete un dato inmediato de 32 bits. Si el valor de los datos inmediatos es 00H-FFH, el código de operación es 6AH; si los datos son 0100H-FFFFH, el código de operación es 68H. La instrucción **PUSH 8**, que mete el número 0008H en la pila, se ensambla como 6A08H. La instrucción **PUSH 1000H** se ensambla como 680010H. Otro ejemplo de una instrucción **PUSH** inmediata es la instrucción **PUSH 'A'**, que mete el número 0041H en la pila. Aquí, 41H es el código ASCII para la letra A.

La tabla 4-7 lista las formas de la instrucción **PUSH** que incluyen a **PUSHA** y **PUSHF**. Observe cómo se utiliza el conjunto de instrucciones para especificar distintos tamaños de datos con el ensamblador.

TABLA 4-7 La instrucción **PUSH**.

Simbólica	Ejemplo	Observación
PUSH reg16	PUSH BX	Registro de 16 bits.
PUSH reg32	PUSH EDX	Registro de 32 bits.
PUSH mem16	PUSH WORD PTR[BX]	Apuntador de 16 bits.
PUSH mem32	PUSH DWORD PTR[EBX]	Apuntador de 32 bits.
PUSH seg	PUSH DS	Registro de segmento.
PUSH inm8	PUSH 'R'	8 bits, inmediato.
PUSH inm16	PUSH 1000H	16 bits, inmediato.
PUSH inm32	PUSHD 20	32 bits, inmediato.
PUSHA	PUSHA	Guarda todos los registros de 16 bits.
PUSHAD	PUSHAD	Guarda todos los registros de 32 bits.
PUSHF	PUSHF	Guarda las banderas.
PUSHFD	PUSHFD	Guarda EFLAGS.

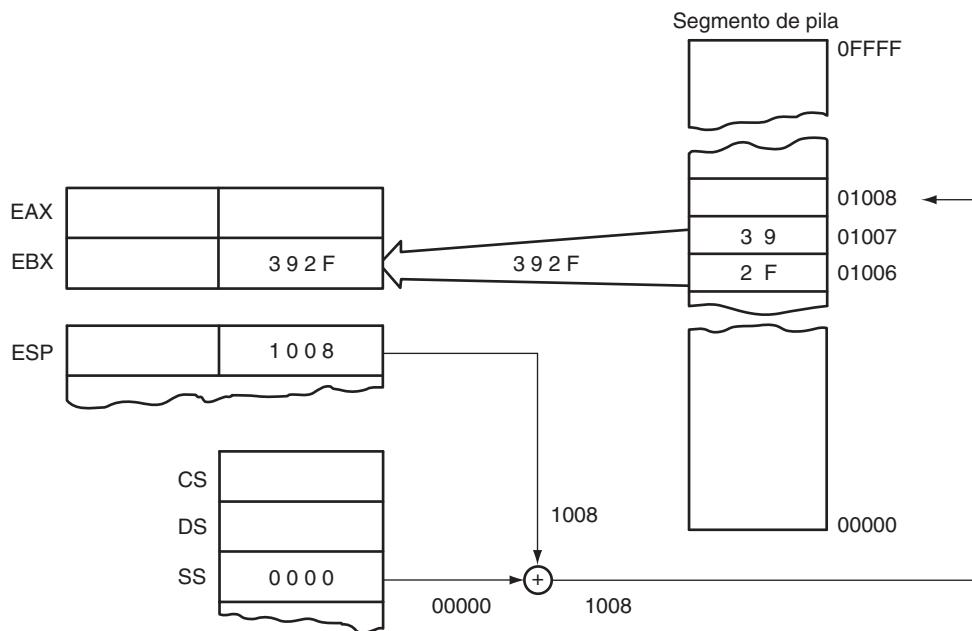


FIGURA 4-13 La instrucción POP BX, en donde se muestra cómo se sacan los datos de la pila. La instrucción se muestra después de su ejecución.

POP

Esta instrucción realiza la operación inversa de una instrucción PUSH. La instrucción POP saca datos de la pila y los coloca en el destino que puede ser un registro de 16 bits, un registro de segmento o una posición de memoria de 16 bits. En los microprocesadores 80386 y superiores, una instrucción POP también puede sacar datos de 32 bits de la pila y puede utilizar una dirección de 32 bits. La instrucción POP no está disponible como POP inmediata. La instrucción POPF (**sacar banderas**) saca un número de 16 bits de la pila y lo coloca en el registro de banderas; la instrucción POPFD saca un número de 32 bits de la pila y lo coloca en el registro de banderas extendido. La instrucción POPA (**sacar todo**) saca 16 bytes de datos de la pila y los coloca en los siguientes registros, en el orden mostrado: DI, SI, BP, SP, BX, DX, CX y AX. Éste es el orden inverso de la forma en que se colocaron en la pila mediante la instrucción PUSHA, lo que hace que regresen los mismos datos a los mismos registros. En los microprocesadores 80386 y superiores, una instrucción POPAD saca los registros de 32 bits de la pila.

Suponga que se ejecuta una instrucción POP BX. El primer byte de datos que se saca de la pila (la posición de memoria direccionada por SP en el segmento de pila) se mueve hacia el registro BL. El segundo byte se saca de la posición de memoria SP + 1 del segmento de pila y se coloca en el registro BH. Después de sacar ambos bytes de la pila, el registro SP se incrementa en 2. La figura 4-13 muestra cómo la instrucción POP BX saca datos de la pila y los coloca en el registro BX.

En la tabla 4-8 aparecen los códigos de operación utilizados para la instrucción POP y todas sus variaciones. Tenga en cuenta que la instrucción POP CS no es válida en el conjunto de instrucciones. Si se ejecuta una instrucción POP CS sólo cambia una porción de la dirección (CS) de la siguiente instrucción. Esto hace que la instrucción POP CS sea impredecible y, por lo tanto, no se permite.

Inicialización de la pila

Cuando se inicializa el área de la pila, se carga tanto el registro de segmento de pila (SS) y el registro de apuntador de pila (SP). Es normal diseñar un área de la memoria como el segmento de pila en donde se carga SS con la posición inferior del segmento de pila.

TABLA 4-8 Las instrucciones POP.

Simbólica	Ejemplo	Observación
POP reg16	POP CX	Registro de 16 bits.
POP reg32	POP EBP	Registro de 32 bits.
POP mem16	POP WORD PTR[BX+1]	Apuntador de 16 bits.
POP mem32	POP DATOS3	Dirección de memoria de 32 bits.
POP seg	POP FS	Registro de segmento.
POPA	POPA	Saca todos los registros de 16 bits.
POPAD	POPAD	Saca todos los registros de 32 bits.
POPF	POPF	Saca las banderas.
POFFD	POFFD	Mete EFLAGS.

Por ejemplo, si el segmento de pila reside en las posiciones de memoria 10000H-1FFFFH, se carga SS con 1000H. (Recuerde que al extremo más a la derecha del registro de segmento de pila se le adjunta un 00H para el direccionamiento en modo real.) Para comenzar la pila en la parte superior de este segmento de pila de 64K, El apuntador de la pila (SP) se carga con un 0000H. De igual forma, para direccionar la parte superior de la pila en la posición 10FFFH, se utiliza un valor de 1000H en SP. La figura 4-14 muestra cómo este valor hace que los datos se metan en la parte superior del segmento de pila, mediante una instrucción PUSH CX. Recuerde que todos los segmentos son cíclicos en naturaleza; esto es, la posición superior de un segmento es contigua a la posición inferior del segmento.

En lenguaje ensamblador, un segmento de pila se establece como se muestra en el ejemplo 4-1. La primera instrucción identifica el inicio del segmento de pila y la última instrucción identifica el final del segmento de la pila. Los programas ensamblador y enlazador colocan la dirección correcta del segmento de pila en SS y la longitud correcta del segmento (parte superior de la pila) en SP. No hay necesidad de cargar estos registros en su programa, a menos que desee cambiar los valores iniciales por alguna razón.

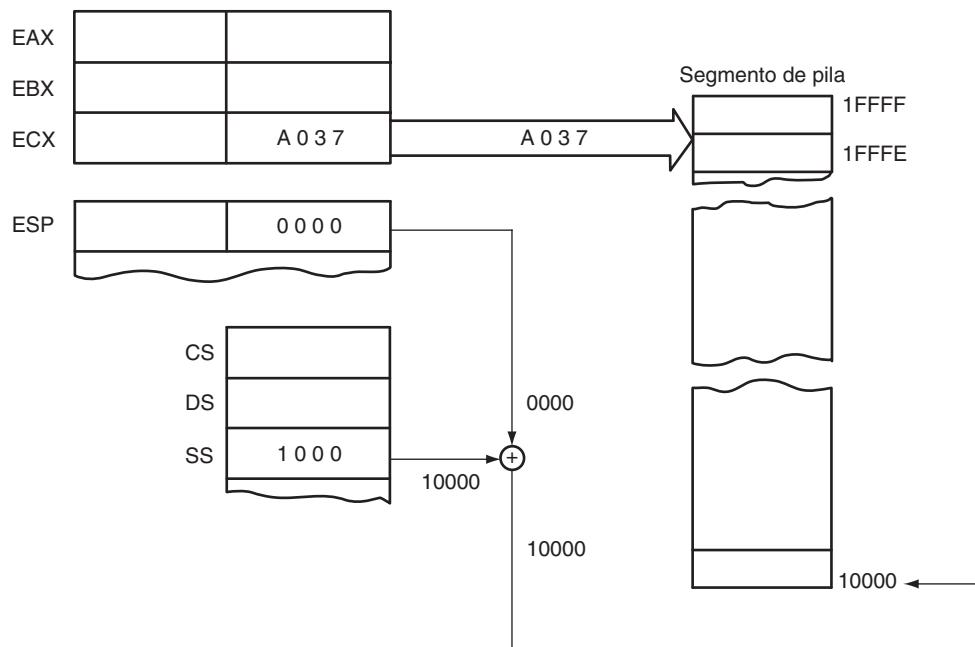


FIGURA 4-14 La instrucción PUSH CX, en donde se muestra la naturaleza cíclica del segmento de pila. Esta instrucción se muestra justo antes de su ejecución, para ilustrar que la parte inferior de la pila es contigua a la parte superior.

EJEMPLO 4-1

```

0000           STACK_SEG      SEGMENT  STACK
0000 0100[      DW      100H DUP (?)
    ???
]
0200           STACK_SEG      ENDS

```

Un método alternativo para definir el segmento de pila se usa con uno de los modelos de memoria para el ensamblador MASM solamente (consulte el apéndice A). Otros ensambladores no utilizan modelos; y si lo hacen, no son exactamente los mismos que con MASM. Aquí la instrucción .STACK, que va seguida del número de bytes asignados a la pila, define el área de la misma (vea el ejemplo 4-2). La función es idéntica al ejemplo 4-1. La instrucción .STACK también inicializa a SS y a SP. En este libro utilizamos modelos de memoria diseñados para el programa MASM (Microsoft Macro Assembler).

EJEMPLO 4-2

```

.MODEL  SMALL
.STACK 200H ;establece el tamaño de la pila

```

Si no se especifica la pila mediante el uso de cualquiera de los métodos antes mencionados, aparecerá una advertencia al enlazar el programa. Esta advertencia puede ignorarse si el tamaño de la pila es de 128 bytes o menos. El sistema signa automáticamente (mediante DOS) por lo menos 128 bytes de memoria a la pila. Esta sección de memoria está ubicada en el **prefijo de segmento del programa** (PSP), el cual se adjunta al principio de cada archivo de programa. Si utiliza más memoria para la pila, borrará información en el PSP que es imprescindible para la operación de su programa y de la computadora. Este error a menudo provoca que el programa de la computadora falle. Si se utiliza el modelo de memoria TINY, la pila se posiciona automáticamente al final del segmento, lo cual le permite usar un área más grande.

4-3**DIRECCIÓN EFECTIVA DE CARGA**

Existen varias instrucciones de dirección efectiva de carga (LEA) en el conjunto de instrucciones del microprocesador. La instrucción LEA carga cualquier registro de 16 bits con la dirección de desplazamiento, según lo que se determine en base al modo de direccionamiento seleccionado para la instrucción.

TABLA 4-9 Instrucciones de dirección efectiva de carga.

<i>Lenguaje ensamblador</i>	<i>Operación</i>
LEA AX,NUMERO	Carga AX con la dirección de desplazamiento de NUMERO.
LEA EAX,NUMERO	Carga EAX con la dirección de desplazamiento de NUMERO.
LDS DI,LISTA	Carga DS y DI con el contenido de 32 bits de la posición de memoria LISTA del segmento de datos.
LDS EDI,LISTA1	Carga DS y EDI con el contenido de 48 bits de la posición de memoria LISTA1 del segmento de datos.
LES BX,GATO	Carga ES y BX con el contenido de 32 bits de la posición de memoria GATO del segmento de datos.
LFS DI,DATOS1	Carga FS y DI con el contenido de 32 bits de la posición de memoria DATOS1 del segmento de datos.
LGS SI,DATOS5	Carga GS y SI con el contenido de 32 bits de la posición de memoria DATOS5 del segmento de datos.
LSS SP,MEM	Carga SS y SP con el contenido de 32 bits de la posición de memoria MEM del segmento de datos.

Las variaciones LDS y LES cargan cualquier registro de 16 bits con la dirección de desplazamiento que se recupera de una posición de memoria, y luego cargan DS o ES con una dirección de segmento que se recupera de la memoria. En los microprocesadores 80386 y superiores, LFS, LGS y LSS se agregan al conjunto de instrucciones, y puede seleccionarse un registro de 32 bits para recibir un desplazamiento de 32 bits de la memoria. La tabla 4-9 lista las instrucciones de dirección efectiva de carga.

LEA

Esta instrucción carga un registro de 16 o de 32 bits con la dirección de desplazamiento de los datos especificados por el operando. Como se muestra en el primer ejemplo de la tabla 4-9, la dirección NUMERO del operando se carga en el registro AX, no el contenido de la dirección NUMERO.

Si comparamos LEA con MOV podemos observar que LEA BX,[DI] carga la dirección de desplazamiento especificada por [DI] (el contenido de DI) en el registro BX; MOV BX,[DI] carga en el registro BX los datos almacenados en la posición de memoria direccionada por [DI].

Ya hemos presentado varios ejemplos en los que se utiliza la directiva OFFSET. Esta directiva realiza la misma función que una instrucción LEA si el operando es un desplazamiento. Por ejemplo, la instrucción MOV BX,OFFSET LISTA realiza la misma función que LEA BX,LISTA. Ambas instrucciones cargan en el registro BX la dirección de desplazamiento de la posición de memoria LISTA. En el ejemplo 4-3 podrá ver un programa corto que carga SI con la dirección de DATOS1 y DI con la dirección de DATOS2. Después intercambia el contenido de estas posiciones de memoria. Las instrucciones LEA y MOV con OFFSET tienen la misma longitud (tres bytes).

EJEMPLO 4-3

```

0000          .MODEL SMALL           ;selecciona el modelo pequeño
0000          .DATA                ;inicio del segmento de datos
0000 2000      DATOS1             ;define DATOS1
0002 3000      DATOS2             ;define DATOS2
0000          .CODE               ;inicio del segmento de código
                                .STARTUP            ;inicio del programa
0017 BE 0000 R    LES SI,DATA1   ;direcciona DATOS1 con SI
001A BF 0002 R    MOV DI,OFFSET DATA2 ;direcciona DATOS2 con DI
001D 8B 1C        MOV BX,[SI]       ;intercambia DATOS1 con DATOS2
001F 8B 0D        MOV CX,[DI]
0021 89 0C        MOV [SI],CX
0023 89 1D        MOV [DI],BX
                                .EXIT
                                END

```

Pero ¿por qué está disponible la instrucción LEA si la directiva OFFSET realiza la misma tarea? En primer lugar, OFFSET funciona solamente con operandos simples como LISTA. Tal vez no pueda usarse para un operando tal como [DI], LISTA [SI], y así sucesivamente. La directiva OFFSET es más eficiente que la instrucción LEA para operandos simples. El microprocesador se tarda más en ejecutar la instrucción LEA BX,LISTA que la instrucción MOV BX,OFFSET LISTA. Por ejemplo, el microprocesador 80486 requiere dos ciclos de reloj para ejecutar la instrucción LEA BX,LISTA y sólo un ciclo de reloj para ejecutar MOV BX,OFFSET LISTA. La razón por la que la instrucción MOV BX,OFFSET LISTA se ejecuta más rápido es porque el ensamblador calcula la dirección de desplazamiento de LISTA, mientras que el microprocesador calcula la dirección para la instrucción LEA. La instrucción MOV BX,OFFSET LISTA en realidad se ensambla como una instrucción de movimiento inmediato, por lo que es más eficiente.

Suponga que el microprocesador ejecuta una instrucción LEA BX,[DI] y que DI contiene 1000H. Como DI contiene la dirección de desplazamiento, el microprocesador transfiere una copia de DI hacia BX. Una instrucción MOV BX,DI realiza esta tarea en menos tiempo, por lo que se utiliza con más frecuencia que la instrucción LEA BX,[DI].

Otro ejemplo es LEA SI,[BX+DI]. Esta instrucción suma BX a DI y almacena la suma en el registro SI. La suma generada por esta instrucción es una suma módulo 64 K. (Una **suma módulo 64 K** descarta el acarreo del resultado de 16 bits.) Si BX = 1000H y DI = 2000H, la dirección de desplazamiento que se mueve a SI es 3000H. Si BX = 1000H y DI = FF00H, la dirección de desplazamiento es 0F00H en vez de 10F00H. El segundo resultado es una suma módulo 64 K de 0F00H.

LDS, LES, LFS, LGS Y LSS

Estas instrucciones cargan cualquier registro de 16 o de 32 bits con una dirección de desplazamiento y el registro de segmento DS, ES, FS, GS o SS con una dirección de segmento. Estas instrucciones utilizan cualquiera de los modos de direccionamiento de memoria para acceder a una sección de 32 o de 48 bits de memoria que contiene tanto la dirección de segmento como la de desplazamiento. La sección de 32 bits de memoria contiene una dirección de desplazamiento y una dirección de segmento de 16 bits, mientras que la sección de 48 bits contiene una dirección de desplazamiento de 32 bits y una dirección de segmento. Estas instrucciones no pueden utilizar el modo de direccionamiento de registros (MOD = 11), y están solamente disponibles en los microprocesadores 80386 y superiores, al igual que los registros de 32 bits.

La figura 4-15 ilustra una instrucción LDS BX,[DI] de ejemplo. Esta instrucción transfiere hacia los registros BX y DS el número de 32 bits que direcciona DI en el segmento de datos. Las instrucciones LDS, LES, LFS, LGS y LSS obtienen una nueva dirección lejana (far) de la memoria. La dirección de desplazamiento aparece primero, seguida de la dirección de segmento. Este formato se utiliza para almacenar todas las direcciones de memoria de 32 bits.

El ensamblador puede almacenar en memoria una dirección lejana (far). Por ejemplo, la instrucción DIREC DD FAR PTR RANA almacena en la posición DIREC la dirección de desplazamiento y de segmento (dirección lejana) de RANA en 32 bits de memoria. La directiva DD indica al ensamblador que debe almacenar una doble palabra (número de 32 bits) en la dirección de memoria DIREC.

En los microprocesadores 80386 y superiores, una instrucción LDS EBX,[DI] carga EBX en base a la sección de 4 bytes de memoria direccionada por DI en el segmento de datos. Después de este desplazamiento de 4 bytes hay una palabra que se carga en el registro DS. Observe que, en vez de direccionar una sección de memoria de 32 bits, los microprocesadores 80386 y superiores direccionan una sección de 48 bits de la memoria siempre que se carga una dirección de desplazamiento de 32 bits en un registro de 32 bits. Los primeros cuatro bytes contienen el valor de desplazamiento que se carga en el registro de 32 bits y los últimos dos bytes contienen la dirección de segmento.

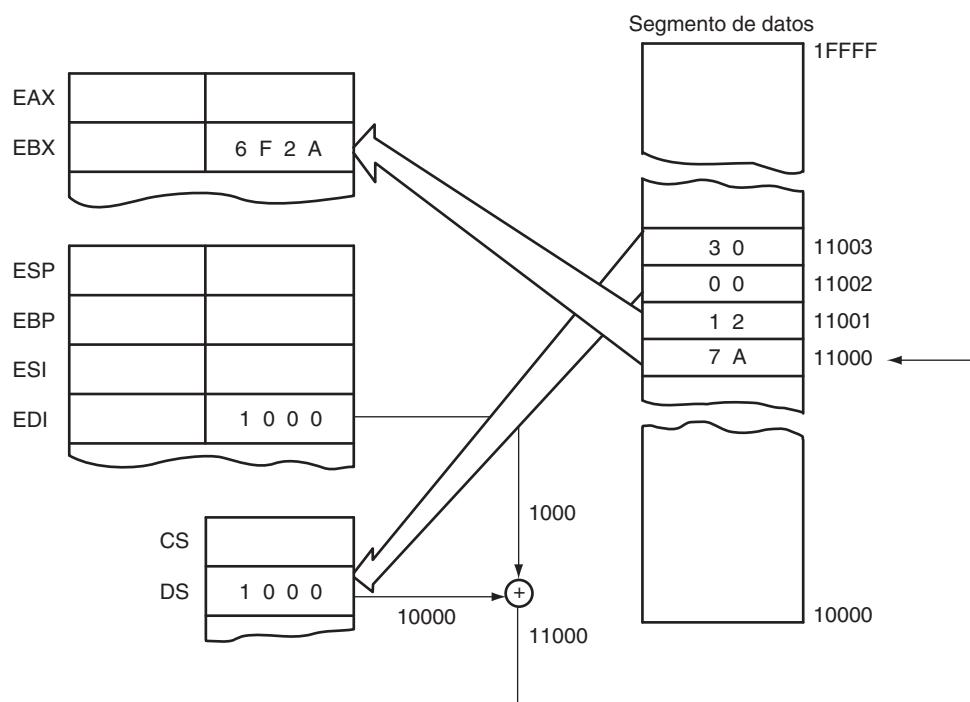


FIGURA 4-15 La instrucción LDS BX,[DI] carga el registro BX en base a las direcciones 11000H y 11001H, y el registro DS en base a la posiciones 11002H y 11003H. Esta instrucción se muestra en el punto justo antes de que DS cambie a 3000H y que BX cambie a 127AH.

La más útil de todas las instrucciones de carga es LSS. El ejemplo 4-4 muestra un programa corto que crea una nueva área de pila después de guardar la dirección del área de pila anterior. Después de ejecutar algunas instrucciones falsas se reactiva el área antigua de la pila al cargar SS y SP mediante la instrucción LSS. Tenga en cuenta que deben incluirse las instrucciones CLI (**deshabilita interrupciones**) y STI (**habilita interrupciones**) para deshabilitar las interrupciones. (Hablaremos sobre este tema casi al final de este capítulo.) Como la instrucción LSS sólo funciona en los microprocesadores 80386 y superiores, la instrucción .386 aparece después de la instrucción .MODEL para seleccionar el microprocesador 80386. Observe cómo se utiliza la directiva WORD PTR para sustituir la definición de doble palabra (DD) por la posición de memoria de la pila anterior. Si se utiliza un microprocesador 80386 o más reciente, se sugiere el uso del modificador .386 para desarrollar software para el microprocesador 80386. Esto se aplica aún y cuando el microprocesador sea Pentium, Pentium Pro, Pentium II, Pentium III o Pentium 4. La razón es que los microprocesadores del 80486 al Pentium 4 agregan sólo unas cuantas instrucciones al conjunto de instrucciones del 80386, las cuales se utilizan raras veces en el desarrollo de software. Si hay necesidad de utilizar cualquiera de las instrucciones CMPXCHG, CMPXCHG8 (nueva para el Pentium), XADD o BSWAP, se debe seleccionar el modificador .486 para el microprocesador 80486 o el modificador .586 para el Pentium. Incluso se puede especificar el Pentium II-Pentium 4 mediante el modificador .686.

EJEMPLO 4-4

```

        .MODEL SMALL           ;selecciona el modelo pequeño
        .386                  ;selecciona el 80386
0000          .DATA          ;inicio del segmento de datos
0000 00000000  DIRECS   DD    ?      ;dirección anterior de la pila
0004 1000 [     AREAS    DW    1000H DUP(?) ;nueva área de la pila
        ???
        ]
2004 = 2004      PILASUP  EQU THIS WORD ;define la parte superior de
                                                la nueva pila
0000          .CODE          ;inicio del segmento de código
0010 FA          .STARTUP       ;inicio del programa
0011 8B C4          CLI           ;deshabilita las interrupciones
0013 A3 0000 R     MOV AX,SP      ;guarda el valor anterior de SP
0016 8C D0          MOV WORD PTR SADDR,AX
0018 A3 0002 R     MOV AX,SS      ;guarda el valor anterior de SS
                           MOV WORD PTR SADDR+2,AX

001B 8C D8          MOV AX,DS      ;carga el nuevo valor de SS
001D 8E D0          MOV SS,AX      ;carga el nuevo valor de SP
001F B8 2004 R     MOV AX,OFFSET STOP
0022 8B E0          MOV SP,AX      ;habilita las interrupciones
0024 FB             STI           ;ejecuta varias instrucciones
                                                falsas
0025 8B C0          MOV AX,AX
0027 8B C0          MOV AX,AX
0029 9F B2 26 0000 R LSS SP,SADDR ;obtiene pila anterior
                                     .EXIT          ;salida al DOS
                                     END           ;fin del listado del programa

```

Existen cinco instrucciones para la transferencia de datos de cadenas: LODS, STOS, MOVS, INS y OUTS. Cada instrucción de cadena permite transferencias de datos ya sea de un solo byte, de una palabra o de una doble palabra (o si se repite, de un bloque de bytes, de palabras o de dobles palabras). Antes de ver las instrucciones de cadena se debe comprender la operación del bit de bandera D (dirección), de DI y de SI, ya que se aplican a estas instrucciones.

La bandera de dirección

Esta bandera (D, que se ubica en el registro de banderas) selecciona la operación de autoincremento ($D = 0$) o de autodeCREMENTO ($D = 1$) para los registros DI y SI durante las operaciones de cadenas. La bandera de dirección se utiliza sólo con las instrucciones de cadenas. La instrucción CLD borra la bandera D ($D = 0$) y la instrucción STD la establece ($D = 1$). Por lo tanto, la instrucción CLD selecciona el modo de autoincremento ($D = 0$) y STD selecciona el modo de autodeCREMENTO ($D = 1$).

Siempre que una instrucción de cadena transfiere un byte, el contenido de DI y/o de SI se incrementa o se decrementa en 1. Si se transfiere una palabra, el contenido de DI y/o de SI se incrementa o se decrementa en 2. Las transferencias de dobles palabras hacen que DI y/o SI se incrementen o decremenTEN en 4. Sólo se incrementan o se decrementan los registros actuales que utilice la instrucción de cadena. Por ejemplo, la instrucción STOSB utiliza el registro DI para direccionar una posición de memoria. Cuando se ejecuta STOSB, sólo DI se incrementa o se decrementa sin afectar a SI. Lo mismo se aplica para la instrucción LODSB, la cual usa el registro SI para direccionar datos de la memoria. LODSB sólo incrementa o decrementa SI, sin afectar a DI.

DI y SI

Durante la ejecución de una instrucción de cadena, los accesos a la memoria se hacen a través de los registros DI y/o SI. La dirección de desplazamiento de DI accede a los datos en el segmento extra para todas las instrucciones de cadena que la utilizan. La dirección de desplazamiento de SI accede a los datos (de manera predeterminada) en el segmento de datos. La asignación del segmento de SI puede cambiarse mediante un prefijo de sustitución de segmentos, como veremos más adelante en este capítulo. La asignación del segmento de DI siempre es en el segmento extra cuando se ejecuta una instrucción de cadena. Esta asignación no puede cambiarse. La razón por la que un apuntador dirige datos en el segmento extra y el otro apuntador en el segmento de datos es para que la instrucción MOVS pueda mover 64 Kbytes de datos de un segmento de memoria a otro.

Al operar en el modo de 32 bits en los microprocesadores 80386 o superiores, se utilizan los registros EDI y ESI en vez de DI y SI. Esto permite que las cadenas utilicen cualquier posición de memoria en todo el espacio de direcciones de 4 Gbytes del microprocesador en modo protegido.

LODS

Esta instrucción carga AL, AX o EAX con los datos almacenados en la dirección de desplazamiento del segmento de datos indizado por el registro SI. (Sólo los microprocesadores 80386 y superiores utilizan EAX.) Después de cargar AL con un byte, AX con una palabra o EAX con una doble palabra, el contenido de SI se incrementa si $D = 0$, o se decrementa si $D = 1$. Se suma o se resta un 1 a SI para una instrucción LODS tipo byte, se suma o se resta un 2 para una instrucción LODS tipo palabra, y se suma o se resta un 4 para una instrucción LODS tipo doble palabra.

La tabla 4-10 lista las formas permitidas de la instrucción LODS. La instrucción LODSB (**carga un byte**) hace que se cargue un byte en AL, la instrucción LODSW (**carga una palabra**) hace que se

TABLA 4-10 Formas de la instrucción LODS.

Lenguaje ensamblador	Operación
LODSB	$AL = DS:[SI]; SI = SI \pm 1$
LODSW	$AX = DS:[SI]; SI = SI \pm 2$
LODSD	$EAX = DS:[SI]; SI = SI \pm 4$
LODS LISTA	$AL = DS:[SI]; SI = SI \pm 1$ (si LISTA es un byte)
LODS DATOS1	$AX = DS:[SI]; SI = SI \pm 2$ (si DATOS1 es una palabra)
LODS RANA	$EAX = DS:[SI]; SI = SI \pm 4$ (si RANA es una doble palabra)

Nota: El registro de segmento puede sustituirse mediante un prefijo de sustitución de segmento, como en LODS ES:DATOS4.

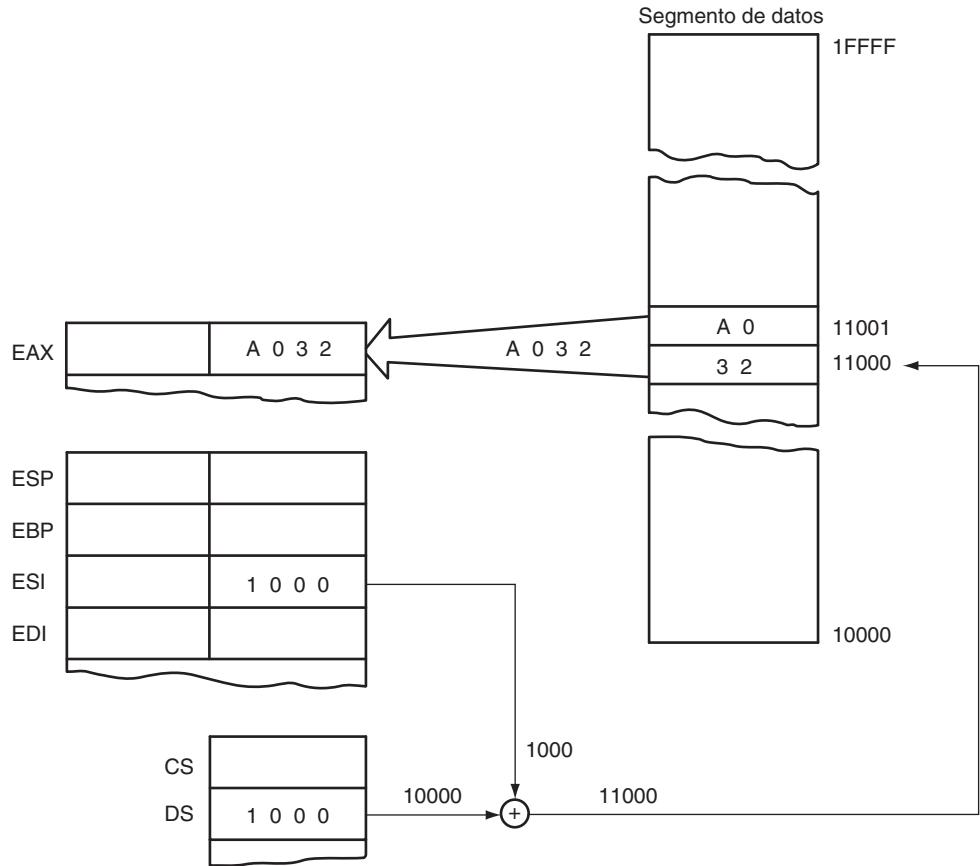


FIGURA 4-16 La operación de la instrucción LODSW si DS = 1000H, D = 0, 11000H = 32 y 11001H = A0. Esta instrucción se muestra después de que AX se carga de la memoria, pero antes de que SI se incremente en 2.

cargue una palabra en AX y la instrucción LODSD (**carga una doble palabra**) hace que se cargue una doble palabra en EAX. Aunque se usa raras veces, una alternativa para LODSB, LODSW y LODSD es la instrucción LODS seguida de un operando tipo byte, palabra o doble palabra para seleccionar una transferencia de byte, de palabra o de doble palabra. Los operandos se definen a menudo como bytes con DB, como palabras con DW y como dobles palabras con DD. La seudooperación DB define bytes, la seudooperación DW define palabras y la seudooperación DD define dobles palabras.

La figura 4-16 muestra el efecto que se produce al ejecutar al instrucción LODSW si la bandera D = 0, SI = 1000H y DS = 1000H. Aquí, un número de 16 bits que se almacena en las posiciones de memoria 11000H y 11001H se mueve hacia AX. Como D = 0 y ésta es una transferencia de palabras, el contenido de SI se incrementa en 2 después de que AX se carga con datos de la memoria.

STOS

Esta instrucción almacena AL, AX o EAX en la posición de memoria del segmento extra direccionada por el registro DI. (Sólo los microprocesadores del 80386 al Pentium 4 utilizan EAX y dobles palabras.) La tabla 4-11 lista todas las formas de la instrucción STOS. Al igual que con LODS, se puede adjuntar una B, una W o una D a una instrucción STOS para realizar transferencias tipo byte, palabra o doble palabra. La instrucción STOSB (**almacena un byte**) almacena el byte que está en AL en la posición de memoria del segmento extra direccionada por DI. La instrucción STOSW (**guarda una palabra**) almacena AX en la posición de memoria del segmento extradireccinada por DI. Con la instrucción STOSD (**almacena una doble palabra**) se almacena una doble palabra en la posición del segmento

TABLA 4-11 Formas de la instrucción STOS.

<i>Lenguaje ensamblador</i>	<i>Operación</i>
STOSS	ES:[DI] = AL; DI = DI ± 1
STOSW	ES:[DI] = AX; DI = DI ± 2
STOSD	ES:[DI] = EAX; DI = DI ± 4
STOS LISTA	ES:[DI] = AL; DI = DI ± 1 (si LISTA es un byte)
STOS DATOS3	ES:[DI] = AX; DI = DI ± 2 (si DATOS3 es una palabra)
STOS DATOS4	ES:[DI] = EAX; DI = DI ± 4 (si DATOS4 es una doble palabra)

extradireccional por DI. Después de almacenar el byte (AL), la palabra (AX) o la doble palabra (EAX) se incrementa o decrementa el contenido de DI.

STOS con un REP. El **prefijo de repetición** (REP) se agrega a cualquier instrucción de transferencia de datos de cadena, excepto a la instrucción LODS. No tiene sentido ejecutar una instrucción LODS repetida. El prefijo REP hace que CX se decremente en 1 cada vez que se ejecuta la instrucción de cadena. Después de decrementar CX se repite la instrucción de cadena. Si CX llega a un valor de 0, la instrucción termina y el programa continúa con la siguiente instrucción secuencial. Por lo tanto, si CX se carga con 100 y se ejecuta una instrucción REP STOSB, el microprocesador repite automáticamente la instrucción STOSB 100 veces. Como el registro DI se incrementa o decrementa en forma automática después de almacenar cada dato, esta instrucción almacena el contenido de AL en un bloque de memoria, en vez de hacerlo en un solo byte de memoria.

Suponga que se utiliza la instrucción STOSW para borrar un área de memoria llamada Bufer mediante el uso de un contador llamado Contador y que el programa va a llamar a la función BorraBufer en el entorno C++ mediante el uso del ensamblador en línea. (Vea el ejemplo 4-5.) Observe que se transfieren las direcciones de Contador y Bufer a la función. La instrucción REP STOSW borra el búfer de memoria llamado Bufer. Observe que Bufer es un apuntador al búfer actual que se borra mediante esta función.

EJEMPLO 4-5

```
void BorraBufer (int Contador, short* Bufer)
{
    __asm{
        push edi           ;guarda los registros
        push es
        push ds
        mov ax,0
        mov ecx, Contador
        mov edi, Bufer
        pop es            ;carga ES con DS
        rep stosw          ;borra el Bufer
        pop es            ;restaura los registros
        pop edi
    }
}
```

Los operandos en un programa pueden modificarse mediante el uso de operadores aritméticos o lógicos tales como la multiplicación (*). En la tabla 4-12 aparecen otros operadores.

MOVS

Una de las instrucciones de transferencia de datos de cadena más útiles es MOVS, ya que transfiere datos de una posición de memoria a otra. Ésta es la única transferencia de memoria a memoria que se permite en los microprocesadores del 8086 al Pentium 4. La instrucción MOVS transfiere un byte,

TABLA 4-12 Modificadores comunes de los operandos.

<i>Operador</i>	<i>Ejemplo</i>	<i>Comentario</i>
+	MOV AL,6+3	Copia 9 en AL
-	MOV AL,6-3	Copia 3 en AL
*	MOV AL,4*3	Copia 12 en AL
/	MOV AL,12/5	Copia 2 en AX (se pierde el residuo)
MOD	MOV AX,12 MOD 7	Copia 5 en AX (se pierde el cociente)
AND	MOV AX,12 AND 4	Copia 4 en AX (1100 AND 0100 = 0100)
OR	MOV EAX,12 OR 1	Copia 13 en EAX (1100 OR 0001 = 1101)
NOT	MOV AL,NOT 1	Copia 254 en AL (NOT 0000 0001 = 1111 1110 o 254)

una palabra o una doble palabra desde la posición del segmento de datos direccionada por SI hasta la posición del segmento extradireccinada por DI. Al igual que con las otras instrucciones de cadena, los apuntadores se incrementan o decrementan después, según lo indique la bandera de dirección. La tabla 4-13 lista todas las formas permitidas de la instrucción MOVS. Sólo el operando de origen (SI), que se encuentra en el segmento de datos, puede sustituirse de manera que pueda utilizarse otro segmento. El operando de destino (DI) siempre debe ubicarse en el segmento extra.

A menudo es necesario transferir el contenido de un área de memoria a otra. Suponga que tenemos dos bloques de memoria tipo doble palabra llamados BloqueA y BloqueB, y necesitamos copiar el BloqueA en el BloqueB. Esto puede lograrse mediante el uso de la instrucción MOVSD, como se muestra en el ejemplo 4-6, el cual es una función en lenguaje C++ escrita mediante el ensamblador en línea. La función recibe tres piezas de información del llamador: TamanioBloque y las direcciones de BloqueA y BloqueB. Observe que en Visual C++ todos los datos se encuentran en el segmento de datos, por lo que necesitamos copiar DS en ES. Para ello utilizamos una instrucción PUSH DS seguida de POP ES. También necesitamos guardar todos los registros que cambiamos, excepto EAX, EBX, ECX y EDX.

El ejemplo 4-7 muestra la misma función escrita sólo en C++, para que se puedan comparar y contrastar ambos métodos. El ejemplo 4-8 muestra la versión en lenguaje ensamblador del ejemplo 4-7 para compararlo con el ejemplo 4-6. Observe que la versión en lenguaje ensamblador es más corta que la versión en C++ generada en el ejemplo 4-8. Es cierto que la versión en C++ es más fácil de escribir, pero si la velocidad de ejecución es importante el ejemplo 4-6 se ejecutará con mucha mayor rapidez que el ejemplo 4-7.

TABLA 4-13 Formas de la instrucción MOVS.

<i>Lenguaje ensamblador</i>	<i>Operación</i>
MOVSB	ES:[DI] = DS[SI]; DI = DI ± 1; SI = SI ± 1 (se transfiere un byte).
MOVSW	ES:[DI] = DS[SI]; DI = DI ± 2; SI = SI ± 2 (se transfiere una palabra).
MOVSD	ES:[DI] = DS[SI]; DI = DI ± 4; SI = SI ± 4 (se transfiere una doble palabra).
MOVS BYTE1,BYTE2	ES:[DI] = DS[SI]; DI = DI ± 1; SI = SI ± 1 (se transfiere un byte si BYTE1 y BYTE2 son bytes).
MOVS PALABRA1,PALABRA2	ES:[DI] = DS[SI]; DI = DI ± 2; SI = SI ± 2 (se transfiere una palabra si PALABRA1 y PALABRA2 son palabras).
MOVS TED, FRED	ES:[DI] = DS[SI]; DI = DI ± 4; SI = SI ± 4 (se transfiere una doble palabra si TED y FRED son dobles palabras).

EJEMPLO 4-6

```
//Función que copia BloqueA en BloqueB mediante el uso del ensamblador en línea
//
void TransfiereBloques (int TamanioBloque, int* BloqueA, int* BloqueB)
{
    _asm{
        push es                      ;guarda los registros
        push edi
        push esi
        push ds                      ;copia DS en ES
        pop  es
        mov  esi, BloqueA            ;direcciona BloqueA
        mov  edi, BloqueB            ;direcciona BloqueB
        mov  ecx, TamanioBloque      ;carga el contador
        rep  movsd                  ;muestra los datos
        pop  es                      ;restaura los registros
        pop  esi
        pop  edi
    }
}
```

EJEMPLO 4-7

```
//Versión en C++ del ejemplo 4-6
//
void TransfiereBloques (int TamanioBloque, int* BloqueA, int* BloqueB)
{
    for (int a = 0; a < TamanioBloque; a++)
    {
        BloqueA = BloqueB++
        BloqueA++;
    }
}
```

EJEMPLO 4-8

```
void TransfiereBloques(int TamanioBloque, int * BloqueA, int* BloqueB)
{
004136A0    push      ebp
004136A1    mov       ebp,esp
004136A3    sub       esp,0D8h
004136A9    push      ebx
004136AA    push      esi
004136AB    push      edi
004136AC    push      ecx
004136AD    lea       edi,[ebp-0D8h]
004136B3    mov       ecx,36h
004136B8    mov       eax, 0CCCCCCCCCh
004136BD    rep stos   dword ptr [edi]
004136BF    pop       ecx
004136C0    mov       dword ptr [ebp-8],ecx
    for( int a = 0; a < TamanioBloque; a++ )
004136C3    mov       dword ptr [a], 0
004136CA    jmp       TransfiereBloques+35h (4136D5h)
004136CC    mov       eax,dword ptr [a]
004136CF    add       eax,1
004136D2    mov       dword ptr [a], eax
004136D5    mov       eax,dword ptr [a]
004136D8    cmp       eax,dword ptr [TamanioBloque]
004136DB    jge       TransfiereBloques+57h (4136F7h)
    {
        BloqueA = BloqueB++;
004136DD    mov       eax,dword ptr [BloqueB]
004136E0    mov       dword ptr [BloqueA],eax
004136E3    mov       ecx,dword ptr [BloqueB]
004136E6    add       ecx,4
004136E9    mov       dword ptr [BloqueB],ecx
}
```

```

        BloqueA++;
004136EC    mov      eax,dword ptr [BloqueA]
004136EF    add      eax,4
004136F2    mov      dword ptr [BloqueA],eax
}
004136F5    jmp      TransfiereBloques+2Ch (4136CCh)
}
004136F7    pop     edi
004136F8    pop     esi
004136F9    pop     ebx
004136FA    mov      esp,ebp
004136FC    pop     ebp
004136FD    ret      0Ch

```

INS

La instrucción INS (**cadena de entrada**) (que no está disponible en los microprocesadores 8086/8088) transfiere un byte, una palabra o una doble palabra de datos desde un dispositivo de E/S hacia la posición de memoria del segmento extra direccionada por el registro DI. La dirección de E/S está contenida en el registro DX. Esta instrucción es útil para introducir un bloque de datos desde un dispositivo de E/S de manera directa hacia la memoria. Una aplicación transfiere datos desde una unidad de disco hacia la memoria. Con frecuencia las unidades de disco se consideran y se conectan como dispositivos de E/S en un sistema computacional.

Al igual que con las instrucciones de cadena anteriores, existen tres formas básicas de la instrucción INS. La instrucción INSB introduce datos desde un dispositivo de E/S de 8 bits y los almacena en la posición de memoria tipo byte indizado por SI. La instrucción INSW introduce datos de E/S de 16 bits y los almacena en una posición de memoria tipo palabra. La instrucción INSD introduce una doble palabra. Estas instrucciones pueden repetirse mediante el prefijo REP, el cual permite que se almacene todo un bloque completo de datos en la memoria desde un dispositivo de E/S. La tabla 4-14 lista las diversas formas de la instrucción INS.

El ejemplo 4-9 muestra una secuencia de instrucciones que introducen 50 bytes de datos desde un dispositivo de E/S cuya dirección es 03ACH y almacenan los datos en el arreglo de memoria LISTAS del segmento extra. Este software supone que los datos están disponibles del dispositivo de E/S en todo momento. De no ser así, el software debe comprobar si el dispositivo de E/S está listo para transferir los datos antes de utilizar un prefijo REP.

EJEMPLO 4-9

```

;Uso de la instrucción REP INSB para introducir datos en un
arreglo de memoria
;
0000 BF 0000 R      MOV DI,OFFSET LISTAS ;direcciona el arreglo
0003 BA 03AC        MOV DX,3ACH   ;direcciona el dispositivo de E/S
0006 FC             CLD          ;autoincremento
0007 B9 0032        MOV CX,50   ;carga el contador
000A F3/6C          REP INSB    ;introduce los datos

```

TABLA 4-14 Formas de la instrucción INS.

Lenguaje ensamblador	Operación
INSB	ES:[DI] = [DX]; DI = DI ± 1 (se transfiere un byte)
INSW	ES:[DI] = [DX]; DI = DI ± 2 (se transfiere una palabra)
INSD	ES:[DI] = [DX]; DI = DI ± 4 (se transfiere una doble palabra)
INS LISTA	ES:[DI] = [DX]; DI = DI ± 1 (si LISTA es un byte)
INS DATOS4	ES:[DI] = [DX]; DI = DI ± 2 (si DATOS4 es una palabra)
INS DATOS5	ES:[DI] = [DX]; DI = DI ± 4 (si DATOS5 es una doble palabra)

Nota: [DX] indica que DX es la dirección del dispositivo de E/S. Estas instrucciones no están disponibles en los microprocesadores 8086 y 8088.

TABLA 4-15 Formas de la instrucción OUTS.

Lenguaje ensamblador	Operación
OUTSB	[DX] = DS:[SI]; SI = SI ± 1 (se transfiere un byte)
OUTSW	[DX] = DS:[SI]; SI = SI ± 2 (se transfiere una palabra)
OUTSD	[DX] = DS:[SI]; SI = SI ± 4 (se transfiere una doble palabra)
OUTS DATOS7	[DX] = DS:[SI]; SI = SI ± 1 (si DATOS7 es un byte)
OUTS DATOS8	[DX] = DS:[SI]; SI = SI ± 2 (si DATOS8 es una palabra)
OUTS DATOS9	[DX] = DS:[SI]; SI = SI ± 4 (si DATOS9 es una doble palabra)

Nota: [DX] indica que DX es la dirección del dispositivo de E/S. Estas instrucciones no están disponibles en los microprocesadores 8086 y 8088.

OUTS

La instrucción OUTS (**cadena de salida**) (no está disponible en los microprocesadores 8086/8088) transfiere hacia un dispositivo de E/S un byte, una palabra o una doble palabra de datos desde la posición de memoria del segmento de datos direccionada por SI. El dispositivo de E/S se dirige mediante el registro DX, de igual forma que en la instrucción INS. La tabla 4-15 muestra las variaciones disponibles para la instrucción OUTS.

El ejemplo 4-10 muestra una secuencia corta de instrucciones que transfieren datos desde un arreglo de memoria (ARREGLO) del segmento de datos hacia un dispositivo de E/S en la dirección de E/S 3ACH. Este software supone que el dispositivo de E/S ya está listo para la transferencia de datos.

EJEMPLO 4-10

```
;Uso de la instrucción REP OUTSB para enviar datos desde un
;arreglo de memoria
;
0000 BE 0064 R      MOV SI,OFFSET ARREGLO ;direcciona el arreglo
0003 BA 03AC        MOV DX,3ACH          ;direcciona el dispositivo de E/S
0006 FC             CLD                 ;autoincremento
0007 B9 0064        MOV CX,100          ;carga el contador
000A F3/6E          REP OUTSB           ;envía los datos
```

4-5

INSTRUCCIONES VARIADAS DE TRANSFERENCIA DE DATOS

No se deje engañar por el término *variadas*; estas instrucciones se utilizan en programas. Las instrucciones de transferencia de datos que veremos en esta sección son XCHG, LAHF, SAHF, XLAT, IN, OUT, BSWAP, MOVSX, MOVZX y CMOV. Como las instrucciones variadas no se utilizan con tanta frecuencia como la instrucción MOV, las hemos agrupado para presentarlas en esta sección.

XCHG

La instrucción XCHG (**intercambio**) intercambia el contenido de un registro con el contenido de otro registro o posición de memoria. La instrucción XCHG no puede intercambiar registros de segmentos o datos de memoria a memoria. Los intercambios pueden ser tipo byte, palabra o doble palabra (en el 80386 y superiores), y utilizan cualquiera de los modos de direccionamiento que vimos en el capítulo 3, excepto el direccionamiento inmediato. La tabla 4-16 muestra algunos ejemplos de la instrucción XCHG.

Esta instrucción, cuando utiliza el registro AX de 16 bits con otro registro de 16 bits, produce el intercambio más eficiente. Esta instrucción ocupa un byte de memoria. Otras instrucciones XCHG requieren dos o más bytes de memoria, dependiendo del modo de direccionamiento seleccionado.

Cuando se utiliza un modo de direccionamiento de memoria y el ensamblador no importa cuál operando dirige la memoria. Para el ensamblador, la instrucción XCHG AL,[DI] es idéntica a la instrucción XCHG [DI],AL.

TABLA 4-16 Formas de la instrucción XCHG.

<i>Lenguaje ensamblador</i>	<i>Operación</i>
XCHG AL,CL	Intercambia el contenido de AL con CL.
XCHG CX,BP	Intercambia el contenido de CX con BP.
XCHG EDX,ESI	Intercambia el contenido de EDX con ESI.
XCHG AL,DATOS2	Intercambia el contenido de AL con la posición DATOS2 del segmento de memoria.

Si está disponible uno de los microprocesadores del 80386 al Pentium 4, la instrucción XCHG puede intercambiar datos tipo doble palabra. Por ejemplo, la instrucción XCHG EAX,EBX intercambia el contenido del registro EAX con el registro EBX.

LAHF y SAHF

Estas instrucciones se utilizan pocas veces, ya que se diseñaron como instrucciones puente. Estas instrucciones permitían traducir el software para el 8085 (uno de los primeros microprocesadores de 8 bits) en software para el 8086 mediante un programa de traducción. Como todo el software que requería de traducción se terminó hace muchos años, estas instrucciones tienen poca aplicación en la actualidad. La instrucción LAHF transfiere los ocho bits de más a la derecha del registro de banderas hacia el registro AH. La instrucción SAHF transfiere el registro AH hacia los ocho bits de más a la derecha del registro de banderas.

A veces, la instrucción SAHF puede tener alguna aplicación con el coprocesador numérico. Este coprocesador contiene un registro de estado que se copia en el registro AX con la instrucción FSTSW AX. La instrucción SAHF se utiliza entonces para copiar del registro AX hacia el registro de banderas. Después las banderas se evalúan para alguna de las condiciones del coprocesador numérico. Esto lo veremos en el capítulo 14, en donde explicaremos la operación y la programación del coprocesador numérico.

XLAT

La instrucción XLAT (**traducción**) convierte el contenido del registro AL en un número que se almacena en una tabla en memoria. Esta instrucción ejecuta la técnica de búsqueda directa en la tabla que se utiliza a menudo para convertir un código en otro. Una instrucción XLAT primero suma el contenido de AL a BX para formar una dirección de memoria dentro del segmento de datos. Después copia el contenido de esta dirección en AL. Ésta es la única instrucción que suma un número de 8 bits a un número de 16 bits.

Suponga que se almacena una tabla de búsqueda de pantalla LED de siete segmentos en la memoria, en la dirección TABLA. La instrucción XLAT utiliza entonces la tabla de búsqueda para traducir el número BCD que está en AL a un código de siete segmentos en AL. El ejemplo 4-11 proporciona una secuencia de instrucciones que convierten de un código BCD a un código de siete segmentos. La figura 4-17 muestra la operación de este programa de ejemplo si TABLA = 1000H, DS = 1000H y el valor inicial de AL = 05H (5 BCD). Después de la traducción, AL = 6DH.

EJEMPLO 4-11

```

TABLA  DB  3FH, 06H, 5BH, 4FH      ;tabla de búsqueda
        DB  66H, 6DH, 7DH, 27H
        DB  7FH, 6FH

0017 B0 05      BUSCA: MOV  AL,5          ;carga AL con 5 (un número de prueba)
0019 BB 1000 R    MOV  BX,OFFSET TABLA ;direcciona la tabla de búsqueda
001C D7          XLAT                 ;convierte

```

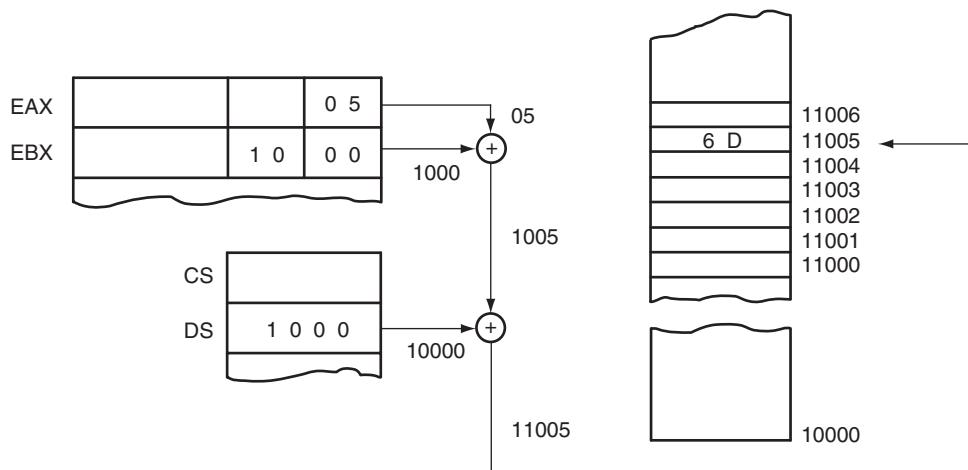


FIGURA 4-17 La operación de la instrucción XLAT en el punto justo antes de que se cargue 6DH en AL.

IN y OUT

La tabla 4-17 lista las formas de las instrucciones IN y OUT, las cuales realizan operaciones de E/S. Observe que el contenido de AL, AX o EAX se transfiere solamente entre el dispositivo de E/S y el microprocesador. Una instrucción IN transfiere datos desde un dispositivo de E/S externo hacia AL, AX o EAX; una instrucción OUT transfiere datos desde AL, AX o EAX hacia un dispositivo de E/S externo. (Tenga en cuenta que sólo los microprocesadores 80386 y superiores contienen el registro EAX.)

Existen dos formas de direccionamiento de dispositivos (puertos) de E/S para IN y OUT: puerto fijo y puerto variable. El *direcciónamiento de puerto fijo* permite la transferencia de datos entre AL, AX o EAX mediante el uso de una dirección de puerto de E/S de 8 bits. Se llama direcciónamiento de puerto fijo porque el número de puerto va después del código de operación de la instrucción, como en el direcciónamiento inmediato. A menudo las instrucciones se guardan en ROM. Una instrucción de puerto fijo almacenada en ROM tiene su número de puerto fijo de manera permanente, debido a la naturaleza de la memoria de sólo lectura. Una dirección de puerto fijo almacenada en RAM puede modificarse, pero dicha modificación no se conforma a las buenas prácticas de programación.

TABLA 4-17
Las instrucciones IN
y OUT.

Lenguaje ensamblador	Operación
IN AL,p8	Se introducen 8 bits en AL desde el puerto de E/S p8.
IN AX,p8	Se introducen 16 bits en AX desde el puerto de E/S p8.
IN EAX,p8	Se introducen 32 bits en EAX desde el puerto de E/S p8.
IN AL,DX	Se introducen 8 bits en AL desde el puerto de E/S DX.
IN AX,DX	Se introducen 16 bits en AX desde el puerto de E/S DX.
IN EAX,DX	Se introducen 32 bits en EAX desde el puerto de E/S DX.
OUT p8,AL	Se envían 8 bits al puerto de E/S p8 desde AL.
OUT p8,AX	Se envían 16 bits al puerto de E/S p8 desde AX.
OUT p8,EAX	Se envían 32 bits al puerto de E/S p8 desde EAX.
OUT DX,AL	Se envían 8 bits al puerto de E/S DX desde AL.
OUT DX,AX	Se envían 16 bits al puerto de E/S DX desde AX.
OUT DX,EAX	Se envían 32 bits al puerto de E/S DX desde EAX.

Nota: p8 = un número de puerto de E/S de 8 bits (de 0000H a 00FFH) y DX = el número de puerto de E/S de 16 bits (de 0000H a FFFFH) que se guarda en el registro DX.

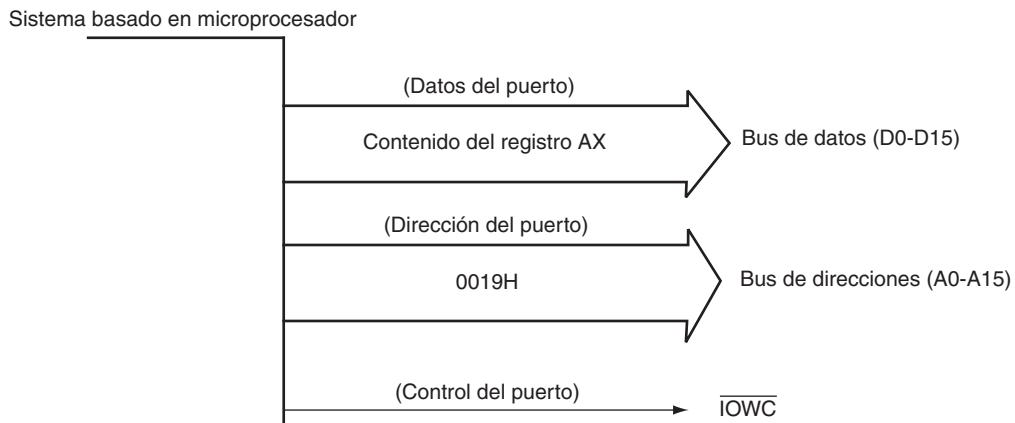


FIGURA 4-18 Las señales del sistema basado en microprocesador para una instrucción OUT 19H,AX.

Durante una operación de E/S, la dirección del puerto aparece en el bus de direcciones. Para las instrucciones de E/S con puerto fijo de 8 bits, la dirección del puerto de 8 bits se extiende con ceros para convertirse en una dirección de 16 bits. Por ejemplo, si se ejecuta la instrucción IN AL,6AH se introducen en AL los datos de la dirección de E/S 6AH. La dirección aparece como el número 006AH de 16 bits en las terminales A0-A15 para el bus de direcciones. Los bits del bus de direcciones A16-A19 (8086/8088), A16-A23 (80286/80386SX), A16-A24 (80386SL/80386SLC/80386EX) o A16-A31 (80386-Pentium 4) están indefinidos para una instrucción IN o OUT. Tenga en cuenta que Intel se reserva los últimos 16 puertos de E/S (FFF0H-FFFFH) para usarlos con algunos de sus componentes periféricos.

El direccionamiento de puerto variable permite la transferencia de datos entre los registros AL, AX o EAX y una dirección de puerto de 16 bits. Se llama *direccionamiento de puerto variable* debido a que el número de puerto de E/S se almacena en el registro DX, el cual puede cambiarse (variar) durante la ejecución de un programa. La dirección del puerto de E/S de 16 bits aparece en las terminales A0-A15 del bus de direcciones. La IBM PC utiliza una dirección de puerto de 16 bits para acceder a su espacio de E/S. El espacio de E/S del bus ISA para una PC se encuentra en los puertos de E/S 0000H-03FFH. Las tarjetas de bus PCI pueden usar direcciones de E/S por encima de 03FFH.

La figura 4-18 ilustra la ejecución de la instrucción OUT 19H,AX que transfiere el contenido de AX al puerto de E/S 19H. El número de puerto de E/S aparece como un 0019H en el bus de direcciones de 16 bits y los datos de AX aparecen en el bus de datos del microprocesador. La señal \overline{IOWC} de control del sistema (control de escritura de E/S) es un cero lógico para habilitar el dispositivo de E/S.

En el ejemplo 4-12 aparece un programa corto que emite un sonido de “clic” en la bocina de la computadora personal. Esta bocina (en el DOS solamente) se controla mediante el acceso al puerto de E/S 61H. Si los dos bits de más a la derecha de este puerto están activados (11) y luego se borran (00), se escucha un clic en la bocina. Este programa utiliza una instrucción OR lógica para establecer estos dos bits, y una instrucción AND lógica para borrarlos. En el capítulo 5 describiremos el funcionamiento de estas operaciones lógicas. La instrucción MOV CX,8000H, seguida de la instrucción LOOP L1, se utiliza como un retraso de tiempo. Si se incrementa la cuenta, el clic se hará más largo; si se acorta la cuenta, el clic se hará más corto. Para obtener una serie de clics que puedan escucharse, se debe modificar el programa para repetirse muchas veces.

EJEMPLO 4-12

```

        .MODEL TINY          ;selecciona el modelo diminuto
0000      .CODE            ;inicio del segmento de código
              .STARTUP       ;inicio del programa
0100  E4 61      IN AL,61H   ;lee el puerto de E/S 61H
0102  0C 03      OR AL,3    ;establece los dos bits de más a la derecha
0104  E6 61      OUT 61H,AL  ;enciende la bocina
0106  B9 8000    MOV CX,8000H ;carga el contador de retraso
0109      L1:           LOOP L1    ;retraso de tiempo
0109  E2 FE

```

TABLA 4-18 Las instrucciones MOVSX y MOVZX.

Lenguaje ensamblador	Operación
MOVSX CX,BL	Extiende BL con signo hacia CX.
MOVSX ECX,AX	Extiende AX con signo hacia ECX.
MOVSX BX,DATOS1	Extiende con signo el byte en DATOS1 hacia BX.
MOVSX EAX,[EDI]	Extiende con signo la palabra en la posición de memoria del segmento de datos direccionada por EDI hacia EAX.
MOVZX DX,AL	Extiende con ceros AL hacia DX.
MOVZX EBP,DI	Extiende con ceros DI hacia EBP.
MOVZX DX,DATOS2	Extiende con ceros el byte en DATOS2 hacia DX.
MOVZX EAX,DATOS3	Extiende con ceros la palabra en DATOS3 hacia EAX.

```

010B E4 61      IN AL, 61H      ;apaga la bocina
010D 24 FC      AND AL, 0FCH
010F E6 61      OUT 61H, AL
.EXIT
END

```

MOVSX y MOVZX

Las instrucciones MOVZX (**movimiento y extensión con signo**) y MOVZC (**movimiento y extensión con ceros**) se encuentran en los conjuntos de instrucciones de los microprocesadores del 80386 al Pentium 4. Estas instrucciones mueven datos y al mismo tiempo los extienden ya sea con signo o con ceros. La tabla 4-18 muestra estas instrucciones con varios ejemplos de cada una.

Cuando un número se extiende con ceros, la parte más significativa se llena con ceros. Por ejemplo, si un 34H de 8 bits se extiende con ceros para convertirse en un número de 16 bits, se convierte en 0034H. La extensión con ceros se utiliza a menudo para convertir números sin signo de 8 o de 16 bits en números sin signo de 16 o de 32 bits, mediante el uso de la instrucción MOVZX.

Un número se extiende con signo cuando su bit de signo se copia en la parte más significativa. Por ejemplo, si un 84H de 8 bits se extiende con signo para convertirse en un número de 16 bits, se convierte en FF84H. El bit de signo de un 84H es un uno, el cual se copia en la parte más significativa del resultado con signo extendido. La extensión con signo se utiliza a menudo para convertir números con signo de 8 o de 16 bits en números con signo de 16 o de 32 bits, mediante el uso de la instrucción MOVSX.

BSWAP

La instrucción BSWAP (**intercambio de bytes**) está disponible solamente en los microprocesadores del 80486 al Pentium 4. Esta instrucción toma el contenido de cualquier registro de 32 bits e intercambia el primer byte con el cuarto, y el segundo con el tercero. Por ejemplo, la instrucción BSWAP EAX en donde EAX = 00112233H intercambia los bytes en EAH, con lo que se obtiene el resultado EAX = 33221100H. Observe que esta instrucción invierte el orden de los cuatro bytes. Esta instrucción se utiliza para convertir datos entre los formatos Little Endian y Big Endian.

CMOV

La clase de instrucción CMOV (**movimiento condicional**) es nueva para los conjuntos de instrucciones del Pentium Pro al Pentium 4. Existen muchas variaciones de la instrucción CMOV. La tabla 4-19 lista estas variaciones de CMOV. Estas instrucciones mueven los datos únicamente si se cumple la condición. Por ejemplo, la instrucción CMOVZ mueve datos sólo si el resultado de una instrucción anterior es cero. El destino se limita a registros de 16 o de 32 bits, pero el origen puede ser un registro o una posición de memoria de 16 o de 32 bits.

Como esta instrucción es nueva, no se puede utilizar con el ensamblador a menos que se agregue el modificador .686 al programa.

TABLA 4-19 Las instrucciones de movimiento condicional.

<i>Lenguaje ensamblador</i>	<i>Bandera(s) que se evalúa(n)</i>	<i>Operación</i>
CMOV _B	C = 1	Mueve si es inferior.
CMOV _{AE}	C = 0	Mueve si es superior o igual.
CMOV _{BE}	Z = 1 o C = 1	Mueve si es inferior o igual.
CMOV _A	Z = 0 y C = 0	Mueve si es superior.
CMOVE o CMOV _Z	Z = 1	Mueve si es igual o cero.
CMOVNE o CMOVNZ	Z = 0	Mueve si no es igual o si no es cero.
CMOV _L	S != 0	Mueve si es menor que.
CMOV _{LE}	Z = 1 o S != 0	Mueve si es menor o igual.
CMOV _G	Z = 0 y S = 0	Mueve si es mayor que.
CMOV _{GE}	S = 0	Mueve si es mayor o igual.
CMOV _S	S = 1	Mueve si hay signo (negativo).
CMOV _{NS}	S = 0	Mueve si no hay signo (positivo).
CMOV _C	C = 1	Mueve si hay acarreo.
CMOV _{NC}	C = 0	Mueve si no hay acarreo.
CMOV _O	O = 1	Mueve si hay desbordamiento.
CMOV _{NO}	O = 0	Mueve si no hay desbordamiento.
CMOV _P o CMOV _{PE}	P = 1	Mueve si hay paridad o si la paridad es par.
CMOVNP o CMOVPO	P = 0	Mueve si no hay paridad o si la paridad es impar.

4-6**PREFIJO DE SUSTITUCIÓN DE SEGMENTO**

Este prefijo, que puede agregarse a casi cualquier instrucción en cualquier modo de direccionamiento de memoria, permite al programador desviarse del segmento predeterminado. El prefijo de sustitución de segmento es un byte adicional que se adjunta al principio de una instrucción para seleccionar un registro de segmento alternativo. Dos de las instrucciones que no pueden utilizar este prefijo son las instrucciones de salto y de llamada que deben utilizar el registro de segmento de código para la generación de direcciones. La sustitución de segmento también se utiliza para seleccionar los segmentos FS y GS en los microprocesadores del 80386 al Pentium 4.

Por ejemplo, la instrucción MOV AX,[DI] accede a los datos dentro del segmento de datos de manera predeterminada. Si lo requiere un programa, esto puede cambiarse mediante la colocación de un prefijo en la instrucción. Suponga que los datos están en el segmento extra, en vez de estar en el segmento de datos. Esta instrucción direcciona el segmento extra si se cambia a MOV AX,ES:[DI].

La tabla 4-20 muestra algunas instrucciones alteradas que seleccionan distintos segmentos de memoria que son diferentes de los normales. Cada vez que se agrega el prefijo de sustitución de segmento a

TABLA 4-20 Instrucciones que incluyen prefijos de sustitución de segmento.

<i>Lenguaje ensamblador</i>	<i>Segmento utilizado</i>	<i>Segmento predeterminado</i>
MOV AX,DS:[BP]	Datos	Pila
MOV AX,ES:[BP]	Extra	Pila
MOV AX,SS:[DI]	Pila	Datos
MOV AX,CS:LISTA	Código	Datos
MOV ES:[SI],AX	Extra	Datos
LODS ES:DATOS1	Extra	Datos
MOV EAX,FS:DATOS2	FS	Datos
MOV GS:[ECX],BL	GS	Datos

una instrucción, ésta se hace un byte más grande. Aunque no es un cambio considerable para la longitud de la instrucción, si afecta el tiempo de ejecución de la misma. Por lo general es costumbre limitar el uso del prefijo de sustitución de segmento y permanecer en los segmentos predeterminados, para poder escribir software más corto y eficiente.

4-7**DETALLES ACERCA DEL ENSAMBLADOR**

El ensamblador (MASM)¹ para el microprocesador puede utilizarse de dos formas: (1) con modelos que sean únicos para un ensamblador específico, y (2) con definiciones de segmento completo que permitan un completo control sobre el proceso de ensamblaje y que sean universales para todos los ensambladores. En esta sección del libro presentaremos ambos métodos y explicaremos cómo organizar el espacio de memoria de un programa mediante el uso del ensamblador. También explicaremos el propósito y el uso de algunas de las directivas más importantes que se utilizan con este ensamblador. En el apéndice A se proporcionan más detalles sobre el ensamblador.

En la mayoría de los casos se utiliza el ensamblador en línea incluido en Visual C++ para desarrollar código ensamblador y utilizarlo en un programa en C++, pero hay ocasiones en las que se requiere escribir módulos de ensamblador separados mediante el uso del programa ensamblador. En esta sección del libro compararemos siempre que sea posible el ensamblador en línea con el ensamblador.

Directivas

Antes de hablar sobre el formato de un programa en lenguaje ensamblador, debe aprender algunos detalles sobre las directivas (**seudoperaciones**) que controlan el proceso ensamblador. En la tabla 4-21 aparecen algunas directivas de lenguaje ensamblador. Las **directivas** indican la manera en que el ensamblador debe procesar un operando o una sección de un programa. Algunas directivas generan y almacenan información en la memoria; otras no. La directiva DB (**define byte**) almacena bytes de datos en la memoria, mientras que la directiva BYTE PTR nunca almacena datos. La directiva **BYTE PTR** indica el tamaño de los datos a los que hace referencia un apuntador o registro de índice. Tenga en cuenta que ninguna de las directivas funciona en el programa ensamblador en línea que forma parte de Visual C++. Si sólo utiliza el ensamblador en línea, puede omitir esta parte del texto. Tenga en cuenta que todavía se escriben secciones completas de código ensamblador mediante el uso de MASM.

El ensamblador acepta sólo instrucciones del 8086/8088 de manera predeterminada, a menos que antes del programa se coloque la directiva .686 o .686P, o uno de los otros modificadores de selección de microprocesador. La directiva .686 indica al ensamblador que debe utilizar el conjunto de instrucciones del Pentium Pro en modo real, y la directiva .686P indica al ensamblador que debe utilizar el conjunto de instrucciones del Pentium Pro en modo protegido. La mayoría del software moderno se escribe con la suposición de que el microprocesador es un Pentium Pro o un modelo más reciente, por lo que el modificador .686 se utiliza a menudo. Windows 95 fue el primer sistema operativo importante en utilizar una arquitectura de 32 bits que se conforma al 80386. Windows XP requiere un equipo clase Pentium (modificador .586) que utilice por lo menos un microprocesador de 233Mhz.

Almacenamiento de datos en un segmento de memoria. Las directivas DB (**define byte**), DW (**define palabra**) y DD (**define doble palabra**) que presentamos por primera vez en el capítulo 1 se utilizan con mucha frecuencia en MASM para definir y almacenar datos en la memoria. Si un coprocesador numérico ejecuta software en el sistema, también es común utilizar las directivas DQ (**define palabra cuádruple**) y DT (**define diez bytes**). Estas directivas etiquetan una posición de memoria con un nombre simbólico e indican su tamaño.

El ejemplo 4-13 muestra un segmento de memoria que contiene varias formas de directivas de definición de datos. También muestra la definición de segmento completo mediante la primera instrucción SEGMENT para indicar el inicio del segmento y su nombre simbólico. Como otra alternativa, al igual

¹ El ensamblador que utilizamos en este libro es el Microsoft MACRO assembler llamado MASM, versión 6.1.x.

TABLA 4-21 Directivas comunes de MASM.

<i>Directiva</i>	<i>Función</i>
.286	Selecciona el conjunto de instrucciones del 80286.
.286P	Selecciona el conjunto de instrucciones del 80286 en modo protegido.
.386	Selecciona el conjunto de instrucciones del 80386.
.386P	Selecciona el conjunto de instrucciones del 80386 en modo protegido.
.486	Selecciona el conjunto de instrucciones del 80486.
.486P	Selecciona el conjunto de instrucciones del 80486 en modo protegido.
.586	Selecciona el conjunto de instrucciones del Pentium.
.586P	Selecciona el conjunto de instrucciones del Pentium en modo protegido.
.686	Selecciona el conjunto de instrucciones del Pentium Pro-Pentium 4.
.686P	Selecciona el conjunto de instrucciones del Pentium Pro-Pentium 4 en modo protegido.
.287	Selecciona el coprocesador matemático 80287.
.387	Selecciona el coprocesador matemático 80387.
.CODE	Indica el inicio del segmento de código (modelos solamente).
.DATA	Indica el inicio del segmento de datos (modelos solamente).
.EXIT	Sale al DOS (modelos solamente).
.MODEL	Selecciona el modelo de programación.
.STACK	Selecciona el inicio del segmento de pila (modelos solamente).
.STARTUP	Indica la instrucción inicial en un programa (modelos solamente).
ALIGN n	Se alinea al límite n (n = 2 para palabras, n = 4 para dobles palabras).
ASSUME	Informa al ensamblador que debe nombrar cada segmento (segmentos completos solamente).
BYTE	Indica el tamaño de byte, como en BYTE PTR.
DB	Define byte(s) (8 bits).
DD	Define doble(s) palabra(s) (32 bits).
DQ	Define palabra(s) cuádruple(s) (64 bits).
DT	Define diez bytes (80 bits).
DUP	Genera duplicados.
DW	Define palabra(s) (16 bits).
DWORD	Indica el tamaño de doble palabra, como en DWORD PTR.
END	Termina un archivo de programa.
ENDM	Termina una MACRO secuencia.
ENDP	Termina un procedimiento.
ENDS	Termina un segmento o estructura de datos.
EQU	Iguala datos o una etiqueta con otra etiqueta.
FAR	Define un apuntador lejano, como en FAR PTR.
MACRO	Designa el inicio de una MACRO secuencia.
NEAR	Define un apuntador cercano, como en NEAR PTR.
OFFSET	Especifica una dirección de desplazamiento.
ORG	Establece el origen dentro de un segmento.
QWORD	Indica palabras octales, como en QWORD PTR.
PROC	Inicia un procedimiento.
PTR	Designa un apuntador.
SEGMENT	Inicia un segmento para segmentos completos.
STACK	Inicia un segmento de pila para segmentos completos.
STRUC	Define el inicio de una estructura de datos.
USES	Mete y saca registros automáticamente.
USE16	Usa el modo de instrucciones de 16 bits.
USE32	Usa el modo de instrucciones de 32 bits.
WORD	Indica el tamaño de palabra, como en WORD PTR.

que en ejemplos anteriores de este capítulo y de los capítulos anteriores, puede usarse el modelo SMALL con la instrucción .DATA. La última instrucción en este ejemplo contiene la directiva ENDS, que indica el final del segmento. El nombre del segmento (LISTA_SEG) puede ser cualquier nombre que desee el programador. Gracias a esto, un programa puede contener tantos segmentos como requiera.

EJEMPLO 4-13

```
;Uso de las directivas DB, DW y DD
;
0000          LISTA_SEG      SEGMENT
;
0000 01 02 03    DATOS1    DB  1,2,3           ;define bytes
0003 45          DB  45H            ;hexadecimal
0004 41          DB  'A'           ;ASCII
0005 F0          DB  11110000B      ;binario
0006 000C 000D    DATOS2    DW  12,13          ;define palabras
000A 0200          DW  LISTA1        ;símólico
000C 2345          DW  2345H         ;hexadecimal
000E 00000300    DATOS3    DD  300H           ;define doble palabra
0012 4007DF3B          DD  2.123          ;real
0016 544269E1          DD  3.34E+12      ;real
001A 00          LISTAA   DB  ?              ;reserva 1 byte
001B 000A[          LISTAB   DB  10 DUP(?)     ;reserva 10 bytes
    ???
]
0025 00          ALIGN   2             ;establece límite en palabra
0026 0100[          LISTAC   DW  100H DUP(0)    ;reserva 100H palabras
    0000
]
0226 0016[          LISTAD   DD  22 DUP(?)     ;reserva 22 dobles palabras
    ????????
]
027E 0064[          SEISES   DB  100 DUP(6)    ;reserva 100 bytes
    06
]
02E2          LISTA_SEG      ENDS
```

El ejemplo 4-13 muestra varias formas de almacenamiento de datos para los bytes en DATOS1. Se puede definir más de un byte en una línea en código binario, hexadecimal, decimal o ASCII. La etiqueta DATOS2 muestra cómo almacenar varias formas de datos tipo palabra. Las dobles palabras se almacenan en DATOS3; incluyen números reales de punto flotante, de precisión simple.

La memoria se **reserva** para usarla en el futuro mediante el uso de un signo de interrogación (?) como operando para la directiva DB, DW o DD. Cuando se utiliza un ? en lugar de un valor numérico o ASCII, el ensamblador separa una posición y no la inicializa con un valor específico. (En realidad, el ensamblador almacena por lo general un cero en las posiciones especificadas con un ?.) La directiva DUP (**duplicado**) crea un arreglo, como se muestra de varias formas en el ejemplo 4-12. Una instrucción 10 DUP (?) reserva 10 posiciones de memoria pero no almacena un valor específico en ninguna de las 10 posiciones. Si aparece un número dentro de la parte () de la instrucción DUP, el ensamblador inicializa la sección reservada de memoria con los datos indicados. Por ejemplo, la instrucción LISTA2 DB 10 DUP (2) reserva 10 bytes de memoria para el arreglo LISTA2 e inicializa cada posición con un 02H.

La directiva ALIGN que utilizamos en este ejemplo se asegura que los arreglos de memoria se almacenen en límites de tipo palabra. Una instrucción ALIGN 2 coloca los datos en *límites de tipo palabra* y una instrucción ALIGN 4 los coloca en *límites de tipo doble palabra*. En los microprocesadores del Pentium al Pentium 4, los datos de tipo palabra cuádruple para los números de punto flotante con precisión doble deben utilizar ALIGN 8. Es importante que los datos de tipo palabra se coloquen en límites de tipo palabra y que los datos de tipo doble palabra se coloquen en límites de tipo doble palabra. De no ser así, el microprocesador invierte tiempo adicional al acceder a estos tipos de datos. Una palabra almacenada en una posición de memoria con numeración impar requiere del doble de tiempo de acceso que una palabra almacenada en una posición de memoria con numeración par. Tenga en cuenta que la directiva ALIGN no puede usarse con modelos de memoria, ya que el tamaño del modelo determina la alineación de los datos. Si todos los datos de tipo doble palabra se definen primero, seguidos de los

datos de tipo palabra y después los datos de tipo byte, no se necesita la instrucción ALIGN para alinear los datos en forma correcta.

ASSUME, EQU y ORG. La directiva de igualación (EQU) iguala un valor numérico, ASCII o una etiqueta con otra etiqueta. Las igualaciones hacen que un programa sea más claro y simplifican su depuración. El ejemplo 4-14 muestra varias instrucciones de igualación y unas cuantas instrucciones que muestran su funcionamiento en un programa.

EJEMPLO 4-14

```
;Uso de la directiva de igualación
;
= 000A      DIEZ      EQU 10
= 0009      NUEVE     EQU 9
0000 B0 0A      MOV AL,DIEZ
0002 04 09      MOV AL,NUEVE
```

La directiva THIS siempre aparece como THIS BYTE, THIS WORD o THIS DWORD. En ciertos casos, se debe hacer referencia a los datos como un byte y como una palabra. El ensamblador sólo puede asignar una dirección de tipo byte, palabra o doble palabra a una etiqueta. Para asignar una etiqueta de tipo byte a una palabra, utilice el software listado en el ejemplo 4-15.

EJEMPLO 4-15

```
;Uso de las directivas THIS y ORG
;
0000          DATOS_SEG   SEGMENT
0300          ORG        300H
= 0300          DATOS1     EQU      THIS BYTE
0300          DATOS2     DW       ?
0302          DATOS_SEG  ENDS
0000          CODIGO_SEG SEGMENT 'CODE'
0000 8A 1E 0300 R    ASSUME  CS:CODIGO_SEG, DS:DATA_SEG
0004 A1 0300 R    MOV      BL,DATOS1
0007 8A 3E 0301 R    MOV      AX,DATOS2
                           MOV      BH,DATOS1+1
000B          CODIGO_SEG ENDS
```

Este ejemplo también muestra la manera en que la instrucción ORG (**origen**) cambia la dirección de desplazamiento inicial de los datos en el segmento de datos a la posición 300H. A veces el origen de los datos o del código debe asignarse a una dirección de desplazamiento absoluto mediante la instrucción ORG. La instrucción ASSUME indica al ensamblador qué nombres se han seleccionado para los segmentos de código, de datos, extra y de pila. Sin esta instrucción, el ensamblador no supone nada y utiliza de manera automática un prefijo de sustitución de segmento en todas las instrucciones que dirigen datos de la memoria. La instrucción ASSUME sólo se utiliza con definiciones de segmento completo, como veremos más adelante en este libro.

PROC y ENDP. Estas directivas indican el inicio y el final de un procedimiento (**subrutina**). Estas directivas *fuerzan la estructura* ya que el procedimiento se define claramente. Si la estructura debe violarse por alguna razón, hay que utilizar las instrucciones CALLF, CALLN, RETF y RETN. Ambas directivas requieren una etiqueta para indicar el nombre del procedimiento. La directiva PROC, que indica el inicio de un procedimiento, también debe ir seguida de NEAR o FAR. Un procedimiento NEAR reside en el mismo segmento de código que el programa. Un procedimiento FAR puede residir en cualquier ubicación en el sistema de memoria. A menudo, la llamada a un procedimiento NEAR se considera como *local* y la llamada a un procedimiento FAR se considera como *global*. Este término denota un procedimiento que puede utilizarse en cualquier programa; *local* define a un procedimiento

que sólo puede utilizarse en el programa actual. Cualquier etiqueta que se defina dentro del bloque del programa también se define como local (NEAR) o global (FAR).

El ejemplo 4-16 muestra un procedimiento que suma BX, CX y DX, y almacena la suma en el registro AX. Aunque este procedimiento es corto y no puede ser útil en lo particular, sí muestra cómo utilizar las directivas PROC y ENDP para delinear el procedimiento. La información sobre la operación del procedimiento debe aparecer como una agrupación de comentarios que muestren los registros que cambia el procedimiento y el resultado del mismo.

EJEMPLO 4-16

```
;Un procedimiento que suma BX, CX y DX, y almacena
;la suma en AX
;
0000          ADDEM  PROC   FAR           ; inicio del
                                         ; procedimiento
0000  03  D9          ADD     BX, CX
0002  03  DA          ADD     BX, DX
0004  8B  C3          MOV     AX, BX
0006  CB              RET
0007          ADDEM  ENDP           ; fin del procedimiento
```

Si está disponible la versión 6.x del programa ensamblador Microsoft MASM, la directiva PROC especifica y guarda en forma automática cualquier registro que se utilice dentro del procedimiento. La instrucción USES indica qué registros utiliza el procedimiento, de manera que el ensamblador pueda guardarlos en forma automática antes de que empiece el procedimiento, y restaurarlos antes de que termine el procedimiento mediante la instrucción RET. Por ejemplo, la instrucción ADDS PROC USES AX BX CX mete los registros AX, BX y CX de forma automática en la pila antes de que empiece el procedimiento, y los saca de la pila antes de que se ejecute la instrucción RET al final del procedimiento. El ejemplo 4-17 muestra un procedimiento escrito con MASM versión 6.x, que muestra la instrucción USES. Observe que los registros en la lista no van separados por comas sino por espacios, y que las instrucciones PUSH y POP se muestran en el listado del procedimiento porque se ensambló con la directiva .LIST ALL. El ensamblador insertó las instrucciones que tienen un asterisco (*) como prefijo; no se escribieron en el archivo fuente. La instrucción USES aparece en otras partes de este libro, por lo que si utiliza la versión 5.10 de MASM, tendrá que modificar el código.

EJEMPLO 4-17

```
;Un procedimiento que incluye la directiva USES para
;guardar BX, CX y DX en la pila y los restaura
;antes de la instrucción de retorno.
;
0000          ADDS   PROC   NEAR   USES BX CX DX
0000  53      *      push   bx
0001  51      *      push   cx
0002  52      *      push   dx
0003  03  D8          ADD     BX, AX
0005  03  CB          ADD     CX, BX
0007  03  D1          ADD     DX, CX
0009  8B  C2          MOV     AX, DX
                                         RET
000B  5A      *      pop    dx
000C  59      *      pop    cx
000D  5B      *      pop    bx
000E  C3      *      ret    0000h
000F          ADDS   ENDP
```

Organización de la memoria

El ensamblador utiliza dos formatos básicos para desarrollar software: un método utiliza modelos y el otro usa definiciones de segmento completo. Los modelos de memoria que presentaremos en esta sección, y que vimos brevemente en capítulos anteriores, son únicos para el programa ensamblador MASM. El ensamblador TASM también utiliza modelos de memoria, pero son algo distintos a los modelos del MASM. Las definiciones de segmento completo son comunes para la mayoría de los ensambladores, incluyendo al ensamblador Intel, y se utilizan con frecuencia para el desarrollo de software. Los modelos son más fáciles de utilizar para tareas sencillas. Las definiciones de segmento completo ofrecen un mejor control sobre la tarea en lenguaje ensamblador y se recomiendan para programas complejos. El modelo se utilizó en capítulos anteriores ya que es más fácil de entender para el programador principiante. Los modelos también se utilizan con procedimientos en lenguaje ensamblador utilizados en lenguajes de alto nivel como C/C++. Aunque en este libro desarrollamos y utilizamos completamente las definiciones del modelo de memoria para sus ejemplos de programación, hay que tener en cuenta que las definiciones de segmento completo ofrecen algunas ventajas sobre los modelos de memoria, como veremos más adelante en esta sección.

Modelos. Hay muchos modelos disponibles para el ensamblador MASM, que varían en tamaño desde diminuto (tiny) hasta enorme (huge). El apéndice A contiene una tabla que lista todos los modelos disponibles para usarse con el ensamblador. Para designar un modelo, se utiliza la instrucción .MODEL seguida del tamaño del sistema de memoria. El **modelo TINY** requiere que todo el software y los datos quepan en un segmento de memoria de 64 Kbytes; es útil para muchos programas pequeños. El **modelo SMALL** requiere que se utilice sólo un segmento de datos con un segmento de código para un total de 128 Kbytes de memoria. Hay otros modelos disponibles, hasta el modelo **HUGE**.

El ejemplo 4-18 ilustra la manera en que la instrucción .MODEL define los parámetros de un programa corto que copia el contenido de un bloque de memoria de 100 bytes (LISTAA) en un segundo bloque de memoria de 100 bytes (LISTAB). También muestra cómo definir los segmentos de pila, de datos y de código. La directiva .EXIT 0 regresa al DOS con un código de error de 0 (no hay error). Si no se agrega un parámetro a .EXIT, de todas formas regresa al DOS pero no se define el código de error. Además se utilizan las directivas especiales tales como @DATOS (vea el apéndice A) para identificar varios segmentos. Si se utiliza la directiva .STARTUP (MASM versión 6.x) puede eliminarse la instrucción MOV AX,@DATOS seguida de MOV DS,AX. La directiva .STARTUP también elimina la necesidad de almacenar la dirección inicial que está enseguida de la etiqueta END. Los modelos son importantes en los sistemas de desarrollo Microsoft Visual C++ y Borland C++ si se incluye lenguaje ensamblador con los programas en C++. Ambos sistemas de desarrollo utilizan programación de ensamblador en línea para agregar instrucciones de lenguaje ensamblador y requieren de una comprensión de los modelos de programación.

EJEMPLO 4-18

```

        .MODEL SMALL          ;selecciona el modelo pequeño
        .STACK 100H           ;define la pila
        .DATA                 ;inicio del segmento de datos

0000  0064[             LISTAA DB      100 DUP (?)
                      ??]
0064  0064[             LISTAB DB      100 DUP (?)
                      ??]
                      ]
                      .CODE                  ;inicio del segmento de código

0000  B9  ----  ?  AQUI:  MOV    AX, @DATOS   ;carga ES y DS
0003  8E  C0            MOV    ES, AX
0005  8E  D8            MOV    DS, AX
0007  FC                CLD
0008  BE  0000  R        MOV    SI, OFFSET LISTAA
000B  BF  0064  R        MOV    DI, OFFSET LISTAB

```

```

000E  B9  0064      MOV      CX, 100
0011  F3  / A4      REP      MOVSB

0013          .EXIT   0           ;salida al DOS
END   Aquí

```

Definiciones de segmento completo. El ejemplo 4-19 muestra el mismo programa mediante el uso de definiciones de segmento completo. Estas definiciones también se utilizan con los entornos Borland y Microsoft C/C++ para desarrollar procedimientos en lenguaje ensamblador. El programa en el ejemplo 4-19 parece ser más grande que el del ejemplo 4-18, pero está más estructurado que el método de los modelos para establecer un programa. El primer segmento definido es PILA SEG, que se delinea en forma muy clara mediante las directivas SEGMENT y ENDS. Dentro de estas directivas, la instrucción DW 100 DUP(?) separa 100H palabras para el segmento de pila. Como la palabra STACK aparece enseguida de SEGMENT, el ensamblador y el enlazador cargan de forma automática tanto el registro de segmento de pila (SS) como el apuntador de la pila (SP).

EJEMPLO 4-19

```

0000          PILA_SEG     SEGMENT      'STACK'
0000 0064[    DW          100H DUP(?)
    ?????
]
0200          PILA_SEG     ENDS

0000          DATOS_SEG    SEGMENT      'DATA'
0000 0064[    LISTAA DB    100 DUP(?)
    ???
]
0064 0064[    LISTAB DB    100 DUP(?)
    ???
]
00CB          DATOS_SEG    ENDS

0000          CODIGO_SEG   SEGMENT      'CODE'
                  ASSUME CS:CODE_SEG, DS:DATA_SEG
                  ASSUME SS:STACK_SEG
0000          MAIN         PROC        FAR
0000 B8 ---- R  MOV AX, DATA_SEG      ;carga DS y ES
0003 8E C0      MOV ES, AX
0005 8E D8      MOV DS, AX
0007 FC          CLD
                  SI, OFFSET LISTA
0008 BE 0000 R  MOV DI, OFFSET LISTB
000B BF 0064 R  MOV CX, 100
000E B9 0064    REP MOVSB
0011 F3/A4
0013 B4 4C      MOV AH, 4CH      ;salida al DOS
0015 CD 21      INT 21H
0017          MAIN         ENDP
0017          CODIGO_SEG  ENDS
END          MAIN

```

A continuación se definen los datos en el segmento DATOS_SEG. Aquí aparecen dos arreglos de datos como LISTAA y LISTAB. Cada arreglo contiene 100 bytes de espacio para el programa. Los nombres de los segmentos en este programa pueden cambiarse a cualquier otro nombre. Siempre se debe incluir el nombre de grupo 'DATA' para que el programa Microsoft CodeView pueda utilizarse con efectividad para depurar este software en forma simbólica. CodeView forma parte del paquete de MASM que se utiliza para depurar software. Para utilizar CodeView, escriba CV seguido por el nombre del archivo en la línea de comandos del DOS; si desea usarlo desde el programa Programmer's Workbench, seleccione la opción Debug bajo el menú Run. Si no se coloca el nombre de grupo en un programa, CodeView puede usarse de todas formas para depurar un programa, pero éste no se depurará en forma simbólica. En el apéndice A se listan otros nombres de grupo tales como 'STACK', 'CODE', etcétera. Debe colocar por lo menos la palabra 'CODE' enseguida de la instrucción SEGMENT para el segmento de código si desea ver el programa en forma simbólica con CodeView.

El segmento CODIGO_SEG se organiza como procedimiento lejano ya que la mayoría del software está orientado a procedimientos. Antes de que el programa empiece, el segmento de código contiene la instrucción ASSUME. Esta instrucción indica al ensamblador y al enlazador que el nombre usado para el segmento de código (CS) es CODIGO_SEG; también le indica al ensamblador y al enlazador que el segmento de datos es DATOS_SEG y que el segmento de pila es PILA_SEG. Observe que se utiliza el nombre de grupo 'CODE' para el segmento de código, de forma que lo pueda usar CodeView. En el apéndice A aparecen otros nombres de grupo con los modelos.

Una vez que el programa carga el registro de segmento extra y el registro de segmento de datos con la posición del segmento de datos, transfiere 100 bytes de LISTAA hacia LISTAB. Después de esto sigue una secuencia de dos instrucciones que devuelven el control al DOS (el sistema operativo en disco). El cargador del programa no inicializa DS y ES en forma automática. Estos registros deben cargarse con la dirección del segmento deseado en el programa.

La última instrucción en el programa es END MAIN. La instrucción END indica el final del programa y la posición de la primera instrucción ejecutada. Aquí se espera que la máquina ejecute el procedimiento principal, de manera que la etiqueta MAIN vaya después de la directiva END.

En los microprocesadores del 80386 al Pentium 4 hay una directiva adicional unida al segmento de código. La directiva USE16 o USE32 indica al ensamblador que debe utilizar ya sea el modo de instrucciones de 16 bits o el de 32 bits para el microprocesador. El software desarrollado para el entorno DOS debe utilizar la directiva USE16 para que el programa de los microprocesadores del 80386 al Pentium 4 pueda funcionar correctamente, ya que MASM supone que todos los segmentos son de 32 bits y todos los modos de instrucciones son de 32 bits de manera predeterminada.

Un programa de ejemplo

El ejemplo 4-20 proporciona un programa de ejemplo en el que se utilizan definiciones de segmento completo, que lee un carácter del teclado y lo muestra en la pantalla CRT. Aunque este programa es trivial, representa un programa funcional completo que funciona en cualquier computadora personal que utilice DOS, desde el primer sistema basado en el 8088 hasta el sistema más reciente basado en el Pentium 4. Este programa también muestra el uso de unas cuantas llamadas a funciones del DOS. (En el apéndice A se listan las llamadas a las funciones del DOS con sus parámetros.) Las llamadas a funciones del BIOS permiten el uso del teclado, la impresora, los discos duros y todo lo demás que esté disponible en su sistema computacional.

Este programa de ejemplo utiliza sólo un segmento de código ya que no hay datos. Debería aparecer un segmento de pila, pero se omite ya que DOS asigna en forma automática una pila de 128 bytes para todos los programas. La única vez que se utiliza la pila en este ejemplo es para las instrucciones INT 21H que llaman a un procedimiento en DOS. Observe que cuando este programa se enlaza, el enlazador indica que no existe un segmento de pila. Esta advertencia puede ignorarse en este ejemplo, ya que la pila es menor de 128 bytes.

Observe también que todo el programa se coloca en un procedimiento lejano llamado MAIN. Es una buena práctica de programación el escribir todo el software en forma de procedimientos, lo cual permitiría utilizar el programa como un procedimiento en el futuro, en caso de ser necesario. También es muy importante documentar el uso de los registros y de cualquier parámetro requerido para el programa en el encabezado del mismo, que consiste en una sección de comentarios que aparecen al inicio del programa.

El programa utiliza las funciones 06H y 4CH del DOS. El número de la función se coloca en AH antes de ejecutar la instrucción INT 21H. La función 06H lee el teclado si DL = 0FFH, o muestra el contenido ASCII de DL si no es 0FFH. Si examinamos el programa de cerca, la primera sección mueve el número 06H hacia AH y el número 0FFH hacia DL, de manera que pueda leerse una tecla desde el teclado. La instrucción INT 21H revisa el teclado; si no se escribe una tecla, devuelve el valor de igual. La instrucción JE evalúa la condición de igualdad y salta a MAIN si no se escribe una tecla.

Cuando se escribe una tecla, el programa continúa en el siguiente paso, en donde se compara el contenido de AL con un símbolo @. Al regresar de la instrucción INT 21H, el carácter ASCII de la tecla que se oprimió se encuentra en AL. Si se escribe un símbolo @ el programa termina. Si no se escribe este símbolo el programa continúa al mostrar el carácter escrito en el teclado mediante la siguiente instrucción INT 21H.

La segunda instrucción INT 21H mueve el carácter ASCII hacia DL para poder mostrarlo en la pantalla CRT. Después de mostrar el carácter se ejecuta una instrucción JMP. Esto hace que el programa continúe en MAIN, en donde repite el proceso de leer una tecla.

Si se escribe el símbolo @ el programa continúa en MAIN1, en donde ejecuta la función del DOS con el número de código 4CH. Esto hace que el programa devuelva el símbolo del sistema DOS, para poder utilizar la computadora para otras tareas.

En el apéndice A y en los siguientes capítulos aparece más información sobre el ensamblador y su aplicación. El apéndice A proporciona todas las generalidades sobre el ensamblador, el enlazador y las funciones del DOS. También proporciona una lista de las funciones del BIOS (sistema básico de E/S). La información que se proporcionará en los siguientes capítulos aclarará el uso del ensamblador para ciertas tareas en distintos niveles del libro.

EJEMPLO 4-20

```
;Un programa de ejemplo de segmento completo del DOS que lee
;una tecla y
;la muestra en pantalla. Una tecla @ termina el programa.
;
0000          CODIGO_SEG      SEGMENT 'CODE'
              ASSUME CS:CODIGO_SEG

0000          MAIN      PROC    FAR
;
0000  B4 06      MOV     AH, 06H      ;lee una tecla
0002  B2 FF      MOV     DL, OFFH
0004  CD 21      INT    21H
0006  74 F8      JE    MAIN        ;si no se escribe una tecla
0008  3C 40      CMP     AL, '@'
000A  74 08      JE    MAIN1       ;si se escribe una tecla @
000C  B4 06      MOV     AH, 06H      ;muestra la tecla (echo)
000E  8A D0      MOV     DL, AL
0010  CD 21      INT    21H
0012  EB EC      JMP    MAIN        ;repite
0014          MAIN1:      :
0014  B4 4C      MOV     AH, 4CH      ;salida al DOS
0016  CD 21      INT    21H
;
0018          MAIN      ENDP
0018          END    MAIN
```

EJEMPLO 4-21

```
;Un programa de ejemplo de modelo de DOS que lee una tecla y
;la muestra
;en pantalla. Una tecla @ termina el programa.
;
0000          .MODEL TINY
              .CODE
              .STARTUP

0100          MAIN:      :
0100  B4 06      MOV     AH, 6      ;lee una tecla
0102  B2 FF      MOV     DL, OFFH
0104  CD 21      INT    21H
0106  74 F8      JE    MAIN        ;si no se escribe una tecla
0108  3C 40      CMP     AL, '@'
010A  74 08      JE    MAIN1       ;si se escribe una tecla @
010C  B4 06      MOV     AH, 06H      ;muestra la tecla (echo)
010E  8A D0      MOV     DL, AL
0110  CD 21      INT    21H
0112  EB EC      JMP    MAIN        ;repite
0114          MAIN1:      :
;
0114          .EXIT      ;salida al DOS
END
```

El ejemplo 4-21 muestra el programa listado en el ejemplo 4-20, sólo que en el ejemplo 4-21 se utilizan modelos en vez de descripciones de segmento completo. Por favor compare los dos programas para determinar las diferencias. Observe cómo los modelos pueden hacer que un programa sea más corto y se vea más limpio.

4-8**RESUMEN**

1. Las instrucciones de movimiento de datos transfieren datos entre registros, entre un registro y la memoria, entre un registro y la pila, entre la memoria y la pila, entre el acumulador y los dispositivos de E/S, y entre las banderas y la pila. Las transferencias de memoria a memoria se permiten solamente con la instrucción MOVS.
2. Las instrucciones de movimiento de datos son MOV, PUSH, POP, XCHG, XLAT, IN, OUT, LEA, LOS, LES, LSS, LGS, LFS, LAHF, SAHF y las siguientes instrucciones de cadenas: LODS, STOS, MOVS, INS y OUTS.
3. El primer byte de una instrucción contiene el código de operación. Este código especifica la operación que va a realizar el microprocesador. En ciertas formas de instrucciones se pueden colocar uno o más prefijos de sustitución antes del código de operación.
4. El bit D, que se utiliza en muchas instrucciones, selecciona la dirección del flujo de los datos. Si D = 0, los datos fluyen desde el campo REG hacia el campo R/M de la instrucción. Si D = 1, los datos fluyen desde el campo R/M hacia el campo REG.
5. El bit W, que se incluye en muchas instrucciones, selecciona el tamaño de la transferencia de datos. Si W = 0, los datos son tipo byte; si W = 1, los datos son tipo palabra. En los microprocesadores 80386 y superiores, W = 1 especifica un registro tipo palabra o doble palabra.
6. MOD selecciona el modo de operación de direccionamiento para el campo R/M de una instrucción en lenguaje máquina. Si MOD = 00 no hay desplazamiento; si MOD = 01 aparece un desplazamiento de 8 bits con extensión de signo; si MOD = 10 ocurre un desplazamiento de 16 bits; y si MOD = 11 se utiliza un registro en vez de una posición de memoria. En los microprocesadores 80386 y superiores, los bits MOD también especifican un desplazamiento de 32 bits.
7. Un código de registro binario de 3 bits especifica los campos REG y R/M cuando MOD = 11. Los registros de 8 bits son AH, AL, BH, BL, CH, CL, DH y DL. Los registros de 16 bits son AX, BX, CX, DX, SP, BP, DI y SI. Los registros de 32 bits son EAX, EBX, ECX, EDX, ESP, EBP, EDI y ESI.
8. Cuando el campo R/M representa un modo de memoria, un código de 3 bits selecciona uno de los siguientes modos: [BX+DI], [BX+SI], [BP+DI], [BP+SI], [BX], [BP], [DI] o [SI] para las instrucciones de 16 bits. En los microprocesadores 80386 y superiores, el campo R/M especifica EAX, EBX, ECX, EDX, EBP, EDI y ESI o uno de los modos de índice escalado para direccionar los datos en la memoria. Si se selecciona el modo de índice escalado (R/M = 100) se agrega un byte adicional (byte de índice escalado) a la instrucción para especificar el registro base, el registro índice y el factor de escala.
9. Todos los modos de direccionamiento direccionan los datos en el segmento de datos de manera predeterminada, a menos que BP o EBP direccionen la memoria. Los registros BP o EBP direccionan datos en el segmento de pila.
10. Los registros de segmento se direccionan solamente mediante las instrucciones MOV, PUSH o POP. La instrucción MOV puede transferir un registro de segmento a un registro de 16 bits, o viceversa. Las instrucciones MOV CS,reg o POP CS no se permiten ya que sólo modifican una porción de la dirección. Los microprocesadores del 80386 al Pentium 4 incluyen dos registros de segmento adicionales: FS y GS.
11. Los datos se transfieren entre un registro o una posición de memoria y la pila mediante las instrucciones PUSH y POP. Las variaciones de estas instrucciones permiten meter datos inmediatos en la pila, transferir las banderas hacia o desde la pila y transferir todos los registros de 16 bits entre la pila y los registros. Cuando se transfieren datos hacia la pila, siempre se mueven dos

- bytes (8086-80286). El byte más significativo se coloca en la posición direccionada por SP – 1, y el byte menos significativo se coloca en la posición direccionada por SP – 2. Después de colocar los datos en la pila, SP se decremente en 2. En los microprocesadores del 80386 al Pentium 4, también pueden transferirse cuatro bytes de datos desde una posición de memoria o un registro hacia la pila.
- 12. Los códigos de operación que transfieren datos entre la pila y las banderas son PUSHF y POPF. Los códigos de operación que transfieren todos los registros de 16 bits entre la pila y los registros son PUSHA y POPA. En los microprocesadores 80386 y superiores, PUSHFD y POPFD transfieren el contenido del registro EFLAGS entre el microprocesador y la pila, y PUSHAD y POPAD transfieren todos los registros de 32 bits.
 - 13. Las instrucciones LEA, LDS y LES cargan un registro o registros con una dirección efectiva. La instrucción LEA carga cualquier registro de 16 bits con una dirección efectiva; LDS y LES cargan cualquier registro de 16 bits y ya sea DS o ES con la dirección efectiva. En los microprocesadores 80386 y superiores las instrucciones adicionales son LFS, LGS y LSS, las cuales cargan un registro de 16 bits y FS, GS o SS.
 - 14. Las instrucciones de transferencia de datos tipo cadena utilizan los registros DI, SI o ambos para direccionar memoria. La dirección de desplazamiento de DI se encuentra en el segmento extra y la dirección de desplazamiento de SI se encuentra en el segmento de datos. Si los microprocesadores del 80386 al Pentium 4 se operan en modo protegido, se utilizan EDI y ESI con las instrucciones de cadena.
 - 15. La bandera de dirección (D) selecciona el modo de operación de autoincremento o de autodecremento para DI y SI, para las instrucciones de cadena. Para hacer que D sea 0 se utiliza la instrucción CLD que selecciona el modo de autoincremento; para hacer que D sea 1 se utiliza la instrucción SLD que selecciona el modo de autodeCREMENTO. Uno de los registros DI o SI (o ambos) se incrementa/decrementa en 1 para una operación con bytes, en 2 para una operación con palabras y en 4 para una operación con dobles palabras.
 - 16. LODS carga AL, AX o EAX con datos desde la posición de memoria direccionada por SI; STOS almacena AL, AX o EAX en la posición de memoria direccionada por DI; y MOVS transfiere un byte, una palabra o una doble palabra desde la posición de memoria direccionada por SI hacia la posición direccionada por DI.
 - 17. INS introduce datos desde un dispositivo de E/S direccionado por DX y los almacena en la posición de memoria direccionada por DI. La instrucción OUTS envía el contenido de la posición de memoria direccionada por SI hacia el dispositivo de E/S direccionado por DX.
 - 18. El prefijo REP puede adjuntarse a cualquier instrucción de cadena para repetirla. El prefijo REP repite la instrucción de cadena el número de veces que se encuentra en el registro CX.
 - 19. En el lenguaje ensamblador pueden usarse operadores aritméticos y lógicos. Un ejemplo es la instrucción MOV AX,34*3, que carga AX con el número 102.
 - 20. XLAT (traduce) convierte los datos en AL en un número que se almacena en la posición de memoria direccionada por BX más AL.
 - 21. IN y OUT transfieren datos entre AL, AX o EAX y un dispositivo de E/S externo. La dirección del dispositivo de E/S se almacena ya sea con la instrucción (direcciónamiento de puerto fijo) o en el registro DX (direcciónamiento de puerto variable).
 - 22. Los microprocesadores del Pentium Pro al Pentium 4 contienen una nueva instrucción llamada CMOV, o movimiento condicional. Esta instrucción sólo realiza el movimiento si se cumple la condición.
 - 23. El prefijo de sustitución de segmento selecciona un registro de segmento distinto para una posición de memoria que sea distinto del segmento predeterminado. Por ejemplo, la instrucción MOV AX,[BX] utiliza el segmento de datos, pero la instrucción MOV AX,ES:[BX] utiliza el segmento extra debido al prefijo de sustitución ES. El uso del prefijo de sustitución de segmento es la única forma de direccionar los segmentos FS y GS en los microprocesadores del 80386 al Pentium 4.
 - 24. Las instrucciones MOVZX (movimiento y extensión con ceros) y MOVSX (movimiento y extensión con signo), incluidas en los microprocesadores 80386 y superiores, incrementan el tamaño de un byte para convertirlo en una palabra, o una palabra en doble palabra. La versión de extensión con ceros incrementa el tamaño del número mediante la inserción de ceros a la izquierda. La versión de extensión con signo incrementa el tamaño del número mediante la copia del bit de signo en los bits más significativos del número.

25. Las directivas de ensamblador DB (define byte), DW (define palabra), DD (define doble palabra) y DUP (duplicado) almacenan datos en el sistema de memoria.
26. La directiva EQU (igual) permite igualar datos o etiquetas con etiquetas.
27. La directiva SEGMENT identifica el inicio de un segmento de memoria y ENDS identifica el final de un segmento cuando se utilizan definiciones de segmento completo.
28. La directiva ASSUME indica al ensamblador qué nombres de segmento se han asignado a CS, DS, ES y SS cuando se utilizan definiciones de segmento completo. En los microprocesadores 80386 y superiores, ASSUME también indica el nombre de segmento para FS y GS.
29. Las directivas PROC y ENDP indican el inicio y el final de un procedimiento. La directiva USES (MASM versión 6.x) guarda y restaura de forma automática cualquier número de registros en la pila, si aparecen con la directiva PROC.
30. El ensamblador supone que el software se desarrolla para el microprocesador 8086/8088 a menos que se utilice la directiva .286, .386, .486, .586 o .686 para seleccionar uno de estos microprocesadores. Esta directiva se coloca después de la instrucción .MODEL si se va a utilizar el modo de instrucciones de 16 bits, y se coloca antes de esa instrucción si se va a utilizar el modo de instrucciones de 32 bits.
31. Los modelos de memoria pueden utilizarse para reducir ligeramente la longitud del programa, pero pueden ocasionar problemas en programas más grandes. Los modelos de memoria no son compatibles con todos los programas ensambladores.

4-9**PREGUNTAS Y PROBLEMAS**

1. El primer byte de una instrucción es el _____, a menos que contenga uno de los prefijos de sustitución.
2. Describa el propósito de los bits D y W que se encuentran en algunas instrucciones en lenguaje máquina.
3. ¿Qué información se especifica mediante el campo MOD en una instrucción en lenguaje máquina?
4. Si el campo de registro (REG) de una instrucción contiene 010 y W = 0, ¿qué registro se selecciona, suponiendo que la instrucción está en modo de 16 bits?
5. ¿Cómo se seleccionan los registros de 32 bits en el microprocesador 80486?
6. ¿Qué modo de direccionamiento de memoria se especifica cuando R/M = 001, con MOD = 00 para una instrucción de 16 bits?
7. Identifique los registros de segmento predeterminados que se asignan a los siguientes:
 - (a) SP
 - (b) EBX
 - (c) DI
 - (d) EBP
 - (e) SI
8. Convierta el número 8B07H de lenguaje máquina a lenguaje ensamblador.
9. Convierta el número 8B9E004CH de lenguaje máquina a lenguaje ensamblador.
10. Si aparece una instrucción MOV SI,[BX+2] en un programa, ¿cuál es su equivalente en lenguaje máquina?
11. Si aparece una instrucción MOV ESI,[EAX] en un programa para un microprocesador Pentium 4 que opere en el modo de instrucciones de 16 bits, ¿cuál es su equivalente en lenguaje máquina?
12. ¿Cuál es el error en la instrucción MOV CS,AX?
13. Forme una secuencia corta de instrucciones que carguen el registro del segmento de datos con el número 1000H.
14. En los microprocesadores del 8086 al 80286, las instrucciones PUSH y POP siempre transfieren un número de _____ bits entre la pila y un registro o posición de memoria.
15. ¿Qué registro de segmento no puede sacarse de la pila?
16. ¿Qué registros se mueven a la pila mediante la instrucción PUSHA?
17. ¿Qué registros se mueven a la pila mediante la instrucción PUSHAD?

18. Describa la operación de cada una de las siguientes instrucciones:
 - (a) PUSH AX
 - (b) POP ESI
 - (c) PUSH [BX]
 - (d) PUSHFD
 - (e) POP DS
 - (f) PUSHD 4
19. Explique lo que ocurre cuando se ejecuta la instrucción PUSH BX. Asegúrese de mostrar en dónde se almacenan BH y BL. (Suponga que SP = 0100H y SS = 0200H.)
20. Repita la pregunta 19 para la instrucción PUSH EAX.
21. La instrucción POP de 16 bits (excepto POPA) incrementa SP en _____.
22. ¿Qué valores aparecen en SP y SS si la pila se direcciona en la posición de memoria 02200H?
23. Compare la operación de la instrucción MOV DI,NUMERO con la instrucción LEA DI,NUMERO.
24. ¿Cuál es la diferencia entre la instrucción LEA SI,NUMERO y la instrucción MOV SI,OFFSET NUMERO?
25. ¿Qué es más eficiente, una instrucción MOV con un OFFSET o una instrucción LEA?
26. Describa cómo opera la instrucción LDS BX,NUMERO.
27. ¿Cuál es la diferencia entre las instrucciones LDS y LSS?
28. Desarrolle una secuencia de instrucciones que muevan el contenido de las posiciones de memoria NUMERO y NUMERO + 1 del segmento de datos hacia BX, DX y SI.
29. ¿Cuál es el propósito de la bandera de dirección?
30. ¿Qué instrucciones establecen y borran la bandera de dirección?
31. ¿Qué instrucción(es) de cadena utilizan a DI y SI para direccionar datos en la memoria?
32. Explique la operación de la instrucción LODSB.
33. Explique la operación de la instrucción STOSW.
34. Explique la operación de la instrucción OUTSB.
35. ¿Qué logra el prefijo REP, y qué tipo de instrucción se utiliza con él?
36. Desarrolle una secuencia de instrucciones que copien 12 bytes de datos de un área de memoria direccionada por ORIGEN, hacia un área de memoria direccionada por DESTINO.
37. ¿En dónde se almacena la dirección de E/S (número de puerto) para una instrucción INSB?
38. Seleccione una instrucción en lenguaje máquina que intercambie el contenido del registro EBX con el registro ESI.
39. ¿Es normal que aparezcan las instrucciones LAHF y SAHF en el software?
40. Explique cómo la instrucción XLAT transforma el contenido del registro AL.
41. Escriba un programa corto que utilice la instrucción XLAT para convertir los números BCD 0-9 en los números 30H-39H codificados en ASCII. Almacene los datos codificados en ASCII en una TABLA ubicada dentro del segmento de datos.
42. Explique qué hace la instrucción IN AL,12H.
43. Explique cómo opera la instrucción OUT DX,AX.
44. ¿Qué es un prefijo de sustitución de segmento?
45. Seleccione una instrucción que mueva un byte de datos desde la posición de memoria direccionada por el registro BX hacia el segmento extra en el registro AH.
46. Desarrolle una secuencia de instrucciones que intercambien el contenido de AX con BX, el de ECX con EDX, y el de SI con DI.
47. ¿Qué hace la instrucción CMOVNE CX,DX en el microprocesador Pentium 4?
48. ¿Qué es una directiva de lenguaje ensamblador?
49. Describa el propósito de las siguientes directivas de lenguaje ensamblador: DB, DW y DD.
50. Seleccione una directiva de lenguaje ensamblador que reserve 30 bytes de memoria para el arreglo LISTA1.
51. Describa el propósito de la directiva EQU.
52. ¿Cuál es el propósito de la directiva .686?
53. ¿Cuál es el propósito de la directiva .MODEL?
54. Si el inicio de un segmento se identifica con .DATA, ¿qué tipo de organización de memoria está en uso?

55. Si la directiva SEGMENT identifica el inicio de un segmento, ¿qué tipo de organización de memoria está en uso?
56. ¿Qué hace la directiva INT 21H si AH contiene 4CH?
57. ¿Qué directivas indican el inicio y el final de un procedimiento?
58. Explique el propósito de la instrucción USES cuando se aplica a un procedimiento con la versión 6.x de MASM.
59. ¿Cómo se indica al microprocesador Pentium 4 que utilice el modo de instrucciones de 16 bits?
60. Desarrolle un procedimiento cercano que almacene el registro AL en cuatro posiciones de memoria consecutivas dentro del segmento de datos, cuando se direcciona mediante el registro DI.
61. Desarrolle un procedimiento lejano que copie el contenido de la posición de memoria tipo palabra CS:DATOS4 en AX, BX, CX, DS y SI.

CAPÍTULO 5

Instrucciones aritméticas y lógicas

INTRODUCCIÓN

En este capítulo se examinarán las instrucciones aritméticas y lógicas. Las instrucciones aritméticas son: suma, resta, multiplicación, división, comparación, negación, incremento y decremento. Las instrucciones lógicas son: AND, OR, OR exclusivo, NOT, desplazamientos, desplazamientos cílicos y la comparación lógica (TEST). En este capítulo también se verán las instrucciones XADD, SHRD, SHLD, pruebas de bits y exploraciones de bits que se incluyen en los microprocesadores del 80386 al Pentium 4. Se concluirá el capítulo con una discusión sobre las instrucciones de comparación de cadenas, las cuales se utilizan para explorar datos tabulares y para comparar secciones de datos en la memoria. Ambas tareas se realizan con eficiencia mediante las instrucciones exploración de cadenas (SCAS) y comparación de cadenas (CMPS).

Si está familiarizado con un microprocesador de 8 bits, reconocerá que el conjunto de instrucciones del 8086 al Pentium 4 es superior a la mayoría de los microprocesadores de 8 bits, ya que gran parte de las instrucciones tienen dos operandos en vez de uno. Aún si usa por primera vez este microprocesador, aprenderá rápidamente que posee un conjunto de instrucciones aritméticas y lógicas poderosas, fácil de usar.

OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

1. Utilizar las instrucciones aritméticas y lógicas para realizar operaciones aritméticas simples con números binarios, BCD y caracteres ASCII.
2. Utilizar las instrucciones AND, OR y OR exclusivo para realizar manipulaciones de bits binarios.
3. Utilizar las instrucciones de desplazamiento y de desplazamiento cílico.
4. Explicar la operación de las instrucciones de intercambio y suma, comparación e intercambio, desplazamiento de doble precisión, evaluación de bits y exploración de bits de los microprocesadores del 80386 al Pentium 4.
5. Revisar el contenido de una tabla para ver si concuerda con las instrucciones de cadenas.

El grupo de las instrucciones aritméticas que se encuentran en cualquier microprocesador está compuesto por la suma, la resta y la comparación. En esta sección se verán las instrucciones de suma, resta y comparación. Además se mostrarán su uso para manipular datos en los registros y en la memoria.

Suma

En el microprocesador, la suma (ADD) aparece de muchas formas. En esta sección se verán con detalle el uso de la instrucción ADD para la suma binaria de 8, 16 y 32 bits. Después presentaremos una segunda forma de hacer la suma, llamada **suma con acarreo**, mediante la instrucción ADC. Por último, presentaremos la instrucción de incremento (INC). Esta instrucción es un tipo especial de suma, en la que se suma un 1 al número. En la sección 5-3 examinaremos otras formas de hacer la suma, como BCD y ASCII. También describiremos la instrucción XADD, que se incluye en los microprocesadores del 80486 al Pentium 4.

La tabla 5-1 muestra los modos de direccionamiento disponibles para la instrucción ADD. (Entre estos modos de direccionamiento se incluyen casi todos los que mencionamos en el capítulo 3.) Pero como hay más de 32,000 variaciones de la instrucción ADD en el conjunto de instrucciones, es imposible listarlos todos en esta tabla. Los únicos tipos de suma que *no* se permiten son la suma de memoria a memoria y la de registros de segmento. Los registros de segmento sólo pueden moverse y meterse o sacarse de la pila. Al igual que con otras instrucciones, los registros de 32 bits sólo están disponibles con los microprocesadores del 80386 al Pentium 4.

TABLA 5-1 Ejemplos de instrucciones de suma.

Lenguaje ensamblador	Operación
ADD AL,BL	$AL = AL + BL$
ADD CX,DI	$CX = CX + DI$
ADD EBP,EAX	$EBP = EBP + EAX$
ADD CL,44H	$CL = CL + 44H$
ADD BX,245FH	$BX = BX + 245FH$
ADD EDX,12345H	$EDX = EDX + 12345H$
ADD [BX],AL	AL se suma al contenido tipo byte de la posición de memoria del segmento de datos direccionada por BX; la suma se almacena en la misma posición de memoria.
ADD CL,[BP]	El contenido tipo byte de la posición de memoria del segmento de pila direccionada por BP se suma a CL; la suma se almacena en CL.
ADD AL,[EBX]	El contenido tipo byte de la posición de memoria del segmento de datos direccionada por EBX se suma a AL; la suma se almacena en AL.
ADD BX,[SI+2]	El contenido tipo palabra de la posición de memoria del segmento de datos direccionada por SI + 2 se suma a BX; la suma se almacena en BX.
ADD CL,TEMP	El contenido tipo byte de la posición de memoria TEMP del segmento de datos se suma a CL; la suma se almacena en CL.
ADD BX,TEMP[DI]	El contenido tipo palabra de la posición de memoria del segmento de datos direccionada por TEMP + DI se suma a BX; la suma se almacena en BX.
ADD [BX+DI],DL	DL se suma al contenido tipo byte de la posición de memoria del segmento de datos direccionada por BX + DI; la suma se almacena en la misma posición de memoria.
ADD BYTE PTR [DI],3	Se suma un 3 al contenido tipo byte de la posición de memoria del segmento de datos direccionada por DI; la suma se almacena en la misma posición.
ADD BX,[EAX+2*ECX]	El contenido tipo palabra de la posición de memoria del segmento de datos direccionada por EAX más 2 por ECX se suma a BX; la suma se almacena en BX.

Suma de registros. El ejemplo 5-1 muestra una secuencia simple de instrucciones que utiliza la suma de registros para sumar el contenido de varios registros. En este ejemplo se suma el contenido de AX, BX, CX y DX para formar un resultado de 16 bits que se almacena en el registro AX.

EJEMPLO 5-1

```
0000 03 C3      ADD     AX, BX
0002 03 C1      ADD     AX, CX
0004 03 C2      ADD     AX, DX
```

Siempre que se ejecutan las instrucciones lógicas y aritméticas, se modifica el contenido del registro de banderas. El contenido de la bandera de interrupción, de atrapamiento y de otras más no cambia. Sólo se modifican las banderas ubicadas en los ocho bits de más a la derecha del registro de banderas y la bandera de desbordamiento. Estas banderas de más a la derecha indican el resultado de la operación aritmética o lógica. Cualquier instrucción ADD modifica el contenido de las banderas de signo, cero, acarreo, acarreo auxiliar, paridad y desbordamiento. Los bits de bandera nunca cambian en la mayoría de las instrucciones de transferencia de memoria que presentamos en el capítulo 4.

Suma inmediata. Esta suma se utiliza siempre que se suman datos constantes o conocidos. En el ejemplo 5-2 aparece una suma inmediata de 8 bits. En este ejemplo, DL se carga primero con 12H mediante una instrucción de movimiento inmediato. Despues se suma el 33H al 12H en DL mediante una instrucción de suma inmediata. Posteriormente, la suma (45H) se mueve hacia el registro DL y las banderas se modifican de la siguiente manera:

Z = 0 (resultado distinto de cero)
C = 0 (sin acarreo)
A = 0 (sin medio acarreo)
S = 0 (resultado positivo)
P = 0 (paridad par)
O = 0 (sin desbordamiento)

EJEMPLO 5-2

```
0000 B2 12      MOV     DL, 12H
0002 80 C2 33    ADD     DL, 33H
```

Suma de memoria a registro. Suponga que una aplicación requiere sumar datos de la memoria al registro AL. El ejemplo 5-3 muestra un programa que suma dos bytes consecutivos de datos, almacenados en las posiciones NUMERO y NUMERO+1 del segmento de datos, al registro AL.

EJEMPLO 5-3

```
0000 BF 0000 R      MOV     DI, OFFSET NUMERO ;direcciona NUMERO
0003 B0 00          MOV     AL, 0           ;borra la suma
0005 02 05          ADD     AL, [DI]        ;suma NUMERO
0007 02 45 01       ADD     AL, [DI + 1]   ;suma NUMERO+1
```

La primera instrucción carga el registro índice de destino (DI) con la dirección de desplazamiento NUMERO. El registro DI que se utiliza en este ejemplo direcciona datos en el segmento de datos que comienza en la posición de memoria NUMERO. Despues de poner la suma en ceros, la instrucción ADD AL,[DI] suma el contenido de la posición de memoria NUMERO a AL. Por último, la instrucción ADD AL,[DI+1] suma el contenido de la posición de memoria NUMERO más un byte al registro DL. Despues de que se ejecutan ambas instrucciones ADD, el resultado aparece en el registro AL como la suma del contenido de NUMERO más el contenido de NUMERO+1.

Suma de arreglos. Los arreglos en memoria son listas secuenciales de datos. Suponga que un arreglo de datos (ARREGLO) contiene 10 bytes, numerados desde el elemento 0 hasta el elemento 9. El ejemplo 5-4 muestra cómo sumar el contenido de los elementos 3, 5 y 7 del arreglo.

Este ejemplo primero pone AL en ceros, por lo que puede utilizarse para acumular la suma. A continuación el registro SI se carga con un 3 para direccionar de manera inicial el elemento 3 del arreglo. La instrucción ADD AL,ARREGLO[SI] suma el contenido del elemento 3 del arreglo a la suma en AL. Las instrucciones que siguen suman los elementos 5 y 7 del arreglo a la suma en AL; mediante el uso de un 3 en SI más un desplazamiento de 2 para direccionar el elemento 5, y un desplazamiento de 4 para direccionar el elemento 7.

EJEMPLO 5-4

0000 B0 00	MOV AL, 0	;borra la suma
0002 BE 0003	MOV SI, 3	;direcciona el elemento 3
0005 02 84 0000 R	ADD AL, ARREGLO[SI]	;suma el elemento 3
0009 02 84 0002 R	ADD AL, ARREGLO[SI+2]	;suma el elemento 5
000D 02 84 0004 R	ADD AL, ARREGLO[SI+4]	;suma el elemento 7

Suponga que un arreglo de datos contiene números de 16 bits que se utilizan para formar una suma de 16 bits en el registro AX. El ejemplo 5-5 muestra una secuencia de instrucciones escritas para los microprocesadores 80386 y superiores, en la que se muestra el formato de índice escalado de direccionamiento para sumar los elementos 3, 5 y 7 de un área de memoria llamada ARREGLO. En este ejemplo, EBX se carga con la dirección ARREGLO y ECX guarda el número de elemento del arreglo. Observe cómo se utiliza el factor de escala para multiplicar el contenido del registro ECX por 2 para direccionar palabras de datos. (Recuerde que las palabras son de dos bytes de longitud.)

EJEMPLO 5-5

0000 66 BB 00000000 R	MOV EBX,OFFSET ARREGLO	;direcciona ARREGLO
0006 66 B9 0000003	MOV ECX,3	;direcciona el elemento 3
000C 67&8B 04 4B	MOV AX,[EBX+2*ECX]	;obtiene el elemento 3
0010 66 B9 0000005	MOV ECX,5	;direcciona el elemento 5
0016 67&03 04 4B	ADD AX,[EBX+2*ECX]	;suma el elemento 5
001A 66 B0 0000007	MOV ECX,7	;direcciona el elemento 7
0020 67&03 04 4B	ADD AX,[EBX+2*ECX]	;suma el elemento 7

Suma de incremento. Esta operación (INC) suma 1 a un registro o a una posición de memoria. La instrucción INC suma 1 a cualquier registro o posición de memoria, excepto a un registro de segmento. La tabla 5-2 muestra algunas de las formas posibles de la instrucción de incremento disponibles en los microprocesadores del 8086 al Pentium 4. Al igual que con otras instrucciones que hemos presentado hasta ahora, es imposible mostrar todas las variaciones de la instrucción INC debido al extenso número disponible.

Con los incrementos de memoria indirectos debe describirse el tamaño de los datos mediante el uso de las directivas BYTE PTR, WORD PTR o DWORD PTR. La razón es que el programa ensamblador no

TABLA 5-2 Ejemplos de instrucciones de incremento.

Lenguaje ensamblador	Operación
INC BL	BL = BL + 1
INC SP	SP = SP + 1
INC EAX	EAX = EAX + 1
INC BYTE PTR[BX]	Suma 1 al contenido en bytes de la posición de memoria del segmento de datos direccionada por BX.
INC WORD PTR[SI]	Suma 1 al contenido tipo palabra de la posición de memoria del segmento de datos direccionada por SI.
INC DWORD PTR[ECX]	Suma 1 al contenido tipo doble palabra de la posición de memoria del segmento de datos direccionada por ECX.
INC DATOS1	Suma 1 al contenido de la posición de memoria DATOS1 del segmento de datos.

puede determinar si, por ejemplo, la instrucción INC [DI] es un incremento tipo byte, palabra o doble palabra. La instrucción INC BYTE PTR [DI] indica en forma clara que se utilizan datos en memoria tipo byte; La instrucción INC WORD PTR [DI] indica sin duda que se utilizan datos en memoria tipo palabra; y la instrucción INC DWORD PTR [DI] indica el uso de datos tipo doble palabra.

El ejemplo 5-6 muestra cómo modificar el ejemplo 5-3 para utilizar la instrucción de incremento para direccionar NUMERO y NUMERO+1. Aquí, una instrucción INC DI cambia el contenido del registro DI, de la dirección de desplazamiento NUMERO a la dirección de desplazamiento NUMERO+1. Ambas secuencias de programa que se muestran en los ejemplos 5-3 y 5-6 suman el contenido de NUMERO y de NUMERO+1. La diferencia entre ellas es la manera en que se forma la dirección a través del contenido del registro DI, mediante el uso de la instrucción de incremento.

EJEMPLO 5-6

0000 BF 0000 R	MOV DI,OFFSET NUMERO	; direcciona NUMERO
0003 B0 00	MOV AL,0	; borra la suma
0005 02 05	ADD AL,[DI]	; suma NUMERO
0007 47	INC DI	; incrementa DI
0008 02 05	ADD AL,[DI]	; suma NUMERO+1

Las instrucciones de incremento afectan a los bits de bandera, al igual que a la mayoría de las otras operaciones aritméticas y lógicas. La diferencia es que las instrucciones de incremento no afectan al bit de bandera de acarreo. El acarreo no cambia porque a menudo utilizamos incrementos en los programas que dependen del contenido de la bandera de acarreo. El incremento se utiliza solamente para apuntar al siguiente elemento de memoria en un arreglo tipo byte de datos. Si se direccionan datos tipo palabra, es mejor utilizar una instrucción ADD DI,2 para modificar el apuntador DI en vez de dos instrucciones INC DI. Para los arreglos tipo doble palabra se utiliza la instrucción ADD DI,4 para modificar el apuntador DI. En algunos casos hay que preservar la bandera de acarreo, lo que significa que podrían aparecer dos o cuatro instrucciones INC en un programa para modificar un apuntador.

Suma con acarreo. La instrucción de suma con acarreo (ADC) suma el bit en la bandera de acarreo (C) con los datos del operando. Esta instrucción aparece principalmente en el software que suma números mayores de 16 bits en los microprocesadores del 8086 al 80286, o mayores de 32 bits en los microprocesadores del 80386 al Pentium 4.

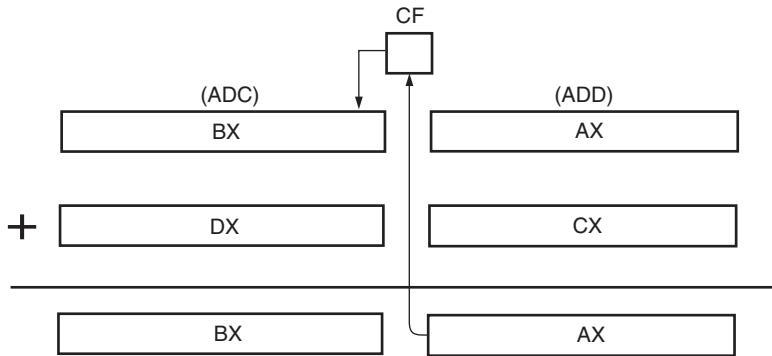
La tabla 5-3 muestra varias instrucciones de suma con acarreo, con comentarios que explican su operación. Al igual que la instrucción ADD; la instrucción ADC afecta a las banderas después de la suma.

Suponga que un programa está escrito para los microprocesadores del 8086 al 80286, el cual suma el número de 32 bits que está en BX y AX con el número de 32 bits en DX y CX. La figura 5-1 ilustra esta adición, de manera que pueda comprenderse la colocación y la función de la bandera de acarreo. Esta suma no puede realizarse fácilmente sin sumar el bit de bandera de acarreo, ya que los

TABLA 5-3 Ejemplos de instrucciones de suma con acarreo.

Lenguaje ensamblador	Operación
ADC AL,AH	AL = AL + AH + acarreo.
ADC CX,BX	CX = CX + BX + acarreo.
ADC EBX,EDX	EBX = EBX + EDX + acarreo.
ADC DH,[BH]	El contenido tipo byte de la posición de memoria del segmento de datos direccionada por BX se suma a DH; la suma se almacena en DH.
ADC BX,[BP+2]	El contenido tipo palabra de la posición de memoria del segmento de pila direccionada por BP más 2 se suma a BX; la suma se almacena en BX.
ADC ECX,[EBX]	El contenido tipo doble palabra de la posición de memoria del segmento de datos direccionada por EBX se suma a ECX; la suma se almacena en ECX.

FIGURA 5-1 Una suma con acarreo en la que se muestra cómo la bandera de acarreo (C) vincula las dos sumas de 16 bits en una suma de 32 bits.



microprocesadores del 8086 al 80286 sólo suman números de 8, 16 o 32 bits. El ejemplo 5-7 muestra cómo se suma el contenido de los registros AX y CX para formar los 16 bits menos significativos de la suma. Esta suma puede o no generar un acarreo. Para que aparezca un acarreo en la bandera de acarreo, la suma debe ser mayor que FFFFH. Como es imposible predecir un acarreo, los 16 bits más significativos de esta adición se suman con la bandera de acarreo mediante el uso de la instrucción ADC. Esta instrucción suma el 1 o el 0 en la bandera de acarreo a los 16 bits más significativos del resultado. Este programa suma BX-AX a DX-CX y la suma aparece en BX-AX.

EJEMPLO 5-7

0000 03 C1	ADD	AX, CX
0002 13 DA	ADC	BX, DX

Suponga que el mismo software se reescribe para los microprocesadores del 80386 al Pentium 4, con la modificación para sumar dos números de 64 bits. Los cambios requeridos para esta operación son el uso de los registros extendidos para guardar los datos y las modificaciones de las instrucciones para los microprocesadores 80386 y superiores. Estos cambios se muestran en el ejemplo 5-8, en el cual se suman dos números de 64 bits.

EJEMPLO 5-8

0000 66 03 C1	ADD	EAX, ECX
0003 66 13 DA	ADC	EBX, EDX

Intercambio y suma para los microprocesadores del 80486 al Pentium 4. En el conjunto de instrucciones del 80486 aparece un nuevo tipo de suma llamado **intercambio y suma** (XADD); esta instrucción se continua usando hasta el Pentium 4. La instrucción XADD suma el origen con el destino y almacena la suma en el destino, al igual que con cualquier suma. La diferencia es que después de realizar la suma, el valor original del destino se copia en el operando de origen. Ésta es una de las pocas instrucciones que cambian el origen.

Por ejemplo, si BL = 12H y DL = 02H, al ejecutar la instrucción XADD BL,DL el registro BL contiene la suma de 14H y DL se convierte en 12H. La suma de 14H se genera y el destino original de 12H sustituye el origen. Esta instrucción funciona con cualquier tamaño de registros y con cualquier operando de memoria, de igual forma que la instrucción ADD.

Resta

En el conjunto de instrucciones aparecen muchas formas de resta (SUB). Estas formas utilizan cualquier modo de direccionamiento con datos de 8, 16 o 32 bits. Una forma especial de substracción (decremento o DEC) resta un 1 a cualquier registro o posición de memoria. La sección 5-3 muestra cómo se restan los datos BCD y ASCII. Al igual que con la suma, en ciertas ocasiones hay que restar números mayores de 16 bits o de 32 bits. La instrucción **resta con acarreo** (SBB) realiza este tipo de resta. En los microprocesadores del 80486 al Pentium 4, el conjunto de instrucciones también incluye una instrucción de comparación y una de intercambio.

TABLA 5-4 Ejemplos de instrucciones de resta.

<i>Lenguaje ensamblador</i>	<i>Operación</i>
SUB CL,BL	CL = CL – BL
SUB AX,SP	AX = AX – SP
SUB ECX,EBP	ECX = ECX – EBP
SUB DH,6FH	DH = DH – 6FH
SUB AX,0CCCCH	AX = AX – 0CCCCH
SUB ESI,2000300H	ESI = ESI – 2000300H
SUB [DI],CH	Resta CH del contenido tipo byte de la posición de memoria del segmento de datos direccionada por DI y almacena la diferencia en la misma posición de memoria.
SUB CH,[BP]	Resta de CH el contenido tipo byte de la posición de memoria del segmento de datos direccionada por BP y almacena la diferencia en CH.
SUB AH,TEMP	Resta de AH el contenido tipo byte de la posición de memoria TEMP y almacena la diferencia en AH.
SUB DI,TEMP[ESI]	Resta de DI el contenido tipo palabra de la posición de memoria del segmento de datos direccionada por TEMP más ESI y almacena la diferencia en DI.
SUB ECX,DATOS1	Resta de ECX el contenido tipo doble palabra de la posición de memoria DATOS1 y almacena la diferencia en ECX.

La tabla 5-4 lista algunos de los diversos modos de direccionamiento que se permiten con la instrucción de resta (SUB). Existen más de 1000 instrucciones de resta posibles, que son demasiadas como para poder listarlas aquí. Dos de los tipos de resta que no se permiten son la resta de memoria a memoria y de registros de segmento. Al igual que las demás instrucciones aritméticas, la instrucción de resta afecta a los bits de bandera.

Resta de registros. El ejemplo 5-9 muestra una secuencia de instrucciones que realizan restas entre registros. En este ejemplo se resta el contenido de 16 bits de los registros CX y DX del contenido del registro BX. Después de cada resta, el microprocesador modifica el contenido del registro de banderas. Las banderas cambian para la mayoría de las operaciones aritméticas y lógicas.

EJEMPLO 5-9

```
0000 2B D9      SUB     BX, CX
0002 2B DA      SUB     BX, DX
```

Resta inmediata. Al igual que con la suma, el microprocesador también permite el uso de operandos inmediatos para la resta de datos constantes. El ejemplo 5-10 presenta una secuencia corta de instrucciones que restan el 44H del 22H. Aquí primero cargamos el 22H en CH mediante el uso de una instrucción de movimiento inmediato. A continuación, la instrucción SUB (mediante el uso del 44H inmediato) resta el 44H del 22H. Después de la resta, la diferencia (0DEH) se mueve al registro CH. En esta resta, las banderas cambian de la siguiente manera:

- Z = 0 (resultado distinto de cero)
- C = 1 (acarreo de resta)
- A = 1 (medio acarreo de resta)
- S = 1 (resultado negativo)
- P = 1 (paridad par)
- O = 0 (sin desbordamiento)

EJEMPLO 5-10

0000	B5	22	MOV	CH, 22H
0002	80	ED	SUB	CH, 44H

Ambas banderas de acarreo (C y A) guardan acarreos de resta después de una resta en vez de acarreos de suma, como después de una suma. En este ejemplo no hay desbordamiento. Se restó el 44H (+68) del 22H (+34) y quedó como resultado un 0DEH(-34). Como el resultado correcto de 8 bits con signo es -34, no hay desbordamiento en este ejemplo. Un desbordamiento de 8 bits se produce sólo si el resultado con signo es mayor de +127 o menor de -128.

Resta con decremento. La resta con decremento (DEC) resta un 1 de un registro o del contenido de una posición de memoria. La tabla 5-5 lista algunas instrucciones de decremento que ilustran el funcionamiento de los decrementos de registro y de memoria.

Las instrucciones de decremento de datos de memoria indirectos requieren las directivas BYTE PTR, WORD PTR o DWORD PTR, ya que el ensamblador no puede diferenciar un byte de una palabra o de una doble palabra cuando un registro tipo índice accede a la memoria. Por ejemplo, la instrucción DEC [SI] es imprecisa ya que el ensamblador no puede determinar si la posición direccionada por SI es un byte, una palabra o una doble palabra. El uso de DEC BYTE PTR [SI], DEC WORD PTR [SI] o DEC DWORD PTR [SI] revela al ensamblador el tamaño de los datos.

Resta con acarreo. Una instrucción de resta con acarreo (SBB) funciona como una resta común, sólo que la bandera de acarreo (C) (que guarda el acarreo de la resta) también se resta de la diferencia. El uso más común para esta instrucción es en las restas mayores de 16 bits en los microprocesadores del 8086 al 80286, o mayores de 32 bits en los microprocesadores del 80386 al Pentium 4. Las restas grandes requieren de la propagación de los acarreos a través de la resta, al igual que las sumas grandes propagan el acarreo.

La tabla 5-6 lista varias instrucciones SBB con comentarios que definen sus operaciones. Al igual que la instrucción SUB, SBB afecta a las banderas. Observe que la instrucción de resta inmediata de memoria en esta tabla requiere una directiva BYTE PTR, WORD PTR o DWORD PTR.

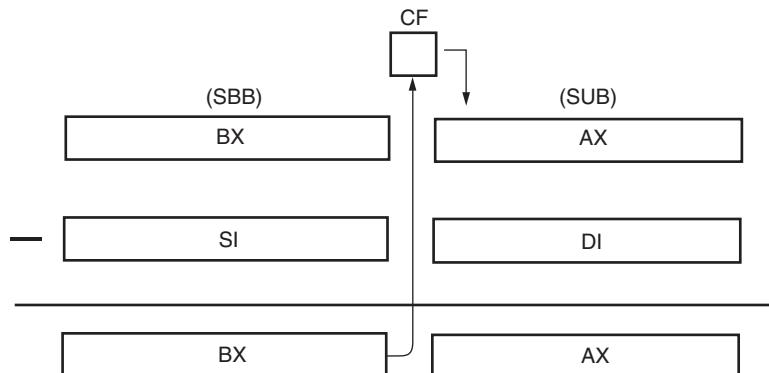
Cuando el número de 32 bits que se guarda en BX y AX se resta del número de 32 bits que se guarda en SI y DI, la bandera de acarreo propaga el acarreo de resta entre las dos restas de 16 bits. La bandera de acarreo guarda el acarreo para la resta. La figura 5-2 muestra cómo se propaga el acarreo de resta a través de la bandera de acarreo (C) para esta tarea. El ejemplo 5-11 muestra la manera de cómo un programa realiza esta resta. En las restas grandes, los datos de 16 o de 32 bits menos significativos se restan mediante la instrucción SUB. Todos los datos subsiguientes y más significativos se restan mediante el uso de la instrucción SBB.

TABLA 5-5 Ejemplos de instrucciones de decremento.

<i>Lenguaje ensamblador</i>	<i>Operación</i>
DEC BH	BH = BH – 1
DEC CX	CX = CX – 1
DEC EDX	EDX = EDX – 1
DEC BYTE PTR[DI]	Resta 1 del contenido tipo byte de la posición de memoria del segmento de datos direccionada por DI.
DEC WORD PTR[BP]	Resta 1 del contenido tipo palabra de la posición de memoria del segmento de pila direccionada por BP.
DEC DWORD PTR[EBX]	Resta 1 del contenido tipo doble palabra de la posición de memoria del segmento de datos direccionada por EBX.
DEC NUMERO	Resta 1 del contenido de datos de la posición de memoria NUMERO del segmento de datos.

TABLA 5-6 Ejemplos de instrucciones de resta con acarreo.

Lenguaje ensamblador	Operación
SBB AH,AL	AH = AH – AL – acarreo.
SBB AX,BX	AX = AX – BX – acarreo.
SBB EAX,ECX	EAX = EAX – ECX – acarreo.
SBB CL,2	CL = CL – 2 – acarreo.
SBB BYTE PTR[DI],3	El 3 y el acarreo se restan de la posición de memoria del segmento de datos direccionada por DI.
SBB [DI],AL	AL y el acarreo se restan de la posición de memoria del segmento de datos direccionada por DI.
SBB DI,[BP+2]	El acarreo y el contenido tipo palabra de la posición de memoria del segmento de pila direccionada por BP más 2 se restan de DI.
SBB AL,[EBX+ECX]	El acarreo y el contenido tipo byte de la posición de memoria del segmento de datos direccionada por EBX más ECX se restan de AL.

FIGURA 5-2 Una resta con acarreo que muestra cómo la bandera de acarreo (C) propaga el acarreo de resta.

En el ejemplo se usa la instrucción SUB para restar DI de AX, también se usa la instrucción SBB para hacer la resta con acarreo de SI de BX.

EJEMPLO 5-11

0000 2B C7	SUB	AX, DI
0002 1B DE	SBB	BX, SI

Comparación

La instrucción de comparación (CMP) es una resta que sólo cambia los bits de bandera; el operando de destino nunca cambia. Una comparación es útil para comprobar todo el contenido de un registro o de una posición de memoria sobre la base de otro valor. Por lo general, una instrucción CMP va seguida de una instrucción de salto condicional, que evalúa la condición de los bits de bandera.

La tabla 5-7 lista una variedad de instrucciones de comparación que utilizan los mismos modos de direccionamiento que las instrucciones de suma y resta que ya presentamos. De manera similar, las únicas formas de comparación que no se permiten son las comparaciones de memoria a memoria y de los registros de segmento.

El ejemplo 5-12 muestra una comparación seguida de una instrucción de salto condicional. En este ejemplo, el contenido de AL se compara con 10H. Las instrucciones de salto condicional que van por lo general después de la comparación son JA (**salto si es mayor**) y JB (**salto si es menor**). Si la

TABLA 5-7 Ejemplos de instrucciones de comparación.

<i>Lenguaje ensamblador</i>	<i>Operación</i>
CMP CL,BL	CL – BL
CMP AX,SP	AX – SP
CMP EBP,ESI	EBP – ESI
CMP AX,2000H	AX – 2000H
CMP [DI],CH	CH se resta del contenido tipo byte de la posición de memoria del segmento de datos direccionada por DI.
CMP CL,[BP]	El contenido tipo byte de la posición de memoria del segmento de pila direccionada por BP se resta de CL.
CMP AH,TEMP	El contenido tipo byte de la posición de memoria TEMP del segmento de datos se resta de AH.
CMP DI,TEMP[BX]	El contenido tipo palabra de la posición de memoria del segmento de datos direccionada por TEMP más BX se resta de DI.
CMP AL,[EDI+ESI]	El contenido tipo byte de la posición de memoria del segmento de datos direccionada por EDI más ESI se resta de AL.

instrucción JA va después de la comparación, el salto se realiza si el valor en AL es mayor de 10H. Si la instrucción JB va después de la comparación, el salto se realiza si el valor en AL es menor de 10H. En este ejemplo, la instrucción JAE va después de la comparación. Esta instrucción hace que el programa continúe en la posición de memoria SUBER si el valor en AL es 10H o mayor. También hay una instrucción JBE (**salto si es menor o igual**) que podría ir después de la comparación para realizar el salto si el resultado es menor o igual a 10H. En los siguientes capítulos se proporcionarán más detalles sobre las instrucciones de comparación y de salto condicional.

EJEMPLO 5-12

0000 3C 10	CMP	AL,10H	;compara AL con 10H
0002 73 1C	JAE	SUBER	;si AL es 10H o mayor

Comparación e intercambio (sólo en los microprocesadores 80486-Pentium 4). La instrucción de comparación e intercambio (CMPXCHG), que se encuentra solamente en los conjuntos de instrucciones de los microprocesadores del 80486 al Pentium 4, compara el operando de destino con el acumulador. Si son iguales, el operando de origen se copia en el destino; si no son iguales, el operando de destino se copia en el acumulador. Esta instrucción funciona con datos de 8, 16 o 32 bits.

CMPXCHG CX,DX es un ejemplo de la instrucción de comparación e intercambio. Esta instrucción compara primero el contenido de CX con el de AX. Si CX es igual a AX, DX se copia en AX; si CX no es igual a AX, CX se copia en AX. Esta instrucción también compara AL con datos de 8 bits y EAX con datos de 32 bits si los operandos son datos de 8 o de 32 bits.

En los procesadores Pentium-Pentium 4 hay una instrucción CMPXCHG8B disponible que compara dos palabras cuádruples. Ésta es la única instrucción nueva de manipulación de datos que se proporciona en el Pentium-Pentium 4 cuando se les compara con versiones anteriores del microprocesador. La instrucción de comparación e intercambio de 8 bytes compara el valor de 64 bits ubicado en EDX:EAX con un número de 64 bits ubicado en la memoria. Un ejemplo es CMPXCHG8B TEMP. Si TEMP es igual a EDX:EAX, el valor en TEMP se sustituye por el valor que se encuentra en ECX:EBX; si TEMP no es igual a EDX:EAX, el número en TEMP se carga en EDX:EAX. La bandera Z (cero) indica que los valores son iguales después de la comparación.

Esta instrucción tiene un error que hace que el sistema operativo falle. Puede obtener más información sobre este error en www.intel.com.

5-2**MULTIPLICACIÓN Y DIVISIÓN**

Sólo los microprocesadores modernos contienen instrucciones de multiplicación y de división. Los primeros microprocesadores de 8 bits no podían multiplicar o dividir sin el uso de un programa que multiplicara o dividiera mediante el uso de una serie de desplazamientos y sumas o restas. Como los fabricantes de microprocesadores estaban conscientes de esta situación inadecuada, incorporaron instrucciones de multiplicación y de división en los conjuntos de instrucciones de los microprocesadores más recientes. Los procesadores del Pentium al Pentium 4 contienen circuitos especiales que realizan una multiplicación en tan sólo un periodo de reloj, mientras que en los primeros microprocesadores se requerían aproximadamente 40 periodos de reloj para realizar la misma multiplicación.

Multiplicación

La multiplicación se realiza sobre bytes, palabras o dobles palabras y puede ser de tipo entero con signo (IMUL) o de tipo entero sin signo (MUL). Hay que tener en cuenta que sólo los procesadores del 80386 al Pentium 4 multiplican dobles palabras de 32 bits. El producto después de una multiplicación es siempre un producto de doble anchura. Si se multiplican dos números de 8 bits, generan un producto de 16 bits; si se multiplican dos números de 16 bits, generan un producto de 32 bits; y si se multiplican dos números de 32 bits, se genera un producto de 64 bits.

Algunos bits de bandera (O [desbordamiento] y C [acarreo]) cambian cuando se ejecuta la instrucción de multiplicación y se producen resultados predecibles. Las otras banderas también cambian, pero sus resultados son impredecibles y, por lo tanto, no se utilizan. En una multiplicación de 8 bits, si los 8 bits más significativos del resultado son cero los bits de bandera C y O son iguales a cero. Estos bits de bandera muestran que el resultado es de 8 bits ($C = 0$) o de 16 bits ($C = 1$). En una multiplicación de 16 bits, si la parte de los 16 bits más significativos del producto es 0, tanto C como O se vuelven cero. En una multiplicación de 32 bits, tanto C como O indican que los 32 bits más significativos del producto son cero.

Multiplicación de 8 bits. En la multiplicación de 8 bits, el multiplicando siempre se encuentra en el registro AL, sin importar si tiene o no signo. El multiplicador puede ser cualquier registro o posición de memoria de 8 bits. No se permite la multiplicación inmediata, a menos que aparezca la instrucción especial de multiplicación inmediata con signo (que veremos más adelante en esta sección) en un programa. La instrucción de multiplicación contiene un operando porque siempre multiplica el operando por el contenido del registro AL. Un ejemplo es la instrucción MUL BL, que multiplica el contenido sin signo de AL por el contenido sin signo de BL. Después de la multiplicación, el producto sin signo se coloca en AX (un producto de doble anchura). La tabla 5-8 muestra algunas instrucciones de multiplicación de 8 bits.

Suponga que BL y CL contienen cada uno dos números sin signo de 8 bits, y que estos números deben multiplicarse para formar un producto de 16 bits que se va a almacenar en DX. Este procedimiento no puede realizarse en una sola instrucción, ya que sólo podemos multiplicar un número por el

TABLA 5-8 Ejemplos de instrucciones de multiplicación de 8 bits.

Lenguaje ensamblador	Operación
MUL CL	AL se multiplica por CL; el producto sin signo está en AX.
IMUL DH	AL se multiplica por DH; el producto con signo está en AX.
IMUL BYTE PTR[BX]	AL se multiplica por el contenido tipo byte de la posición de memoria del segmento de datos direccionada por BX; el producto con signo está en AX.
MUL TEMP	AL se multiplica por el contenido tipo byte de la posición de memoria TEMP del segmento de datos; el producto sin signo está en AX.

registro AL en una multiplicación de 8 bits. El ejemplo 5-13 muestra un programa corto que genera $DX = BL \times CL$. Este ejemplo carga los registros BL y CL con los datos de ejemplo 5 y 10. El producto (50) se mueve de AX a DX después de la multiplicación, mediante el uso de la instrucción MOV DX,AX.

EJEMPLO 5-13

```

0000 B3 05      MOV    BL,5          ;carga los datos
0002 B1 0A      MOV    CL,10         ;carga los datos
0004 8A C1      MOV    AL,CL        ;posiciona los datos
0006 F6 E3      MUL    BL           ;multiplica
0008 8B D0      MOV    DX,AX        ;posiciona el producto

```

En la multiplicación sin signo, el producto se encuentra en forma binaria si es positivo, y en forma de complemento a dos si es negativo. Éstas son las mismas formas que se utilizan para almacenar todos los números con signo positivo y negativo que utiliza el microprocesador. Si el programa del ejemplo 5-13 multiplica dos números con signo, sólo se cambia la instrucción MUL por IMUL.

Multiplicación de 16 bits. La multiplicación de palabras es muy similar a la multiplicación de bytes. La diferencia es que AX contiene el multiplicando en vez de AL, y el producto de 32 bits aparece en DX-AX en vez de AX. El registro DX siempre contiene los 16 bits más significativos del producto, y AX contiene los 16 bits menos significativos. Al igual que con la multiplicación de 8 bits, la elección del multiplicador depende del programador. La tabla 5-9 muestra varias instrucciones de multiplicación de 16 bits distintas.

Una multiplicación inmediata especial de 16 bits. Los microprocesadores 8086/8088 no podían realizar la multiplicación inmediata; los procesadores del 80186 al Pentium 4 pueden hacerlo mediante el uso de una versión especial de la instrucción de multiplicación. La multiplicación inmediata debe ser multiplicación con signo, y el formato de las instrucciones es distinto porque contiene tres operandos. El primer operando es el registro de destino de 16 bits, el segundo operando es un registro o una posición de memoria que contiene el multiplicando de 16 bits; y el tercer operando está compuesto por datos inmediatos de 8 o de 16 bits que se utilizan como multiplicador.

La instrucción IMUL CX,DX,12H multiplica 12H por DX y deja un producto de 16 bits con signo en CX. Si los datos inmediatos son de 8 bits, se extienden con signo a un número de 16 bits antes de que ocurra la multiplicación. Otro ejemplo es IMUL BX,NUMERO,1000H, que multiplica NUMERO por 1000H y deja el producto en BX. Tanto el destino como el multiplicando deben ser números de 16 bits. Aunque ésta es una multiplicación inmediata, las restricciones que se le imponen limitan su uso, en especial el hecho de que es una multiplicación con signo y que el producto es de 16 bits.

Multiplicación de 32 bits. En los microprocesadores 80386 y superiores se permite la multiplicación de 32 bits, ya que estos microprocesadores contienen registros de 32 bits. Al igual que con la multiplicación de 8 y de 16 bits, la multiplicación de 32 bits puede ser con o sin signo, mediante el uso de las instrucciones IMUL y MUL. En la multiplicación de 32 bits, el contenido de EAX se multiplica por el operando especificado con la instrucción. El producto (de 64 bits) se encuentra en EDX-EAX, en donde EAX contiene los 32 bits menos significativos del producto. La tabla 5-10 lista algunas de las instrucciones de multiplicación de 32 bits que se encuentran en el conjunto de instrucciones de los microprocesadores 80386 y superiores.

TABLA 5-9 Ejemplos de instrucciones de multiplicación de 16 bits.

Lenguaje ensamblador	Operación
MUL CX	AX se multiplica por CX; el producto sin signo está en DX-AX.
IMUL DI	AX se multiplica por DI; el producto con signo está en DX-AX.
MUL WORD PTR[SI]	AX se multiplica por el contenido tipo palabra de la posición de memoria del segmento de datos direccionada por SI; el producto sin signo está en DX-AX.

TABLA 5-10 Ejemplos de instrucciones de multiplicación de 32 bits.

<i>Lenguaje ensamblador</i>	<i>Operación</i>
MUL ECX	EAX se multiplica por ECX; el producto sin signo está en EDX-EAX.
IMUL EDI	EAX se multiplica por EDI; el producto con signo está en EDX-EAX.
MUL DWORD PTR[ESI]	EAX se multiplica por el contenido tipo doble palabra de la posición de memoria del segmento de datos direccionada por ESI; el producto sin signo está en EDX-EAX.

División

Al igual que con la multiplicación, la división ocurre en números de 8 o 16 bits en los microprocesadores 8086-80286, y en números de 32 bits en los microprocesadores 80386-Pentium 4. Estos números pueden ser enteros con signo (IDIV) o sin signo (DIV). El dividendo siempre es un dividendo de doble anchura que se divide entre el operando. Esto significa que una división de 8 bits divide un número de 16 bits entre un número de 8 bits; una división de 16 bits divide un número de 32 bits entre un número de 16 bits; y una división de 32 bits divide un número de 64 bits entre un número de 32 bits. No hay una instrucción de división inmediata disponible para ningún microprocesador.

En una división, ninguno de los bits de bandera cambia de forma predecible. Una división puede producir dos tipos distintos de errores: uno es el intento de dividir entre cero y el otro es un desbordamiento de división. Este último ocurre cuando un número pequeño se divide entre un número grande. Por ejemplo, suponga que AX = 3000 y se divide entre 2. Como el cociente para una división de 8 bits aparece en AL, el resultado de 1500 produce un desbordamiento de división ya que el 1500 no cabe en AL. En cualquier caso, el microprocesador genera una interrupción si ocurre un error de división. En la mayoría de los sistemas, una interrupción por error de división muestra un mensaje de error en la pantalla de vídeo. En el capítulo 6 explicaremos la función de la interrupción por error de división y otras interrupciones para el microprocesador.

División de 8 bits. Esta división utiliza el registro AX para almacenar el dividendo que se va a dividir entre el contenido de cualquier registro o posición de memoria de 8 bits. El cociente se mueve hacia AL después de la división, y AH contiene un residuo tipo número entero. En una división con signo el cociente es positivo o negativo; el residuo siempre asume el signo del dividendo y es siempre un entero. Por ejemplo, si AX = 0010H (+16) y BL = 0FDH (-3), al ejecutarse la división IDIV BL el registro AX = 01FBH. Esto representa un cociente de -5 (AL) con un residuo de 1 (AH). Por otro lado, si -16 se divide entre +3, el resultado será un cociente de -5 (AL) con un residuo de -1 (AH). La tabla 5-11 lista algunas de las instrucciones de división de 8 bits.

En la división de 8 bits, los números por lo general son de 8 bits. Esto significa que uno de ellos (el dividendo) debe convertirse en número de 16 bits en AX. Esto se logra de forma distinta para los números con signo y los números sin signo. En el número sin signo, los 8 bits más significativos deben hacerse cero (**extensión con ceros**). Puede usarse la instrucción MOVZX que describimos en el capítulo 4 para extender con ceros un número en los procesadores del 80386 al Pentium 4. Para los números

TABLA 5-11 Ejemplos de instrucciones de división de 8 bits.

<i>Lenguaje ensamblador</i>	<i>Operación</i>
DIV CL	AX se divide entre CL; el cociente sin signo está en AL y el residuo sin signo está en AH.
IDIV BL	AX se divide entre BL; el cociente con signo está en AL y el residuo con signo está en AH.
DIV BYTE PTR[BP]	AX se divide entre el contenido tipo byte de la posición de memoria del segmento de datos direccionada por BP; el cociente sin signo está en AL y el residuo sin signo está en AH.

con signo, los 8 bits menos significativos se extienden con signo hacia los 8 bits más significativos. En el microprocesador, una instrucción especial extiende con signos AL hacia AH, o convierte un número con signo de 8 bits en AL hacia un número con signo de 16 bits en AX. La instrucción CBW (**convierte byte en palabra**) realiza esta conversión. En los microprocesadores del 80386 al Pentium 4, una instrucción MOVSX (vea el capítulo 4) extiende con signo un número.

EJEMPLO 5-14

0000 A0 0000 R	MOV	AL, NUMERO	; obtiene NUMERO
0003 B4 00	MOV	AH, 0	; extiende con ceros
0005 F6 36 0002 R	DIV	NUMERO1	; divide entre NUMERO1
0009 A2 0003 R	MOV	RESPC, AL	; almacena el cociente
000C 88 26 0004 R	MOV	RESPR, AH	; almacena el residuo

El ejemplo 5-14 ilustra un programa corto que divide el contenido tipo byte sin signo de la posición de memoria NUMERO entre el contenido sin signo de la posición de memoria NUMERO1. Aquí el cociente se almacena en la posición RESPC y el residuo se almacena en la posición RESPR. Observe cómo el contenido de la posición NUMERO se recupera de memoria y luego se extiende con signo para formar un número sin signo de 16 bits para el dividendo.

División de 16 bits. La división de dieciséis bits es similar a la división de 8 bits, sólo que en vez de dividir entre AX, el número de 16 bits se divide entre DX-AX, un dividendo de 32 bits. Despues de una división de 16 bits, el cociente aparece en AX y el residuo aparece en DX. La tabla 5-12 lista algunas de las instrucciones de división de 16 bits.

Al igual que en la división de 8 bits, los números deben convertirse con frecuencia a la forma apropiada para el dividendo. Si se coloca un número de 16 bits sin signo en AX, DX debe volverse cero. En los microprocesadores 80386 y superiores el número se extiende con ceros mediante el uso de la instrucción MOVZX. Si AX es un número de 16 bits con signo, la instrucción CWD (**convierte palabra en doble palabra**) se extiende con signo a un número de 32 bits con signo. Si está disponible el microprocesador 80386 o superior, también puede usarse la instrucción MOVSX para extender con signo un número.

EJEMPLO 5-15

0000 B8 FF9C	MOV	AX, -100	; carga un -100
0003 B9 0009	MOV	CX, 9	; carga +9
0006 99	CWD		; extiende con signo
0007 F7 F9	IDIV	CX	

El ejemplo 5-15 muestra la división de dos números con signo de 16 bits. Aquí, el -100 en AX se divide entre el +9 en CX. La instrucción CWD convierte el -100 en AX a 100 en DX-AX antes de la división. Despues de la división, los resultados aparecen en DX-AX como un cociente de -11 en AX y un residuo de -1 en DX.

División de 32 bits. Los procesadores del 80386 al Pentium 4 realizan la división de 32 bits en números con o sin signo. El contenido de 64 bits de EDX-EAX se divide entre el operando especificado

TABLA 5-12 Ejemplos de instrucciones de división de 16 bits.

Lenguaje ensamblador	Operación
DIV CX	DX-AX se divide entre CX; el cociente sin signo está en AX y el residuo sin signo está en DX.
IDIV SI	DX-AX se divide entre SI; el cociente con signo está en AX y el residuo con signo está en DX.
DIV NUMERO	DX-AX se divide entre el contenido tipo palabra de la posición de memoria NUMERO del segmento de datos; el cociente sin signo está en AX y el residuo sin signo está en DX.

TABLA 5-13 Ejemplos de instrucciones de división de 32 bits.

<i>Lenguaje ensamblador</i>	<i>Operación</i>
DIV ECX	EDX-EAX se divide entre ECX; el cociente sin signo está en EAX y el residuo sin signo en EDX.
IDIV DATOS4	EDX-EAX se divide entre el contenido tipo doble palabra que está en la posición de memoria DATOS4 del segmento de datos; el cociente con signo está en EAX y el residuo con signo en EDX.
DIV DWORD PTR[EDI]	EDX-EAX se divide entre el contenido tipo doble palabra de la posición de memoria del segmento de datos direccionada por EDI; el cociente sin signo está en EAX y el residuo sin signo en EDX.

por la instrucción, lo que deja un cociente de 32 bits en EAX y un residuo de 32 bits en EDX. Aparte del tamaño de los registros, esta instrucción funciona de la misma forma que las divisiones de 8 y de 16 bits. La tabla 5-13 muestra algunas instrucciones de división de 32 bits. La instrucción CDQ (**convierte una doble palabra en palabra cuádruple**) se utiliza antes de una división con signo para convertir el contenido de 32 bits de EAX en un número de 64 bits con signo en EDX-EAX.

El residuo. ¿Qué se hace con el residuo después de una división? Hay algunas opciones. El residuo podría utilizarse para redondear el cociente o simplemente descartarse para truncarlo. Si la división es sin signo, el redondeo requiere que el residuo se compare con la mitad del divisor para decidir en dónde se va a redondear el cociente. El residuo también podría convertirse en un residuo fraccionario.

EJEMPLO 5-16

0000 F6 F3	DIV	BL	;divide
0002 02 E4	ADD	AH,AH	;doble residuo
0004 3A E3	CMP	AH, BL	;evalúa el redondeo
0006 72 02	JB	SIGUIENTE	;si está bien
0008 FE C0	INC	AL	;redondea
000A	SIGUIENTE:		

El ejemplo 5-16 muestra una secuencia de instrucciones que dividen AX entre BL y redondean el resultado sin signo. Este programa duplica el residuo antes de compararlo con BL para decidir si se va a redondear el cociente o no. Aquí, una instrucción INC redondea el contenido de AL después de la comparación.

Suponga que se requiere un residuo fraccionario en vez de un residuo entero. Para ello se guarda el cociente. A continuación se borra el registro AL para volverlo cero. El número que queda en AX se divide ahora por el operando original para generar un residuo fraccionario.

EJEMPLO 5-17

0000 B8 000D	MOV	AX,13	;carga 13
0003 B3 02	MOV	BL,2	;carga 2
0005 F6 F3	DIV	BL	;13/2
0007 A2 0003 R	MOV	RESPC,AL	;guarda el cociente
000A B0 00	MOV	AL,0	;borra AL
000C F6 F3	DIV	BL	;genera el residuo
000E A2 0004 R	MOV	RESPR,AL	;almacena el residuo

El ejemplo 5-17 muestra cómo se divide 13 entre 2. El cociente de 8 bits se almacena en la posición de memoria RESPC y después se borra AL. A continuación, el contenido de AX se divide de nuevo entre 2 para generar un residuo fraccionario. Después de la división, el registro AL es igual a 80H. Esto es 10000000₂. Si el punto binario (base) se coloca antes del bit más a la izquierda de AL, el residuo fraccionario en AL es 0.10000000₂ o 0.5 decimal. El residuo se almacena en la posición de memoria RESPR en este ejemplo.

5-3

ARITMÉTICA BCD Y ASCII

El microprocesador permite la manipulación aritmética de datos BCD (**decimal codificado en binario**) y ASCII (**Código estándar estadounidense para el intercambio de información**). Esto se logra mediante instrucciones que ajustan los números para la aritmética BCD y ASCII.

Las operaciones BCD ocurren en sistemas tales como las terminales de punto de venta (por ejemplo, las cajas registradoras) y en otros sistemas que muy pocas veces requieren de aritmética compleja. Las operaciones ASCII se realizan con los datos ASCII que utilizan muchos programas. En muchos casos, es raro que se utilice la aritmética BCD o ASCII en la actualidad, pero algunas de las operaciones pueden utilizarse para otros fines.

Aritmética BCD

Hay dos técnicas aritméticas que operan con datos BCD: la suma y la resta. El conjunto de instrucciones cuenta con dos instrucciones que corrigen el resultado de una suma BCD y una resta BCD. La instrucción DAA (**ajuste decimal después de la suma**) va después de la suma BCD, y la instrucción DAS (**ajuste decimal después de la resta**) va después de la resta BCD. Ambas instrucciones corrigen el resultado de la suma o de la resta, de manera que sea un número BCD.

Para los datos BCD, los números siempre aparecen en la forma BCD empaquetada y se almacenan como dos dígitos BCD por byte. Las instrucciones de ajuste funcionan sólo con el registro AL después de la suma y la resta BCD.

La instrucción DAA. Esta instrucción va después de la instrucción ADD o ADC para ajustar el resultado y convertirlo en BCD. Suponga que cada uno de los registros DX y BX contienen cuatro números BCD empacados de cuatro dígitos. El ejemplo 5-18 proporciona un programa corto que suma los números BCD en DX y BX, y después almacena el resultado en CX.

EJEMPLO 5-18

0000 BA 1234	MOV DX, 1234H	;carga el 1234 BCD
0003 BB 3099	MOV BX, 3099H	;carga el 3099 BCD
0006 8A C3	MOV AL, BL	;suma BL y DL
0008 02 C2	ADD AL, DL	
000A 27	DAA	
000B 8A C8	MOV CL, AL	;la respuesta va a CL
000D 9A C7	MOV AL, BH	;suma BH, DH con acarreo
000F 12 C6	ADC AL, DH	
0011 27	DAA	
0012 8A E8	MOV CH, AL	;la respuesta va a CH

Como la instrucción DAA funciona solamente con el registro AL, esta suma debe realizarse 8 bits a la vez. Después de sumar los registros BL y DL, el resultado se ajusta con una instrucción DAA antes de almacenarse en CL. Despues se suman los registros BH y DH con acarreo; el resultado se ajusta luego con DAA antes de almacenarse en CH. En este ejemplo, el 1234H se suma al 3099 para generar la suma 4333, que se mueve hacia CX después de la operación. Observe que 1234 BCD es igual que 1234H.

La instrucción DAS. Ésta funciona igual que la instrucción DAA, sólo que va después de una resta en vez de una suma. El ejemplo 5-19 es igual que el ejemplo 5-18, sólo que resta DX y BX en vez de sumarlos. La principal diferencia en estos programas es que la instrucción DAA cambia a DAS, y las instrucciones ADD y ADC cambian a SUB y SBB, respectivamente.

EJEMPLO 5-19

0000 BA 1234	MOV DX, 1234H	;carga el 1234 BCD
0003 BB 3099	MOV BX, 3099H	;carga el 3099 BCD
0006 8A C3	MOV AL, BL	;resta DL a BL
0008 02 C2	SUB AL, DL	

000A 2F	DAS		
000B 8A C8	MOV CL, AL		<i>; la respuesta va a CL</i>
000D 9A C7	MOV AL, BH		<i>; resta DH</i>
000F 1A C6	SBB AL, DH		
0011 2F	DAS		
0012 8A E8	MOV CH, AL		<i>; la respuesta va a CH</i>

Aritmética ASCII

Las instrucciones aritméticas ASCII funcionan con números en código ASCII. El valor de estos números varía de 30H a 39H para los números del 0 al 9. En las operaciones aritméticas ASCII se utilizan cuatro instrucciones: AAA (**ajuste ASCII después de la suma**), AAD (**ajuste ASCII después de la división**), AAM (**ajuste ASCII después de la multiplicación**) y AAS (**ajuste ASCII después de la resta**). Estas instrucciones utilizan el registro AX como origen y como destino.

La instrucción AAA. La suma de dos números de dos dígitos en código ASCII no produce datos útiles. Por ejemplo, si se suman 31H y 39H el resultado es 6AH. Esta suma ASCII (1 + 9) debe producir un resultado ASCII de dos dígitos, equivalente a un 10 decimal, que es un 31H y un 30H en código ASCII. Si la instrucción AAA se ejecuta después de esta suma, el registro AX recibe el número 0100H. Aunque éste no es código ASCII, puede convertirse si se suma el 3030H a AX, con lo que se genera el 3130H. La instrucción AAA borra AH si el resultado es menor que 10 y suma un 1 a AH si el resultado es mayor que 10.

EJEMPLO 5-20

0000 B8 0031	MOV AX, 31H	<i>;carga el 1 ASCII</i>
0003 04 39	ADD AL, 39H	<i>;suma el 9 ASCII</i>
0005 37	AAA	<i>;ajusta la suma</i>
0006 05 3030	ADD AX, 3030H	<i>;la respuesta se convierte a ASCII</i>

El ejemplo 5-20 muestra la manera en que funciona la suma ASCII en el microprocesador. Tome en cuenta que AH se hace cero antes de la suma mediante el uso de la instrucción MOV AX,31H. El operando de 0031H coloca un 00H en AH y un 31H en AL.

La instrucción AAD. A diferencia de las demás instrucciones de ajuste, la instrucción AAD aparece antes de una división. Esta instrucción requiere que el registro AX contenga un número BCD empacado de dos dígitos (no ASCII) antes de ejecutarse. Después de ajustar el registro AX con AAD, se divide entre un número BCD desempacado para generar un resultado de un solo dígito en AL, y cualquier residuo que resulte se coloca en AH.

El ejemplo 5-21 muestra cómo el 72 en BCD desempacado se divide entre 9 para producir un cociente de 8. El 0702H que se carga en el registro AX se ajusta mediante la instrucción AAD a 0048H. Esta operación convierte un número BCD desempacado de dos dígitos en un número binario, de manera que pueda dividirse con la instrucción de división binaria (DIV). La instrucción AAD convierte a binario los números BCD desempacados entre 00 y 99.

EJEMPLO 5-21

0000 B3 09	MOV BL, 9	<i>;carga el divisor</i>
0002 B8 0702	MOV AX, 702H	<i>;carga el dividendo</i>
0005 D5 0A	AAD	<i>;ajuste</i>
0007 F6 F3	DIV BL	<i>;división</i>

La instrucción AAM. Esta instrucción va después de la instrucción de multiplicación, cuando se multiplican dos números BCD desempacados de un dígito. El ejemplo 5-22 muestra un programa corto que multiplica 5 por 5. El resultado después de la multiplicación es 0019H en el registro AX. Después de ajustar el resultado con la instrucción AAM, AX contiene 0205H. Éste es un resultado BCD desempacado de 25. Si se suma el 3030H al 0205H, se obtiene un resultado ASCII de 3235H.

EJEMPLO 5-22

0000 B0 05	MOV AL, 5	;carga el multiplicando
0002 B1 03	MOV CL, 3	;carga el multiplicador
0004 F6 E1	MUL CL	
0006 D4 0A	AAM	;ajuste

La instrucción AAM realiza esta conversión al dividir AX entre 10. El residuo se encuentra en AL y el cociente en AH. El segundo byte de la instrucción contiene 0AH. Si el 0AH se cambia por otro valor, AAM divide entre el nuevo valor. Por ejemplo, si el segundo byte se cambia a 0BH, la instrucción AAM divide entre 11. Esto se realiza mediante el uso de DB 0D4H,0BH en lugar de AAM, con lo que se obliga a que la instrucción AAM multiplique por 11.

Un beneficio adicional de la instrucción AAM es que convierte de binario a BCD desempacado. Si aparece un número binario entre 0000H y 0063H en el registro AX, la instrucción AAM lo convierte en BCD. Por ejemplo, si AX contiene el 0060H antes de AAM, contendrá el 0906H después de que AAM se ejecute. Éste es el equivalente en BCD desempacado del 96 decimal. Si se suma el 3030H al 0906H, el resultado cambia a código ASCII.

El ejemplo 5-23 muestra cómo se convierte el contenido binario de 16 bits de AX en una cadena de caracteres ASCII de cuatro dígitos, mediante el uso de las instrucciones de división y AAM. Esto funciona para los números entre 0 y 9999. Primero DX se hace cero y luego DX-AX se divide entre 100. Por ejemplo, si AX = 245₁₀, después de la división AX = 2 y DX = 45. Estas mitades separadas se convierten en BCD mediante el uso de AAM y después se suma el 3030H para convertirlo en código ASCII.

EJEMPLO 5-23

0000 33 D2	XOR DX, DX	;borra DX
0002 B9 0064	MOV CX, 100	;divide DX-AX entre 100
0005 F7 F1	DIV CX	
0007 D4 0A	AAM	;convierte a BCD
0009 05 3030	ADD AX, 3030H	;convierte a ASCII
000C 92	XCHG AX, DX	;repite para el residuo
000D D4 0A	AAM	
000F 05 3030	ADD AX, 3030H	

El ejemplo 5-24 utiliza la función 21H del DOS AH = 02H para mostrar un número de ejemplo en decimal en la pantalla de vídeo, mediante la instrucción AAM. Observe cómo se utiliza AAM para convertir AL en BCD. A continuación, la instrucción ADD AX,3030H convierte en ASCII el código BCD que está en AX para mostrarlo con la función INT 21H del DOS. Una vez que los datos se convierten en código ASCII, se muestran al cargar DL con el dígito más significativo de AH. Después, el dígito menos significativo se muestra desde AL. Observe que la función INT 21H del DOS modifica AL.

EJEMPLO 5-24

```

;Un programa que muestra el número en AL, el cual se carga
;con la primera instrucción (48H).
;
0000      .MODEL TINY           ;selecciona el modelo diminuto
          .CODE              ;inicio del segmento de código
          .STARTUP            ;inicio del programa
0100 B0 48      MOV  AL, 48H   ;carga los datos de prueba
0102 B4 00      MOV  AH, 0    ;borra AH
0104 D4 0A      AAM             ;convierte en BCD
0106 05 3030    ADD  AX, 3030H ;convierte en ASCII
0109 8A D4      MOV  DL, AH   ;muestra el dígito más significativo
010B B4 02      MOV  AH, 2    ;muestra el dígito menos significativo
010D 50          PUSH AX
010E CD 21      INT  21H
0110 58          POP  AX
0111 8A D0      MOV  DL, AL   ;muestra el dígito menos significativo
0113 CD 21      INT  21H
          .EXIT              ;salida al DOS
          END

```

La instrucción AAS. Al igual que otras instrucciones de ajuste ASCII, AAS ajusta el registro AX después de una resta ASCII. Por ejemplo, suponga que el 35H se resta del 39H. El resultado es 04H, que no requiere corrección. Aquí, AAS no modifica a AH ni a AL. Por otro lado, si el 38H se resta del 37H, entonces AL es igual a 09H y el número en AH se decrementa en 1. Este decremento permite la resta de varios números ASCII de múltiples dígitos.

5-4

INSTRUCCIONES LÓGICAS BÁSICAS

Estas instrucciones son: AND, OR, OR exclusivo y NOT. TEST es otra de las instrucciones lógicas, la cual explicaremos en esta sección porque su operación es una forma especial de la instrucción AND. También explicaremos la instrucción NEG, que es similar a la instrucción NOT.

Las operaciones lógicas proporcionan el control de los bits binarios en el software de bajo nivel. Las instrucciones lógicas permiten establecer, borrar o complementar bits. El software de bajo nivel aparece en forma de lenguaje máquina o de lenguaje ensamblador, y se utiliza a menudo para controlar los dispositivos de E/S en un sistema. Todas las instrucciones lógicas afectan a los bits de bandera. Estas operaciones siempre borran las banderas de acarreo y desbordamiento, mientras que las otras banderas cambian para reflejar la condición del resultado.

Cuando se manipulan datos binarios en un registro o en una posición de memoria, la posición del bit más a la derecha siempre se enumera como el bit 0. Los números de posición de los bits se incrementan desde el bit 0 hacia la izquierda, hasta el bit 7 para un byte, y hasta el bit 15 para una palabra. Una doble palabra (32 bits) utiliza la posición de bit 31 para su bit de más a la izquierda.

AND

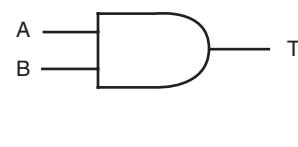
Esta operación realiza la multiplicación lógica, como se muestra en la tabla de verdad en la figura 5-3. Aquí se aplica la instrucción AND a dos bits (A y B) para producir el resultado X. Como lo indica la tabla de verdad, X es un 1 lógico solamente cuando A y B son 1s lógicos. Para todas las demás combinaciones de entrada de A y B, X es un 0 lógico. Es importante recordar que 0 AND nada es siempre 0, y que 1 AND 1 es siempre 1.

La instrucción AND puede sustituir las compuertas AND discretas si la velocidad requerida no es muy grande, aunque por lo general eso se reserva para las aplicaciones de control integradas. (Tenga en cuenta que Intel sacó al mercado el controlador integrado 80386EX, el cual incorpora la estructura básica del sistema de computadora personal.) Con el microprocesador 80386, la instrucción AND por lo general se ejecuta en aproximadamente un microsegundo. En las versiones más recientes del microprocesador, la velocidad de ejecución se incrementa considerablemente. Por ejemplo, el Pentium de 3.0 GHz con su tiempo de reloj de 1/3 ns puede ejecutar hasta tres instrucciones por ciclo del reloj (1/9 ns por cada operación AND). Si el circuito que sustituye la operación AND opera a una velocidad mucho menor que el microprocesador, la instrucción AND es un reemplazo lógico. Este reemplazo puede ahorrar una cantidad considerable de dinero. El circuito integrado de una sola compuerta AND (74HC08) cuesta aproximadamente 40 centavos de dólar, mientras que cuesta menos de 1/100 centavos de dólar.

FIGURA 5-3 (a) La tabla de verdad para la operación AND y (b) el símbolo lógico de una compuerta AND.

A	B	T
0	0	0
0	1	0
1	0	0
1	1	1

(a)



(b)

FIGURA 5-4 La operación de la función AND que muestra cómo se hacen cero los bits de un número.

$\begin{array}{r} \times \times \times \times \times \times \\ \cdot 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \\ \hline 0 \ 0 \ 0 \ x \ x \ x \end{array}$	Número desconocido
	Máscara
	Resultado

guardar la instrucción AND en memoria de sólo lectura. Tenga en cuenta que un reemplazo de circuito lógico como éste sólo aparece en sistemas de control basados en microprocesadores y, por lo general, no encuentra una aplicación en la computadora personal.

La operación AND borra los bits de un número binario. A la tarea de borrar un bit en un número binario se le llama **enmascaramiento**. La figura 5-4 muestra el proceso de enmascaramiento. Observe que los cuatro bits de más a la izquierda se hacen 0, porque 0 AND cualquier cosa es 0. Las posiciones de bit en las que se aplica la operación AND con 1s no cambian. Esto ocurre porque si a un 1 se le aplica AND con 1, el resultado es 1; si a un 1 se le aplica AND con 0, el resultado es 0.

La instrucción AND utiliza cualquier modo de direccionamiento, excepto el de memoria a memoria y el de registros de segmento. La tabla 5-14 muestra algunas instrucciones AND con comentarios sobre su operación.

Un número en código ASCII puede convertirse a BCD mediante el uso de la instrucción AND para enmascarar las cuatro posiciones de bits binarios de más a la izquierda. Esto convierte un número ASCII entre 30H y 39H en un número del 0 al 9. El ejemplo 5-25 muestra un programa corto que convierte el contenido ASCII de BX en BCD. La instrucción AND en este ejemplo convierte dos dígitos de ASCII a BCD en forma simultánea.

EJEMPLO 5-25

0000 BB 3135 0003 81 E3 0F0F	MOV BX, 3135H AND BX, 0F0FH	;carga el valor ASCII ;enmascara BX
---------------------------------	--------------------------------	--

OR

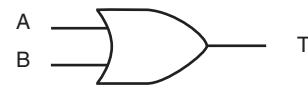
La **operación OR** realiza la suma lógica y se conoce muchas veces como función *OR inclusivo*. La función OR genera una salida de 1 lógico si cualquier entrada es 1. Un 0 aparece en la salida sólo cuando todas las entradas son 0. En la figura 5-5 aparece la tabla de verdad para la función OR. Aquí se aplica OR a las entradas A y B para producir la salida X. Es importante recordar que 1 OR cualquier cosa produce un 1.

TABLA 5-14 Ejemplos de instrucciones AND.

Lenguaje ensamblador	Operación
AND AL,BL	AL = AL and BL
AND CX,DX	CX = CX and DX
AND ECX,EDI	ECX = ECX and EDI
AND CL,33H	CL = CL and 33H
AND DI,4FFFH	DI = DI and 4FFFH
AND ESI, 34H	ESI = ESI and 34H
AND AX,[DI]	Se aplica un AND entre AX y el contenido tipo palabra de la posición de memoria del segmento de datos direccionada por DI.
AND ARREGLO[SI],AL	Se aplica un AND entre AL y el contenido tipo byte de la posición de memoria del segmento de datos direccionada por ARREGLO más SI.
AND [EAX],CL	Se aplica un AND entre CL y el contenido tipo byte de la posición de memoria del segmento de datos direccionada por ECX.

FIGURA 5-5 (a) La tabla de verdad para la operación OR y (b) el símbolo lógico de una compuerta OR.

A	B	T
0	0	0
0	1	1
1	0	1
1	1	1



(a)

(b)

FIGURA 5-6 La operación de la función OR que muestra cómo los bits de un número se hacen uno.

$$\begin{array}{r}
 \text{x } \times \text{x } \times \text{x } \times \text{x } \times \text{x } \\
 + 0 \text{ 0 } 0 \text{ 0 } 1 \text{ 1 } 1 \text{ 1 } \\
 \hline
 \text{x } \times \text{x } 1 \text{ 1 } 1 \text{ 1 }
 \end{array}
 \begin{array}{l}
 \text{Número desconocido} \\
 \text{Máscara} \\
 \text{Resultado}
 \end{array}$$

En las aplicaciones de controladores integrados, la instrucción OR también puede sustituir las compuertas OR discretas. Esto produce ahorros considerables ya que una compuerta OR cuádruple de dos entradas (74HC32) cuesta aproximadamente 40 centavos de dólar, mientras que la instrucción OR cuesta menos de 1/100 centavos de dólar en almacenarse en una memoria de sólo lectura.

La figura 5-6 muestra cómo la compuerta OR activa (1) cualquier bit de un número binario. Aquí se aplica un OR entre un número desconocido (XXXX XXXX) y 0000 1111 para producir un resultado de XXXX 1111. Los cuatro bits de más a la derecha se activan, mientras que los cuatro bits de más a la izquierda permanecen sin cambio. La operación OR activa cualquier bit; la operación AND borra cualquier bit.

La instrucción OR utiliza cualquiera de los modos de direccionamiento permitidos en cualquier otra instrucción, excepto el direccionamiento de registros de segmento. La tabla 5-15 muestra varios ejemplos de instrucciones OR con comentarios sobre su operación.

Suponga que se multiplican dos números BCD y se ajustan con la instrucción AAM. El resultado aparece en AX como un número BCD desempacado de dos dígitos. El ejemplo 5-26 ilustra esta multiplicación y muestra cómo cambiar el resultado a un número en código ASCII de dos dígitos, mediante el uso de la instrucción OR. Aquí, la instrucción OR AX,3030H convierte el 0305H que se encuentra en AX a un 3335H. La operación OR puede sustituirse con una instrucción ADD AX,3030H para obtener los mismos resultados.

TABLA 5-15 Ejemplos de instrucciones OR.

Lenguaje ensamblador	Operación
OR AH,BL	AL = AL or BL
OR SI,DX	SI = SI or DX
OR EAX,EBX	EAX = EAX or EBX
OR DH,0A3H	DH = DH or 0A3H
OR SP,990DH	SP = SP or 990DH
OR EBP,10	EBP = EBP or 10
OR DX,[BX]	Se aplica un OR entre DX y el contenido tipo palabra de la posición de memoria del segmento de datos direccionada por BX.
OR FECHAS[DI+2],AL	Se aplica un OR entre AL y el contenido tipo byte de la posición de memoria del segmento de datos direccionada por DI más 2.

EJEMPLO 5-26

0000 B0 05	MOV AL, 5	;carga los datos
0002 B3 07	MOV BL, 7	
0004 F6 E3	MUL BL	
0006 D4 0A	AAM	
0008 0D 3030H	OR AX, 3030H	;ajuste ;convierte en ASCII

OR exclusivo

La instrucción **OR exclusivo** (XOR) difiere del OR inclusivo (OR) en cuanto a que una condición de 1,1 de la función OR produce un 1; la condición 1,1 de la operación OR exclusivo produce un 0. La operación OR exclusivo excluye esta condición; el OR inclusivo la incluye.

La figura 5-7 muestra la tabla de verdad de la función OR exclusivo. (Compare esta figura con la figura 5-5 para que pueda apreciar la diferencia entre estas dos funciones OR.) Si ambas entradas de la función OR exclusivo son 0 o 1, la salida es 0. Si las entradas son distintas, la salida es 1. Debido a esto, al OR exclusivo se le denomina algunas veces como comparador.

La instrucción XOR utiliza cualquier modo de direccionamiento, excepto el direccionamiento de registros de segmento. La tabla 5-16 muestra varias instrucciones OR exclusivo y sus operaciones.

Al igual que con las funciones AND y OR, el OR exclusivo puede sustituir circuitos lógicos discretos en las aplicaciones integradas. La compuerta OR exclusivo cuádruple de dos entradas (74HC86) se sustituye por una instrucción XOR. El 74HC86 cuesta aproximadamente 40 centavos de dólar, mientras que cuesta menos de 1/100 centavos de dólar almacenar la instrucción en la memoria. La sustitución de sólo un 74HC86 ahorra una cantidad considerable de dinero, en especial si se construyen muchos sistemas.

FIGURA 5-7 (a) La tabla de verdad para la operación OR exclusivo y (b) el símbolo lógico de una compuerta OR exclusivo.

A	B	T
0	0	0
0	1	1
1	0	1
1	1	0



(a)

(b)

TABLA 5-16 Ejemplos de instrucciones OR exclusivo.

Lenguaje ensamblador	Operación
XOR CH,DL	CH = CH xor DL
XOR SI,BX	SI = SI xor BX
XOR EBX,EDI	EBX = EBX xor EDI
XOR AH,0EEH	AH = AH xor 0EEH
XOR DI,00DDH	DI = DI xor 00DDH
XOR ESI,100	ESI = ESI xor 100
XOR DX,[SI]	Se aplica un OR exclusivo entre DX y el contenido tipo palabra de la posición de memoria del segmento de datos direccionada por SI.
XOR TRATO[BP+2],AH	Se aplica un OR exclusivo entre AH y el contenido tipo byte de la posición de memoria del segmento de pila direccionada por BP más 2.

FIGURA 5-8 La operación de la función OR exclusivo que muestra cómo se invierten los bits de un número.

$$\begin{array}{r}
 \text{x } \text{ Número desconocido} \\
 + \text{0 } \text{0 } \text{0 } \text{0 } \text{1 } \text{1 } \text{1 } \text{1 } \text{ Máscara} \\
 \hline
 \text{x } \text{x } \text{x } \text{x } \text{\bar{x}} \text{\bar{x}} \text{\bar{x}} \text{\bar{x}} \text{ Resultado}
 \end{array}$$

La instrucción OR exclusivo es útil si deben invertirse ciertos bits de un registro o posición de memoria. Esta instrucción permite invertir o complementar parte de un número. La figura 5-8 muestra exactamente cómo puede invertirse parte de una cantidad desconocida mediante XOR. Cuando se aplica un OR exclusivo entre un 1 y X, el resultado es X. Si se aplica un OR exclusivo entre 0 y X, el resultado es X.

Suponga que deben invertirse los 10 bits de más a la izquierda del registro BX sin cambiar los seis bits de más a la derecha. La instrucción XOR BX,0FFC0H realiza esta tarea. La instrucción AND borra (0) bits, la instrucción OR los activa (1) y ahora la instrucción OR exclusivo los invierte. Estas tres instrucciones permiten que un programa obtenga el control completo sobre cualquier bit almacenado en cualquier registro o posición de memoria. Esto es ideal para las aplicaciones de sistemas de control en las que el equipo debe encenderse (1), apagarse (0) y cambiar de encendido a apagado o viceversa.

Un uso común para la instrucción OR exclusivo es hacer que un registro sea cero. Por ejemplo, la instrucción XOR CH,CH hace que el registro CH sea 00H y se requieren dos bytes de memoria para almacenar la instrucción. De igual forma, la instrucción MOV CH,00H también hace que CH sea 00H, pero requiere tres bytes de memoria. Debido a este ahorro, la instrucción XOR se utiliza a menudo para borrar un registro, en vez de usar un movimiento inmediato.

El ejemplo 5-27 muestra una secuencia corta de instrucciones que borra los bits 0 y 1 de CX, activa los bits 9 y 10 de CX e invierte el bit 12 de CX. La instrucción OR se utiliza para activar bits, la instrucción AND se utiliza para borrar bits y la instrucción XOR invierte bits.

EJEMPLO 5-27

0000 81 C9 0600	OR	CX,0600H	; activa los bits 9 y 10
0004 83 E1 FC	AND	CX,0FFFCH	; borra los bits 0 y 1
0007 81 F1 1000	XOR	CX,1000H	; invierte el bit 12

Instrucciones TEST y de prueba de bits

La **instrucción TEST** realiza la operación AND. La diferencia es que la instrucción AND modifica el operando de destino, mientras que TEST no. Una instrucción TEST sólo afecta la condición del registro de banderas, el cual indica el resultado de la prueba. La instrucción TEST usa los mismos modos de direccionamiento que la instrucción AND. La tabla 5-17 muestra algunas instrucciones TEST y sus operaciones.

La instrucción TEST funciona de la misma manera que una instrucción CMP. La diferencia es que la instrucción TEST por lo general evalúa un solo bit (o en algunas ocasiones varios bits), mientras que la instrucción CMP evalúa todo el byte, palabra o doble palabra. La bandera cero (Z) es un 1 lógico (lo que indica un resultado de cero) si el bit que se evalúa es un cero, y Z = 0 (lo que indica un resultado distinto de cero) si el bit que se evalúa es distinto de cero.

TABLA 5-17 Ejemplos de instrucciones TEST.

Lenguaje ensamblador	Operación
TEST DL,DH	Se aplica un AND entre DL y DH
TEST CX,BX	Se aplica un AND entre CX y BX
TEST EDX,ECX	Se aplica un AND entre EDX y ECX
TEST AH,4	Se aplica un AND entre AH y 4
TEST EAX,256	Se aplica un AND entre EAX y 256

Por lo general, la instrucción TEST va seguida de la instrucción JZ (**salto si es cero**) o JNZ (**salto si es distinto de cero**). Lo común es evaluar el operando de destino y compararlo con datos inmediatos. El valor de los datos inmediatos es 1 para evaluar la posición del bit de más a la derecha, 2 para evaluar el siguiente bit, 4 para el siguiente y así sucesivamente.

El ejemplo 5-28 muestra un programa corto que evalúa las posiciones de los bits de más a la derecha y de más a la izquierda del registro AL. Aquí, 1 selecciona el bit de más a la derecha y 128 selecciona el bit de más a la izquierda. (Nota: Un 128 es un 80H.) La instrucción JNZ va después de cada prueba para saltar hacia distintas posiciones de memoria, dependiendo del resultado de las pruebas. La instrucción JNZ salta a la dirección del operando (DERECHA o IZQUIERDA en el ejemplo) si el bit que se evalúa es distinto de cero.

EJEMPLO 5-28

0000 A8 01	TEST AL, 1	;evalúa el bit derecho
0002 75 1C	JNZ DERECHA	;si está activado
0004 A8 80	TEST AL, 128	;evalúa el bit izquierdo
0006 75 38	JNZ IZQUIERDA	;si está activado

Los procesadores del 80386 al Pentium 4 contienen instrucciones de prueba adicionales que evalúan las posiciones de bits individuales. La tabla 5-18 lista las cuatro instrucciones de prueba de bits distintas que están disponibles para estos microprocesadores.

Las cuatro formas de la instrucción de prueba de bits evalúan la posición del bit en el operando de destino seleccionado por el operando de origen. Por ejemplo, la instrucción BT AX,4 evalúa la posición del bit 4 en AX. El resultado de la prueba se encuentra en el bit de la bandera de acarreo. Si la posición del bit 4 es 1, el acarreo está activado; si la posición del bit 4 es un 0, el acarreo está desactivado.

Las tres instrucciones de prueba de bits restantes también colocan el bit a evaluar en la bandera de acarreo y lo modifican después de la prueba. La instrucción BTC AX,4 complementa la posición del bit 4 después de evaluarla, la instrucción BTR AX,4 lo borra (0) después de la prueba y la instrucción BTS AX,4 lo activa (1) después de la prueba.

El ejemplo 5-29 repite la secuencia de instrucciones que se listan en el ejemplo 5-27. Aquí la instrucción BTR borra los bits en CX, BTS activa los bits en CX y BTC invierte los bits en CX.

EJEMPLO 5-29

0000 0F BA E9 09	BTS CX, 9	;activa el bit 9
0004 0F BA E9 0A	BTS CX, 10	;activa el bit 10
0008 0F BA F1 00	BTR CX, 0	;borra el bit 0
000C 0F BA F1 01	BTR CX, 1	;borra el bit 1
0010 0F BA F9 0C	BTC CX, 12	;complementa el bit 12

NOT y NEG

La inversión lógica, o **complemento a uno** (NOT), y la inversión aritmética, o **complemento a dos** (NEG), son las últimas dos funciones lógicas que presentaremos (excepto el desplazamiento y el desplazamiento

TABLA 5-18 Instrucciones de prueba de bits.

Lenguaje ensamblador	Operación
BT	Evaluá un bit en el operando de destino, el cual se especifica mediante el operando de origen.
BTC	Evaluá y complementa un bit en el operando de destino, el cual se especifica mediante el operando de origen.
BTR	Evaluá y restablece un bit en el operando de destino, el cual se especifica mediante el operando de origen.
BTS	Evaluá y establece un bit en el operando de destino, el cual se especifica mediante el operando de origen.

TABLA 5-19 Ejemplos de instrucciones NOT y NEG.

<i>Lenguaje ensamblador</i>	<i>Operación</i>
NOT CH	CH se complementa a uno.
NEG CH	CH se complementa a dos.
NEG AX	AX se complementa a dos.
NOT EBX	EBX se complementa a uno.
NEG ECX	ECX se complementa a dos.
NOT TEMP	El contenido de la posición de memoria TEMP del segmento de datos se complementa a uno.
NOT BYTE PTR[BX]	El contenido tipo byte de la posición de memoria del segmento de datos direccionada por BX se complementa a uno.

cíclico que veremos en la siguiente sección.) Éstas son dos de unas cuantas instrucciones que contienen sólo un operando. La tabla 5-19 lista algunas variaciones de las instrucciones NOT y NEG. Al igual que la mayoría de las otras instrucciones, NOT y NEG pueden utilizar cualquier modo de direccionamiento excepto el direccionamiento de registros de segmento.

La instrucción NOT invierte todos los bits de un byte, una palabra o una doble palabra. La instrucción NEG complementa a dos un número, lo que significa que el signo aritmético de un número con signo cambia de positivo a negativo o viceversa. La función NOT se considera lógica; la función NEG se considera una operación aritmética.

5-5

DESPLAZAMIENTO (SHIFT) Y DESPLAZAMIENTO CÍCLICO (ROTATE)

Estas instrucciones manipulan números binarios a nivel de bit binario, como las instrucciones AND, OR, OR exclusivo y NOT. Los desplazamientos y los desplazamientos cíclicos se aplican con más frecuencia en el software de bajo nivel que se utiliza para controlar dispositivos de E/S. El microprocesador contiene un conjunto completo de instrucciones de desplazamiento y de desplazamiento cíclico que se utilizan para desplazar o desplazar en forma cíclica cualquier dato de memoria o registro.

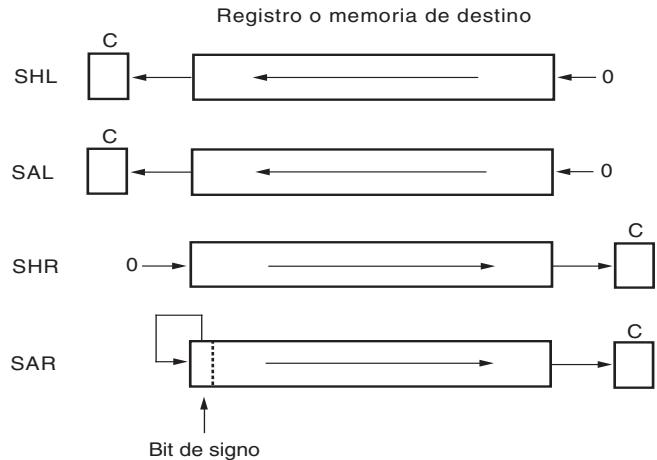
Desplazamiento

Estas instrucciones posicionan o mueven números a la izquierda o a la derecha dentro de un registro o posición de memoria. También realizan aritmética simple tal como la multiplicación por potencias de 2^{+n} (desplazamiento a la izquierda) y la división entre potencias de 2^{-n} (desplazamiento a la derecha). El conjunto de instrucciones del microprocesador contiene cuatro instrucciones de desplazamiento distintas: dos son desplazamientos lógicos y dos son desplazamientos aritméticos. En la figura 5-9 se muestran las cuatro operaciones de desplazamiento.

Observe en la figura 5-9 que hay dos desplazamientos a la derecha y dos a la izquierda. Los desplazamientos lógicos mueven un 0 en la posición de bit de más a la derecha para un desplazamiento lógico a la izquierda, y un 0 en la posición del bit de más a la izquierda para un desplazamiento lógico a la derecha. También hay dos desplazamientos aritméticos. El desplazamiento aritmético a la izquierda y el desplazamiento lógico a la izquierda son idénticos. El desplazamiento aritmético a la derecha y el desplazamiento lógico a la derecha son distintos, ya que el desplazamiento aritmético a la derecha copia el bit de signo al recorrer el número, mientras que el desplazamiento lógico a la derecha copia un 0 al recorrer el número.

Las operaciones de desplazamiento lógico funcionan en números sin signo, y los desplazamientos aritméticos funcionan en números con signo. Los desplazamientos lógicos multiplican o dividen datos sin signo, y los desplazamientos aritméticos multiplican o dividen datos con signo. Un desplazamiento

FIGURA 5-9 Las instrucciones de desplazamiento que muestran la operación y la dirección del desplazamiento.



a la izquierda siempre multiplica por 2 cada posición de bit desplazada, y un desplazamiento a la derecha siempre divide entre 2 cada posición de bit desplazada. Si se desplaza un número dos posiciones, se multiplica o se divide entre 4.

La tabla 5-20 muestra algunos modos de direccionamiento permitidos para las diversas instrucciones de desplazamiento. Existen dos formas distintas de desplazamientos que permiten desplazar cualquier registro (excepto el registro de segmento) o posición de memoria. Uno de los modos utiliza un contador de desplazamiento inmediato y el otro utiliza el registro CL para almacenar la cuenta del desplazamiento. Tenga en cuenta que CL debe almacenar la cuenta de desplazamiento. Cuando esto ocurre, CL no cambia si se ejecuta la instrucción de desplazamiento. El contador de desplazamiento es un contador módulo 32, lo que significa que una cuenta de desplazamiento de 33 desplaza los datos una posición ($33/32 = \text{residuo de } 1$).

El ejemplo 5-30 muestra cómo desplazar el registro DX 14 lugares a la izquierda de dos formas distintas. El primer método utiliza un contador de desplazamiento inmediato de 14. El segundo método carga un 14 en CL y luego utiliza este registro como contador de desplazamiento. Ambas instrucciones desplazan el contenido del registro DX en forma lógica a la izquierda, 14 posiciones o lugares de bits binarios.

EJEMPLO 5-30

0000 C1 E2 0E	SHL DX, 14
○	
0003 B1 0E	MOV CL, 14
0005 D3 E2	SHL DX, CL

TABLA 5-20 Ejemplos de instrucciones de desplazamiento.

Lenguaje ensamblador	Operación
SHL AX,1	AX se desplaza en forma lógica 1 posición a la izquierda.
SHR BX,12	BX se desplaza en forma lógica 12 posiciones a la derecha.
SHR ECX,10	ECX se desplaza en forma lógica 10 posiciones a la derecha.
SAL DATOS1,CL	El contenido de la posición de memoria DATOS1 del segmento de datos se desplaza en forma aritmética a la izquierda, el número de espacios especificado por CL.
SAR SI,2	SI se desplaza en forma aritmética 2 posiciones a la derecha.
SAR EDX,14	EDX se desplaza en forma aritmética 14 posiciones a la derecha.

Suponga que el contenido de AX debe multiplicarse por 10, como se muestra en el ejemplo 5-31. Esto puede hacerse de dos formas: mediante la instrucción MUL o mediante desplazamientos y sumas. Un número se duplica cuando se desplaza una posición a la izquierda. Cuando un número se duplica y luego se suma al mismo número multiplicado por 8, el resultado es 10 veces ese número. El número 10 decimal es 1010 en binario. Un 1 lógico aparece en la posición 2 y 8. Si el número se multiplica por 2 y luego se suma al mismo número multiplicado por 8, el resultado es 10 veces ese número. Mediante esta técnica puede escribirse un programa para multiplicar por cualquier constante. Esta técnica se ejecuta por lo general más rápido que la instrucción de multiplicación que se incluye en las primeras versiones del microprocesador Intel.

EJEMPLO 5-31

```

;Multiplica AX por 10 (1010)
;
0000 D1 E0           SHL    AX,1          ;AX por 2
0002 8B D8           MOV    BX,AX
0004 C1 E0 02         SHL    AX,2          ;AX por 8
0007 03 C3           ADD    AX,BX        ;AX por 10
;
;Multiplica AX por 18 (10010)
;
0009 D1 E0           SHL    AX,1          ;AX por 2
000B 8B D8           MOV    BX,AX
000D C1 E0 03         SHL    AX,3          ;AX por 16
0010 03 C3           ADD    AX,BX        ;AX por 18
;
;Multiplica AX por 5 (101)
;
0012 8B D8           MOV    BX,AX
0014 C1 E0 02         SHL    AH,2          ;AX por 4
0017 03 C3           ADD    AX,BX        ;AX por 5

```

Desplazamientos de doble precisión (sólo en el 80386-Pentium 4). Los microprocesadores 80386 y superiores contienen dos desplazamientos de doble precisión: SHLD (desplazamiento a la izquierda) y SHRD (desplazamiento a la derecha). Cada instrucción contiene tres operandos en vez de los dos que contienen las otras instrucciones de desplazamiento. Ambas instrucciones funcionan con dos registros de 16 o de 32 bits, o con una posición de memoria y un registro de 16 o 32 bits.

La instrucción SHRD AX,BX,12 es un ejemplo de la instrucción de desplazamiento a la derecha de doble precisión. Esta instrucción desplaza en forma lógica el registro AX 12 posiciones a la derecha. Los 12 bits de más a la derecha de BX se desplazan hacia los 12 bits de más a la izquierda de AX. Esta instrucción no modifica el contenido de BX. El contador de desplazamiento puede ser un contador inmediato como en este ejemplo, o puede encontrarse en el registro CL, al igual que en las otras instrucciones de desplazamiento.

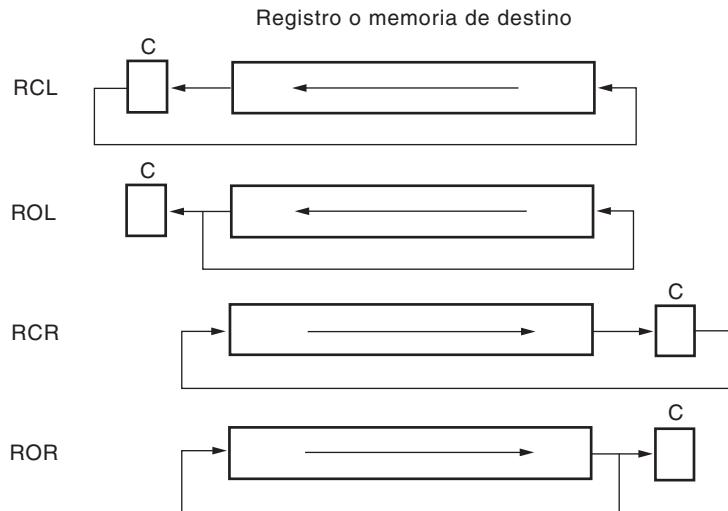
La instrucción SHLD EBX,ECX,16 desplaza el registro EBX a la izquierda. Los 16 bits de más a la izquierda de ECX llenan los 16 bits de más a la derecha de EBX después del desplazamiento. Al igual que antes, el contenido de ECX (el segundo operando) no cambia. Esta instrucción, al igual que SHRD, afecta a los bits de bandera.

Desplazamiento cíclico. Estas instrucciones posicionan datos binarios mediante la rotación de la información en un registro o posición de memoria, ya sea de un extremo a otro o a través de la bandera de acarreo. A menudo se utilizan para desplazar o posicionar números que son mayores de 16 bits en los microprocesadores del 8086 al 80286, o mayores de 32 bits en los microprocesadores del 80386 al Pentium 4. En la figura 5-10 aparecen las cuatro instrucciones de desplazamiento cíclico disponibles.

Los números se desplazan en forma cíclica a través de un registro o de una posición de memoria por medio de la bandera C (acarreo) o a través de un registro o posición de memoria solamente. El programador puede seleccionar un desplazamiento cíclico a la izquierda o a la derecha con cualquiera de los dos tipos de instrucción de desplazamiento cíclico. Los modos de direccionamiento que se utilizan con el desplazamiento cíclico son los mismos que se utilizan con los desplazamientos. Un contador de desplazamiento cíclico puede ser inmediato o puede encontrarse en el registro CL. La tabla 5-21 lista algunas de las instrucciones de desplazamiento cíclico posibles. Si se utiliza CL como contador de

FIGURA 5-10

Las instrucciones de desplazamiento cíclico que muestran la dirección y la operación de cada desplazamiento cíclico.



desplazamiento cíclico, su contenido no cambia. Al igual que con los desplazamientos, el contador en CL es módulo 32.

Las instrucciones de desplazamiento cíclico se utilizan a menudo para desplazar números grandes a la izquierda o a la derecha. El programa que se lista en el ejemplo 5-32 desplaza el número de 48 bits que está en los registros DX, BX y AX una posición binaria a la izquierda. Observe que los 16 bits menos significativos (AX) se desplazan primero a la izquierda. Esto hace que se mueva el bit de más a la izquierda de AX hacia el bit de bandera de acarreo. A continuación, la instrucción de desplazamiento cíclico de BX desplaza en forma cíclica el acarreo hacia BX, y su bit de más a la izquierda se mueve hacia el acarreo. La última instrucción desplaza el acarreo en forma cíclica hacia DX, y con esto termina el desplazamiento.

EJEMPLO 5-32

0000 D1 E0	SHL AX,1
0002 D1 D3	RCL BX,1
0004 D1 D2	RCL DX,1

Instrucciones de exploración de bits

Aunque estas instrucciones no desplazan números con instrucciones de desplazamiento o de desplazamiento cíclico, sí exploran a través de un número en busca de un bit cuyo valor sea 1. Como esto se realiza dentro del microprocesador mediante el desplazamiento del número, incluimos las instrucciones de exploración de bits en esta sección.

TABLA 5-21 Ejemplos de instrucciones de desplazamiento cíclico.

Lenguaje ensamblador	Operación
ROL SI,14	SI se desplaza en forma cíclica 14 posiciones a la izquierda.
RCL BL,6	BL se desplaza en forma cíclica 6 posiciones a la izquierda, a través del acarreo.
ROL ECX,18	ECX se desplaza en forma cíclica 18 posiciones a la izquierda
RCR AH,CL	AH se desplaza en forma cíclica a través del acarreo el número de posiciones que especifique CL.
ROR WORD PTR[BP],2	El contenido tipo palabra de la posición de memoria del segmento de pila direccionada por BP se desplaza en forma cíclica 2 lugares a la derecha.

Las instrucciones de exploración de bits BSF (**exploración de bits hacia delante**) y BSR (**exploración de bits hacia atrás**) están disponibles sólo en los procesadores del 80386 al Pentium 4. Ambas formas exploran a través del número de origen, en busca del primer bit que sea 1. La instrucción BSF explora el número desde el bit de más a la izquierda y lo recorre hacia la derecha, y BSR explora el número desde el bit de más a la derecha y lo recorre hacia la izquierda. Si se encuentra un bit que sea 1, se activa la bandera de cero y el número de posición de bit que es 1 se coloca en el operando de destino. Si no se encuentra un bit que sea 1 (es decir, que el número sólo contenga ceros), se borra la bandera cero. Por lo tanto, el resultado es distinto de cero si no se encuentra un bit que sea 1.

Por ejemplo, si EAX = 6000000H y se ejecuta la instrucción BSF EBX,EAX, el número se explora desde el bit de más a la izquierda y se recorre hacia la derecha. El primer bit que vale 1 se encuentra en la posición del bit 30, el cual se coloca en EBX y se activa la bandera de cero. Si se utiliza el mismo valor de EAX para la instrucción BSR, el registro EBX se carga con 29 y se activa el bit de bandera cero.

5-6

COMPARACIONES DE CADENAS

Como se ilustró en el capítulo 4, las instrucciones de cadena son muy poderosas ya que permiten que el programador manipule grandes bloques de datos con relativa facilidad. La manipulación de bloques de datos se lleva a cabo con las instrucciones de cadena MOVS, LODS, STOS, INS y OUTS. En esta sección se verán las instrucciones de cadena adicionales que permiten evaluar una sección de memoria para compararla con una constante o con otra sección de memoria. Para realizar estas tareas se utilizarán las instrucciones SCAS (**exploración de cadena**) o CMPS (**comparación de cadena**).

SCAS

Esta instrucción compara el registro AL con un bloque de bytes en la memoria, el registro AX con un bloque de palabras en la memoria, o el registro EAX (80386-Pentium 4) con un bloque de dobles palabras en la memoria. La instrucción SCAS resta la memoria de AL, AX o EAX sin afectar ni al registro ni a la posición de memoria. El código de operación que se utiliza para la comparación de bytes es SCASB, el código de operación que se utiliza para la comparación de palabras es SCASW y el código de operación que se utiliza para la comparación de dobles palabras es SCASD. En todos los casos se compara el contenido de la posición de memoria del segmento extradireccionada por DI con AL, AX o EAX. Recuerde que este segmento predeterminado (ES) no puede cambiarse mediante un prefijo de sustitución de segmento.

Al igual que las otras instrucciones de cadena, las instrucciones SCAS utilizan la bandera de dirección (D) para seleccionar la operación de autoincremento o de autodecremento para DI. También se repiten si se utiliza un prefijo de repetición condicional.

Suponga que una sección de memoria es de 100 bytes y que empieza en la posición BLOQUE. Esta sección de memoria debe evaluarse para ver si alguna posición contiene el valor 00H. El programa en el ejemplo 5-33 muestra cómo buscar el 00H en esta parte de la memoria mediante la instrucción SCASB. En este ejemplo, la instrucción SCASB tiene un prefijo REPNE (**se repite mientras no sea igual**). Este prefijo hace que la instrucción SCASB se repita hasta que el registro CX llegue a 0, o hasta que exista una condición de igualdad como resultado de la comparación de la instrucción SCASB. Otro prefijo de repetición condicional es REPE (**se repite mientras sea igual**). Con cualquiera de los dos prefijos de repetición, el contenido de CX se decrementa sin afectar a los bits de bandera. La instrucción SCASB y la comparación que hace modifica las banderas.

EJEMPLO 5-33

0000	BF	0011	R	MOV	DI,OFFSET BLOQUE	;direcciona los datos
0003	FC			CLD		;autoincremento
0004	B9	0064		MOV	CX,100	;carga el contador
0007	32	C0		XOR	AL,AL	;borra AL
0009	F2	/ AE		REPNE	SCASB	

Suponga que debe desarrollar un programa que ignore los espacios en código ASCII en un arreglo de memoria. (Esta tarea aparece en el procedimiento listado en el ejemplo 5-34.) Este procedimiento supone que el registro DI ya direcciona la cadena de caracteres en código ASCII y que la longitud de esta cadena es de 256 bytes o menor. Como este programa debe ignorar los espacios (20H), se utiliza el prefijo REPE con una instrucción SCASB. Esta instrucción repite la comparación en busca de un 20H, siempre y cuando exista una condición de igualdad.

EJEMPLO 5-34

0000	FC		CLD		;autoincremento
0001	B9	0100	MOV	CX, 256	;carga el contador
0004	B0	20	MOV	AL, 20H	;obtiene un espacio
0006	F3 / AE		REPE	SCASB	

CMP\$

Esta instrucción siempre compara dos secciones de datos en memoria como bytes (CMPSB), palabras (CMPSW) o dobles palabras (CMPSD). Tenga en cuenta que sólo los microprocesadores del 80386 al Pentium 4 pueden utilizar dobles palabras. El contenido de la posición de memoria del segmento de datos direccionada por SI se compara con el contenido de la posición de memoria del segmento extra-direccionada por DI. La instrucción CMPS incrementa o decrementa a SI y a DI. Esta instrucción se utiliza por lo general con el prefijo REPE o REPNE. Hay otros dos prefijos como alternativa para los anteriores: REPZ (se repite mientras sea cero) y REPNZ (se repite mientras sea distinto de cero), pero por lo general se utilizan los prefijos REPE o REPNE en la programación.

El ejemplo 5-35 muestra un procedimiento corto que compara dos secciones de memoria en busca de una concordancia. La instrucción CMPSB contiene el prefijo REPE. Esto hace que la búsqueda continúe siempre y cuando exista una condición de igualdad. Si el registro CX se hace 0 o si existe una condición desigual, la instrucción CMPSB deja de ejecutarse. Al terminar la instrucción CMPSB el registro CX se hace 0 o las banderas indican una condición de igualdad cuando las dos cadenas concuerdan. Si CX no es 0 o si las banderas indican una condición desigual, las cadenas no concuerdan.

EJEMPLO 5-35

0000	BE	0075	R	MOV	SI,OFFSET LINEA	;direcciona LINEA
0003	BF	007F	R	MOV	DI,OFFSET TABLA	;direcciona TABLA
0006	FC			CLD		;autoincremento
0007	B9	000A		MOV	CX, 10	;carga el contador
000A	F3 / A6			REPE	CMPSB	;búsqueda

1. La suma (ADD) puede ser de 8, de 16 o de 32 bits. La instrucción ADD permite cualquier modo de direccionamiento excepto el direccionamiento de registros de segmento. La mayoría de las banderas (C, A, S, Z, P y O) se modifican cuando se ejecuta la instrucción ADD. Un tipo distinto de suma llamado suma con acarreo (ADC) suma dos operandos y el contenido de la bandera de acarreo (C). Los procesadores del 80486 al Pentium 4 tienen una instrucción adicional (XADD) que combina una suma con un intercambio.
2. La instrucción de incremento (INC) suma 1 al contenido tipo byte, palabra o doble palabra de un registro o posición de memoria. Esta instrucción afecta a los mismos bits de bandera que ADD, excepto la bandera de acarreo. Las directivas BYTE PTR, WORD PTR y DWORD PTR aparecen con la instrucción INC cuando el contenido de una posición de memoria se direcciona mediante un apuntador.

3. La resta (SUB) puede ser de bytes, palabras o dobles palabras y se realiza en un registro o en una posición de memoria. La única forma de direccionamiento que no se permite en la instrucción SUB es el direccionamiento de registros de segmento. La instrucción de suma afecta a las mismas banderas que ADD y resta el acarreo si se utiliza la forma SBB.
4. La instrucción de decremento (DEC) resta 1 al contenido de un registro o de una posición de memoria. Los únicos modos de direccionamiento que no se permiten con DEC son el direccionamiento inmediato o el de registros de segmento. La instrucción DEC no afecta a la bandera de acarreo y se utiliza a menudo con BYTE PTR, WORD PTR o DWORD PTR.
5. La instrucción de comparación (CMP) es una forma especial de resta en la que no se almacena la diferencia, sino que las banderas cambian para reflejarla. Esta instrucción se utiliza para comparar un byte o una palabra completos, que pueden estar ubicados en cualquier registro (excepto los de segmento) o posición de memoria. En los microprocesadores del 80486 al Pentium 4 hay una instrucción de comparación adicional (CMPXCHG) que es una combinación de las instrucciones de comparación y de intercambio. En los procesadores del Pentium al Pentium 4, la instrucción CMPXCHG8B compara e intercambia datos tipo palabra cuádruple.
6. La multiplicación puede ser de bytes, palabras o dobles palabras, y puede ser con signo (IMUL) o sin signo (MUL). En la multiplicación de 8 bits siempre se multiplica el registro AL por un operando y el producto se deja en AX. En la multiplicación de 16 bits siempre se multiplica el registro AX por un operando y el producto se deja en DX-AX. En la multiplicación de 32 bits siempre se multiplica el registro EAX por un operando y el producto se deja en EDX-EAX. En los procesadores del 80186 al Pentium 4 existe una instrucción IMUL especial que contiene tres operandos. Por ejemplo, la instrucción IMUL BX,CX,3 multiplica CX por 3 y deja el producto en BX.
7. La división puede ser de bytes, palabras o dobles palabras y puede ser con signo (IDIV) o sin signo (DIV). En una división de 8 bits el registro AX se divide entre el operando; el cociente aparece en AL y el residuo en AH. En la división de 16 bits el registro DX-AX se divide entre el operando; después de la división el registro AX contiene el cociente y DX contiene el residuo. En la división de 32 bits el registro EDX-EAX se divide entre el operando, después de lo cual el registro EAX contiene el cociente y EDX el residuo. El residuo después de una división con signo siempre asume el signo del dividendo.
8. Los datos BCD se suman o se restan en forma empacada, mediante el ajuste del resultado de la suma con DAA o de la resta con DAS. Los datos ASCII se suman, se restan, se multiplican o se dividen cuando las operaciones se ajustan con AAA, AAS, AAM y AAD.
9. La instrucción AAM tiene una característica adicional interesante, la cual le permite convertir un número binario en BCD desempacado. Esta instrucción convierte un número binario entre 00H-63H en BCD desempacado y lo coloca en AX. La instrucción AAM divide AX entre 10; después deja el residuo en AL y el cociente en AH.
10. Las instrucciones AND, OR y OR exclusivo realizan funciones lógicas sobre un byte, una palabra o una doble palabra almacenada en un registro o posición de memoria. Todas las banderas cambian con estas instrucciones, en donde se borran las banderas de acarreo (C) y desbordamiento (O).
11. La instrucción TEST realiza la operación AND, pero se pierde el producto lógico. Esta instrucción cambia los bits de bandera para indicar el resultado de la prueba.
12. Las instrucciones NOT y NEG realizan la inversión lógica y la inversión aritmética. La instrucción NOT complementa el operando a uno y la instrucción NEG complementa el operando a dos.
13. Existen ocho instrucciones distintas de desplazamiento (shift) y desplazamiento cíclico (rotate). Cada una de estas instrucciones trabaja con datos tipo byte, palabra o doble palabra. Estas instrucciones tienen dos operandos: el primero es la posición de los datos que se van a desplazar y el segundo es un contador de desplazamiento o desplazamiento cíclico, o CL. Si el segundo operando es CL, el contador se guarda en el registro CL. Los procesadores del 80386 al Pentium 4 tienen dos desplazamientos adicionales de doble precisión (SHRD y SHLD).
14. La instrucción de exploración de cadenas (SCAS) compara AL, AX o EAX con el contenido de la posición de memoria del segmento extradireccional por DI.
15. La instrucción de comparación de cadenas (CMPS) compara el contenido tipo byte, palabra o doble palabra de dos secciones de memoria. Una sección se direcciona mediante DI en el segmento extra, y la otra se direcciona mediante SI en el segmento de datos.

16. Las instrucciones SCAS y CMPS se repiten con los prefijos REPE o REPNE. El prefijo REPE repite la instrucción de cadena mientras exista una condición de igualdad, y la instrucción REPNE repite la instrucción de cadena mientras exista una condición desigual.

5-8**PREGUNTAS Y PROBLEMAS**

1. Seleccione una instrucción ADD que:
 - (a) Sume BX a AX.
 - (b) Sume 12H a AL.
 - (c) Sume EDI y EBP.
 - (d) Sume 22H a CX.
 - (e) Sume a AL los datos direccionados por SI.
 - (f) Sume CX a los datos almacenados en la posición de memoria RANA.
2. ¿Cuál es el error en la instrucción ADD ECX,AX?
3. ¿Es posible sumar CX a DS con la instrucción ADD?
4. Si AX = 1001H y DX = 20FFH, liste la suma y el contenido de cada bit del registro de banderas (C, A, S, Z y O) después de ejecutar la instrucción ADD AX,DX.
5. Desarrolle una secuencia corta de instrucciones para sumar AL, BL, CL, DL y AH. Almacene la suma en el registro DH.
6. Desarrolle una secuencia corta de instrucciones para sumar AX, BX, CX, DX y SP. Almacene la suma en el registro DI.
7. Desarrolle una secuencia corta de instrucciones para sumar ECX, EDX y ESI. Almacene la suma en el registro EDI.
8. Seleccione una instrucción para sumar BX a DX, y que también sume el contenido de la bandera de acarreo (C) al resultado.
9. Seleccione una instrucción que sume 1 al contenido del registro SP.
10. ¿Cuál es el error en la instrucción INC [BX]?
11. Seleccione una instrucción SUB que:
 - (a) Reste BX de CX.
 - (b) Reste 0EEH de DH.
 - (c) Reste DI de SI.
 - (d) Reste 3322H de EBP.
 - (e) Reste de CH la dirección de datos direccionada por SI.
 - (f) Reste de DX los datos almacenados 10 palabras después de la posición direccionada por SI.
 - (g) Reste AL de la posición de memoria RANA.
12. Si DL = 0F3H y BH = 72H, liste la diferencia después de restar BH de DL y muestre el contenido de los bits del registro de banderas.
13. Escriba una secuencia corta de instrucciones para restar los números en DI, SI y BP del registro AX. Almacene la diferencia en el registro BX.
14. Seleccione una instrucción que reste 1 del registro EBX.
15. Explique la función de la instrucción SBB [DI-4],DX.
16. Explique la diferencia entre las instrucciones SUB y CMP.
17. Cuando se multiplican dos números de 8 bits, ¿en dónde se encuentra el producto?
18. Cuando se multiplican dos números de 16 bits, ¿cuáles son los dos registros que guardan el producto? Muestre los registros que contienen las porciones más y menos significativas del producto.
19. Cuando se multiplican dos números, ¿qué ocurre con los bits de las banderas O y C?
20. ¿En dónde se almacena el producto de la instrucción MUL EDI?
21. ¿Cuál es la diferencia entre las instrucciones IMUL y MUL?
22. Escriba una secuencia de instrucciones que eleven al cubo el número de 8 bits que se encuentra en DL. Cargue DL con 5 al principio y asegúrese de que su resultado sea un número de 16 bits.
23. Describa la operación de la instrucción IMUL BX,DX,100H.

24. Cuando se dividen números de 8 bits, ¿en cuál registro se encuentra el dividendo?
25. Cuando se dividen números de 16 bits, ¿en cuál registro se encuentra el cociente?
26. ¿Qué errores se detectan durante una división?
27. Explique la diferencia entre las instrucciones IDIV y DIV.
28. ¿En dónde se encuentra el residuo después de una división de 8 bits?
29. Escriba una secuencia corta de instrucciones que divida el número en BL entre el número en CL, y luego multiplique el resultado por 2. La respuesta final debe ser un número de 16 bits almacenado en el registro DX.
30. ¿Qué instrucciones se utilizan con las operaciones aritméticas BCD?
31. ¿Qué instrucciones se utilizan con las operaciones aritméticas ASCII?
32. Explique cómo la instrucción AAM convierte de binario a BCD.
33. Desarrolle una secuencia de instrucciones que conviertan el número sin signo en AX (valores de 0-65535) en un número BCD de 5 dígitos que se almacene en memoria, empezando en la posición direccionada por el registro BX en el segmento de datos. Observe que el carácter más significativo se almacena primero y no se hace un intento por borrar los ceros a la izquierda.
34. Desarrolle una secuencia de instrucciones que sumen el número BCD de 8 dígitos en AX y BX con el número BCD de 8 dígitos en CX y DX. (AX y CX son los registros más significativos. El resultado debe encontrarse en CX y DX después de la suma.)
35. Seleccione una instrucción AND que:
 - (a) Realice una operación AND entre BX y DX, y que guarde el resultado en BX.
 - (b) Realice una operación AND entre 0FAH y DH.
 - (c) Realice una operación AND entre DI y BP, y que guarde el resultado en DI.
 - (d) Realice una operación AND entre 1122H y EAX.
 - (e) Realice una operación AND entre los datos direccionados por BP y CX, y que guarde el resultado en memoria.
 - (f) Realice una operación AND entre DX y los datos almacenados en cuatro palabras antes de la posición direccionada por SI, y que guarde el resultado en DX.
 - (g) Realice una operación AND entre AL y la posición de memoria CUAL, y que guarde el resultado en la posición CUAL.
36. Desarrolle una secuencia corta de instrucciones para borrar (0) los tres bits de más a la izquierda de DH sin cambiar el resto de DH y que guarde el resultado en BH.
37. Seleccione una instrucción OR que:
 - (a) Realice una operación OR entre BL y AH, y que guarde el resultado en AH.
 - (b) Realice una operación OR entre 88H y ECX.
 - (c) Realice una operación OR DX y SI, y que guarde el resultado en SI.
 - (d) Realice una operación OR entre 1122H y BP.
 - (e) Realice una operación OR entre CX y los datos direccionados por BX, y que guarde el resultado en memoria.
 - (f) Realice una operación OR entre AL y los datos almacenados 40 bytes después de la posición direccionada por BP, y que guarde el resultado en AL.
 - (g) Realice una operación OR entre AH y la posición de memoria CUANDO, y que guarde el resultado en CUANDO.
38. Desarrolle una secuencia corta de instrucciones para activar (1) los cinco bits de más a la derecha de DI, sin cambiar los bits restantes de DI. Almacene los resultados en SI.
39. Seleccione la instrucción XOR que:
 - (a) Realice una operación XOR entre BH y AH, y que guarde el resultado en AH.
 - (b) Realice una operación XOR entre 99H y CL.
 - (c) Realice una operación XOR entre DX y DI, y que guarde el resultado en DX.
 - (d) Realice una operación XOR entre 1A23H y ESP.
 - (e) Realice una operación XOR entre DX y los datos direccionados por EBX, y que guarde el resultado en memoria.
 - (f) Realice una operación XOR entre DI y los datos almacenados 30 palabras después de la posición direccionada por BP, y que guarde el resultado en DI.
 - (g) Realice una operación XOR entre DI y la posición de memoria BIEN, y que guarde el resultado en DI.

40. Desarrolle una secuencia de instrucciones para activar (1) los cuatro bits de más a la derecha de AX; que borre (0) los tres bits de más a la izquierda de AX; y que invierta los bits 7, 8 y 9 de AX.
41. Describa la diferencia entre las instrucciones AND y TEST.
42. Seleccione una instrucción para evaluar la posición del bit 2 del registro CH.
43. ¿Cuál es la diferencia entre la instrucción NOT y la instrucción NEG?
44. Seleccione la instrucción correcta para realizar cada una de las siguientes tareas:
 - (a) Desplazar DI tres posiciones a la derecha; los ceros deben moverse hacia el bit de más a la izquierda.
 - (b) Mover todos los bits en AL una posición a la izquierda, asegurándose que un 0 se mueva hacia la posición del bit de más a la derecha.
 - (c) Desplazar en forma cíclica todos los bits de AL tres posiciones a la izquierda.
 - (d) Desplazar en forma cíclica el acarreo una posición a través de EDX.
 - (e) Mover el registro DH una posición a la derecha, asegurándose que el signo del resultado sea igual que el signo del número original.
45. ¿Qué hace la instrucción SCASB?
46. Para las instrucciones de cadena, DI siempre direcciona los datos en el segmento de _____.
47. ¿Cuál es el propósito del bit de bandera D?
48. Explique qué hace el prefijo REPE cuando se utiliza con la instrucción SCASB.
49. ¿Qué condición o condiciones terminan la instrucción de cadena repetida REPNE SCASB?
50. Describa qué hace la instrucción CMPSB.
51. Desarrolle una secuencia de instrucciones para explorar a través de una sección de memoria de 300H bytes llamada LISTA, ubicada en el segmento de datos; el número a buscar es 66H.
52. ¿Qué ocurre si AH = 02H y DL = 43H cuando se ejecuta la instrucción INT 21H?

CAPÍTULO 6

Instrucciones de control de programas

INTRODUCCIÓN

Las instrucciones de control de programas dirigen el flujo de un programa y permiten que este flujo cambie. Muchas veces ocurre un cambio en el flujo cuando existe una instrucción de salto condicional después de una decisión realizada con la instrucción CMP o TEST. En este capítulo se explicarán las instrucciones de control de programas: saltos, llamadas, retornos, interrupciones e instrucciones de control de la máquina.

En este capítulo también se presentarán las instrucciones relacionales en lenguaje ensamblador (.IF, .ELSE, .ELSEIF, .ENDIF, .WHILE, .ENDW, .REPEAT y .UNTIL) disponibles en la versión 6.xx o superior de MASM o de TASM; estableceremos la versión 5.xx para compatibilidad con MASM. Estos comandos relacionales en lenguaje ensamblador permiten al programador desarrollar las porciones del programa relacionadas con el flujo de control, con una eficiencia similar al lenguaje C/C++.

OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

1. Utilizar las instrucciones de salto condicional e incondicional para controlar el flujo de un programa.
2. Utilizar en los programas las instrucciones relacionales en lenguaje ensamblador .IF, .REPEAT, .WHILE y demás instrucciones.
3. Utilizar las instrucciones de llamada y retorno para incluir procedimientos en la estructura del programa.
4. Explicar la operación de las interrupciones y de las instrucciones de control de interrupciones.
5. Utilizar las instrucciones de control de máquina para modificar los bits de bandera.
6. Utilizar ENTER y LEAVE para entrar y salir de las estructuras de programación.

6-1

EL GRUPO DE SALTOS

La instrucción principal de control de un programa, conocida como **jump** (*JMP*), permite al programador omitir secciones de un programa y ramificarse a cualquier parte de la memoria para la siguiente instrucción. Una instrucción de salto condicional permite al programador realizar decisiones con base en pruebas numéricas. El resultado de las pruebas numéricas se guarda en los bits de bandera, que después se evalúan mediante instrucciones de salto condicional. Además de las instrucciones de salto condicional, en esta sección también explicaremos el ajuste condicional, otra instrucción similar al salto condicional.

En esta sección se ilustrará el uso de todas las instrucciones de salto mediante programas de ejemplo. También se dará un repaso de las instrucciones LOOP y LOOP condicional, las cuales vimos por primera vez en el capítulo 3, ya que también son formas de la instrucción de salto.

Salto incondicional (JMP)

Hay tres tipos de instrucciones de salto condicional (vea la figura 6-1) disponibles para el microprocesador: salto corto, salto cercano y salto lejano. El **salto corto** es una instrucción de dos bytes que permite saltos o ramificaciones hacia posiciones de memoria que estén dentro del rango de +127 y -128 bytes desde la dirección que sigue después del salto. El **salto cercano** de tres bytes permite una ramificación o salto dentro del rango de ± 32 Kbytes (o en cualquier parte del segmento de código actual) desde la instrucción en el segmento de código actual. Recuerde que los segmentos son cíclicos por naturaleza, lo que significa que una posición más allá de la dirección de desplazamiento FFFFH es la dirección de desplazamiento 0000H. Por esta razón, si usted salta dos bytes hacia delante en la memoria y el apuntador de instrucciones direcciona la dirección de desplazamiento FFFFH, el flujo continúa en la dirección de desplazamiento 0001H. Por ende, un desplazamiento de ± 32 Kbytes permite un salto a cualquier posición dentro del segmento de código actual. Por último, la instrucción **salto lejano** de cinco bytes permite un salto hacia cualquier posición de memoria dentro del sistema de memoria real. Los saltos cortos y cercanos se denominan a menudo **saltos intrasegmento**, y los saltos lejanos se denominan a menudo **saltos intersegmento**.

En los microprocesadores del 80386 al Pentium 4, el salto cercano está dentro del rango de ± 2 Gbytes si el equipo se opera en modo protegido, con un segmento de código de 4 Gbytes. Si se opera en el modo real, el salto cercano está dentro del rango de ± 32 Kbytes. En el modo protegido, los microprocesadores 80386 y superiores utilizan un desplazamiento de 32 bits que no se muestra en la figura 6-1.

Salto corto. Los saltos cortos se llaman **saltos relativos** ya que pueden moverse, junto con su software relacionado, hacia cualquier posición en el segmento de código actual sin necesidad de modificarse. Esto se debe a que la dirección de salto no se almacena con el código de operación. En vez de una dirección de salto, se utiliza una **distancia** o desplazamiento después del código de operación. El desplazamiento del salto corto es una distancia que se representa mediante un número de un byte con signo, cuyo valor está entre +127 y -128. La instrucción de salto corto aparece en la figura 6-2. Cuando el microprocesador ejecuta un salto corto, el desplazamiento se extiende con signo y se suma al apuntador de instrucciones (IP/EIP) para generar la dirección de salto dentro del segmento de código actual. La instrucción de salto corto se ramifica hacia esta nueva dirección para la siguiente instrucción en el programa.

El ejemplo 6-1 muestra cómo las instrucciones de salto corto pasan el control de una parte del programa a otra. También ilustra el uso de una **etiqueta** (un nombre simbólico para una dirección de memoria) con la instrucción de salto. Observe cómo un salto (JMP SHORT SIGUIENTE) utiliza la directiva SHORT para forzar un salto corto, mientras que el otro no. La mayoría de los programas ensambladores seleccionan la mejor forma de instrucción de salto, de manera que la segunda instrucción

FIGURA 6-1 Las tres formas principales de la instrucción JMP. Observe que Desp puede ser un desplazamiento o distancia con signo de 8 o de 16 bits.

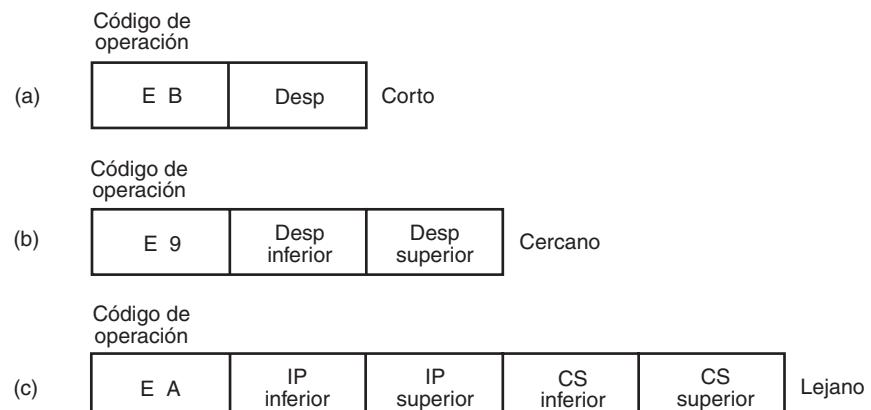
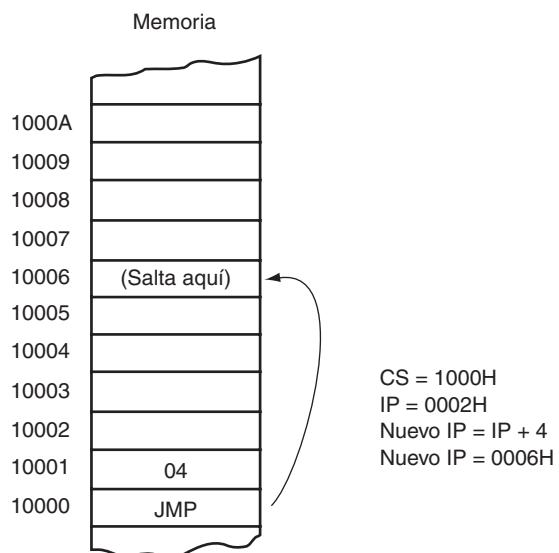


FIGURA 6-2 Un salto corto hacia cuatro posiciones de memoria por debajo de la dirección de la siguiente instrucción.



de salto (JMP INICIO) también se ensambla como un salto corto. Si la dirección de la siguiente instrucción (0009H) se suma al desplazamiento con signo extendido (0017H) del primer salto, la dirección de SIGUIENTE se encuentra en la posición 0017H + 0009H, o 0020H.

EJEMPLO 6-1

0000 33 DB		XOR	BX, BX
0002 B8 0001	INICIO:	MOV	AX, 1
0005 03 C3		ADD	AX, BX
0007 EB 17		JMP	SHORT SIGUIENTE

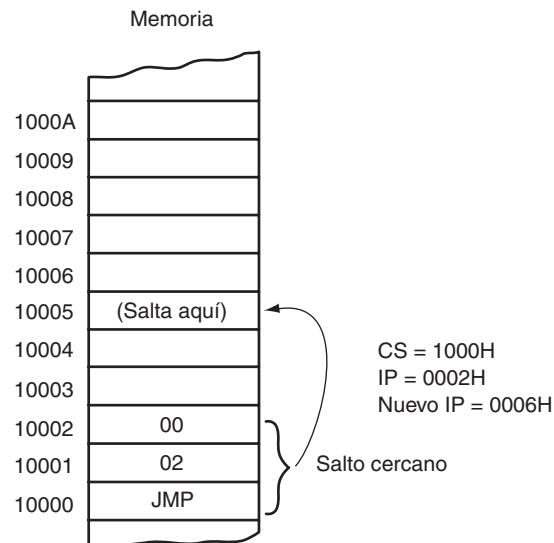
<posiciones de memoria que se omiten>

0020 8B D8	SIGUIENTE:	MOV	BX, AX
0022 EB DE		JMP	INICIO

Siempre que una instrucción de salto hace referencia a una dirección, por lo general hay una etiqueta que identifica a la dirección. La instrucción JMP SIGUIENTE es un ejemplo; salta a la etiqueta SIGUIENTE para la siguiente instrucción. Es muy raro que se utilice una dirección hexadecimal en una instrucción de salto, pero el ensamblador soporta el direccionamiento en relación con el apuntador de instrucciones mediante el uso del desplazamiento \$+a. Por ejemplo, la instrucción \$+2 salta sobre las dos siguientes posiciones de memoria (bytes) que siguen después de la instrucción JMP. La etiqueta SIGUIENTE debe ir seguida del signo de dos puntos (SIGUIENTE:) para permitir que una instrucción haga referencia a ella en un salto. Si no se coloca el signo de dos puntos después de una etiqueta, no se puede saltar hacia ella. Tenga en cuenta que la única vez que se utiliza el signo de dos puntos después de una etiqueta es cuando ésta se utiliza en una instrucción de salto o llamada. Esto también se aplica en Visual C++.

Salto cercano. El salto cercano es similar al salto corto, sólo que la distancia es mayor. Un salto cercano pasa el control a una instrucción en el segmento de código actual que esté ubicada dentro del rango de ± 32 Kbytes desde la instrucción de salto cercano. La distancia es de ± 2 Gbytes en los microprocesadores 80386 y superiores, cuando se operan en modo protegido. El salto cercano es una instrucción de tres bytes que contiene un código de operación seguido de un desplazamiento de 16 bits con signo. En los microprocesadores del 80386 al Pentium 4, el desplazamiento es de 32 bits y el salto cercano es de cinco bytes. El desplazamiento con signo se suma al apuntador de instrucciones (IP) para generar la

FIGURA 6-3 Un salto cercano que suma el desplazamiento (0002H) al contenido de IP.



dirección de salto. Como el desplazamiento con signo se encuentra dentro del rango de ± 32 Kbytes, un salto cercano puede saltar hacia cualquier posición de memoria dentro del segmento de código actual en modo real. En los microprocesadores 80386 y superiores, el segmento de código en modo protegido puede ser de 4 Gbytes, por lo que el desplazamiento de 32 bits permite un salto cercano hacia cualquier posición dentro del rango de ± 2 Gbytes. La figura 6-3 ilustra la operación de la instrucción de salto cercano en modo real.

El salto cercano también puede reubicarse (al igual que el salto corto) ya que también es un salto relativo. Si el segmento de código se mueve hacia una nueva posición en la memoria, la distancia entre la instrucción de salto y la dirección del operando permanece igual. Esto permite la reubicación de un segmento de código con sólo moverlo. Esta característica, junto con los segmentos de datos reubicables, hace a la familia de microprocesadores Intel ideal para su uso en un sistema computacional de propósito general. Puede escribirse software y cargarse en cualquier parte de la memoria, y puede funcionar sin modificaciones debido a los saltos relativos y los segmentos de datos reubicables.

El ejemplo 6-2 muestra el mismo programa básico que apareció en el ejemplo 6-1, sólo que la distancia de salto es mayor. El primer salto (JMP SIGUIENTE) pasa el control a la instrucción que está en la posición de memoria de desplazamiento 0200H dentro del segmento de código. Observe que la instrucción se ensambla como E9 0200 R. La letra R denota una dirección de **salto reubicable** de 0200H. La dirección reubicable de 0200H es para uso interno del programa ensamblador solamente. La verdadera instrucción en lenguaje máquina se ensambla como E9 F6 01, la cual no aparece en el listado en lenguaje ensamblador. El desplazamiento real para esta instrucción de salto es de 01F6H. El ensamblador lista la dirección de salto como 0200 R, por lo que es más fácil de interpretar a medida que se desarrolla el software. Si el archivo de ejecución enlazado (.EXE) o el archivo de comandos (.COM) se muestra en código hexadecimal, la instrucción de salto aparece como E9 F6 01.

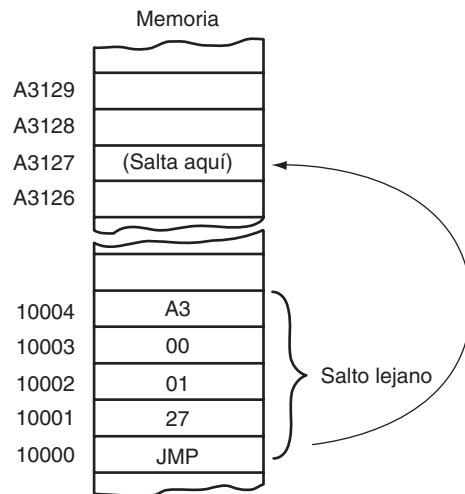
EJEMPLO 6-2

0000 33DB		XOR	BX, BX
0002 B8 0001	INICIO:	MOV	AX, 1
0005 03 C3		ADD	AX, BX
0007 E9 0200 R		JMP	SIGUIENTE

<posiciones de memoria que se omiten>

0200 8B D8	SIGUIENTE:	MOV	BX, AX
0202 E9 0002 R		JMP	INICIO

FIGURA 6-4 Una instrucción de salto lejano sustituye el contenido de CS y de IP con cuatro bytes que se colocan después del código de operación.



Salto lejano. Una instrucción de salto lejano (vea la figura 6-4) obtiene un nuevo segmento y una nueva dirección de desplazamiento para realizar el salto. Los bytes 2 y 3 de esta instrucción de cinco bytes contienen la nueva dirección de desplazamiento; los bytes 4 y 5 contienen la nueva dirección de segmento. Si el microprocesador (del 80286 al Pentium 4) se opera en modo protegido, la dirección de segmento accede a un descriptor que contiene la dirección base del segmento de salto lejano. La dirección de desplazamiento, que puede ser de 16 o de 32 bits, contiene la dirección de desplazamiento dentro del nuevo segmento de código.

El ejemplo 6-3 lista un programa corto que utiliza una instrucción de salto lejano. Esta instrucción aparece algunas veces con la directiva FAR PTR, como se muestra. Otra manera de obtener un salto lejano es mediante la definición de una etiqueta como **etiqueta lejana**. Una etiqueta es lejana solamente si es externa al segmento de código actual o procedimiento. La instrucción JMP ARRIBA en el ejemplo hace referencia a una etiqueta lejana. La etiqueta ARRIBA se define como una etiqueta lejana mediante la directiva EXTERN ARRIBA:FAR. Las **etiquetas externas** aparecen en programas que contienen más de un archivo de programa. Otra manera de definir una etiqueta como global es mediante el uso de un signo doble de dos puntos (ETIQUETA:::) después de la etiqueta, en lugar del signo de punto y coma. Esto se requiere dentro de bloques de procedimientos que se definen como cercanos si la etiqueta se utiliza desde el exterior del bloque de procedimientos.

Cuando se unen los archivos de programa, el enlazador inserta la dirección de la etiqueta ARRIBA en la instrucción JMP ARRIBA. También inserta la dirección del segmento en la instrucción JMP INICIO. La dirección de segmento en JMP FAR PTR INICIO se lista como - - - R cuando es reubicable; la dirección de segmento en JMP ARRIBA se lista como - - - E cuando es externa. En ambos casos, los - - - se llenan mediante el enlazador cuando éste enlaza o une los archivos de programa.

EJEMPLO 6-3

			EXTRN	ARRIBA:FAR
0000	33	DB		
0002	B8	0001	INICIO:	XOR BX,BX
0005	E9	0200 R		ADD AX,1
				JMP SIGUIENTE
<posiciones de memoria que se omiten>				
0200	8B	D8	SIGUIENTE:	MOV BX,AX
0202	EA	0002 - - - R		JMP FAR PTR INICIO
0207	EA	0000 - - - R		JMP ARRIBA

Saltos con registros como operandos. La instrucción de salto también puede utilizar un registro de 16 o de 32 bits como operando. Esto establece automáticamente la instrucción como un **salto indirecto**.

La dirección del salto se encuentra en el registro especificado por la instrucción de salto. A diferencia del desplazamiento asociado con el salto cercano, el contenido del registro se transfiere directamente hacia el apuntador de instrucciones. Un salto indirecto no se suma al apuntador de instrucciones, como los saltos cortos y cercanos. Por ejemplo, la instrucción JMP AX copia el contenido del registro AX en el registro IP cuando ocurre el salto. Esto permite un salto hacia cualquier posición dentro del segmento de código actual. En los microprocesadores 80386 y superiores, una instrucción JMP EAX también salta hacia cualquier posición dentro del segmento de código actual; la diferencia es que en modo protegido el segmento de código puede ser de 4 Gbytes, por lo que se necesita una dirección de desplazamiento de 32 bits.

El ejemplo 6-4 muestra cómo la instrucción JMP AX accede a una tabla de saltos en el segmento de código. Este programa de DOS lee una tecla del teclado y después modifica el código ASCII al colocar un 00H en AL para un ‘1’, un 01H para un ‘2’ y un 02H para un ‘3’. Si se escribe un ‘1’, un ‘2’ o un ‘3’, AH cambia a 00H. Como la tabla de saltos contiene direcciones de desplazamiento de 16 bits, el contenido de AX se duplica a 0, 2 o 4, de manera que pueda accederse a una entrada de 16 bits en la tabla. Después se carga en SI la dirección de desplazamiento del inicio de la tabla de saltos, y se suma AX para formar la referencia a la dirección de salto. A continuación, la instrucción MOV AX,[SI] obtiene una dirección de la tabla de saltos para que la instrucción JMP AX salte hacia las direcciones (UNO, DOS o TRES) almacenadas en la tabla.

EJEMPLO 6-4

```
;Instrucciones que leen 1, 2, o 3 del teclado.
;El número se muestra como 1, 2 o 3 mediante una tabla de
;saltos
;
.MODEL SMALL ;selecciona el modelo SMALL
. DATA ;inicio del segmento de datos
TABLA: DW UNO ;tabla de saltos
        DW DOS
        DW TRES
;
.CODE ;inicio del segmento de código
.STARTUP ;inicio del programa
PSUP: MOV AH,1 ;introduce en AL la tecla leída
        INT 21H
        SUB AL,31 ;convierte a BCD
        JB PSUP ;si la tecla < 1
        CMP AL,2
        JA PSUP ;si la tecla > 3
        MOV AH,0 ;duplica el código de la tecla
        ADD AX,AX
        ADD AX,AX
        MOV SI,OFFSET TABLA ;direcciona TABLA
        ADD SI,AX ;forma la dirección de búsqueda
        MOV AX,[SI] ;obtiene UNO, DOS o TRES
        JMP AX ;salta a UNO, DOS o TRES
        UNO: MOV DL,'1' ;obtiene el 1 ASCII
        JMP PINF
        DOS: MOV DL,'2' ;obtiene el 2 ASCII
        JMP PINF
        TRES: MOV DL,'3' ;obtiene el 3 ASCII
        PINF: MOV AH,2 ;muestra el número
        INT 21H
.EXIT
END
```

Saltos indirectos mediante un índice. La instrucción de salto también puede usar la forma [] de direccionamiento para acceder directamente a la tabla de saltos. Esta tabla puede contener direcciones de desplazamiento para saltos cercanos indirectos, o direcciones de segmento y desplazamiento para saltos lejanos indirectos. (Este tipo de salto también se conoce como salto **doble indirecto** si al salto de registro se le llama salto indirecto.) El ensamblador supone que el salto es cercano, a menos que la directiva FAR PTR indique una instrucción de salto lejano. Aquí, el ejemplo 6-5 repite el ejemplo 6-4 mediante el uso de JMP TABLA [SI] en vez de JMP AX. Esto reduce la longitud del programa.

EJEMPLO 6-5

```

;Instrucciones que leen 1, 2, o 3 del teclado.
;El número se muestra como 1, 2 o 3 mediante una tabla de
;saltos
;
;           .MODEL SMALL          ;selecciona el modelo SMALL
0000      .DATA             ;inicio del segmento de datos
0000 002D R    TABLA: DW    UNO   ;tabla de saltos
0002 0031 R    DW    DOS
0004 0035 R    DW    TRES
0000      .CODE             ;inicio del segmento de código
0017 B4 01     .STARTUP        ;introduce en AL la tecla leída
0019 CD 21     PSUP: MOV AH,1
001B 2C 31     INT 21H
001D 72 F9     SUB AL,31      ;convierte a BCD
001F 32 02     JB PSUP        ;si la tecla < 1
0021 77 F4     CMP AL,2
0023 B4 00     JA PSUP        ;si la tecla > 3
0025 03 C0     MOV AH,0
0027 B5 F0     ADD AX,AX      ;duplica el código de la tecla
0029 FF A4 0000 R    MOV SI,AX      ;forma la dirección de búsqueda
002D B2 31     JMP TABLA[SI]  ;salta a UNO, DOS o TRES
002E EB 06     UNO: MOV DL,'1'    ;obtiene el 1 ASCII
002F EB 06     JMP PINF       ;obtiene el 2 ASCII
0031 B2 32     DOS: MOV DL,'2'
0033 EB 02     JMP PINF       ;obtiene el 3 ASCII
0035 B2 33     TRES: MOV DL,'3'
0037 B4 02     PINF: MOV AH,2    ;muestra el número
0039 CD 21     INT 21H
0040          .EXIT
0041          END

```

El mecanismo utilizado para acceder a la tabla de saltos es idéntico, con una referencia de memoria normal. La instrucción JMP TABLA [SI] apunta a una dirección de salto almacenada en la posición de desplazamiento del segmento de código direccionada por SI. Salta a la dirección almacenada en la memoria en esta posición. Las instrucciones de salto indizado indirecto y de registro direccionan por lo general un desplazamiento de 16 bits. Esto significa que ambos tipos de saltos son cercanos. Si en un programa aparece una instrucción JMP FAR PTR[SI] o JMP TABLA[SI], y los datos de TABLA se definen mediante la directiva DD, el microprocesador supone que la tabla de saltos contiene direcciones de 32 bits tipo doble palabra (IP y CS).

Saltos condicionales y ajustes condicionales

Las instrucciones de salto condicional siempre son saltos cortos en los microprocesadores del 8086 al 80286. Esto limita el rango de salto a un valor entre +127 bytes y -127 bytes, desde la posición que sigue al salto condicional. En los microprocesadores 80386 y superiores, los saltos condicionales pueden ser cortos o cercanos. Gracias a ello, estos microprocesadores pueden utilizar un salto condicional hacia cualquier posición dentro del segmento de código actual. La tabla 6-1 lista todas las instrucciones de salto condicional con sus condiciones de prueba. El ensamblador MASM versión 6.x ajusta en forma automática los saltos condicionales si la distancia es demasiado grande.

Las instrucciones de salto condicional evalúan los siguientes bits de bandera: signo (S), cero (Z), acarreo (C), paridad (P) y desbordamiento (0). Si la condición que se evalúa es verdadera, se produce una ramificación hacia la etiqueta asociada con la instrucción de salto. Si la condición es falsa, se ejecuta el siguiente paso secuencial en el programa. Por ejemplo, una instrucción JC realiza el salto si el bit de acarreo está activado.

La operación de la mayoría de las instrucciones de salto condicional es simple ya que por lo general sólo evalúan un bit de bandera, aunque algunas evalúan más de uno. Las comparaciones de magnitud relativa requieren de instrucciones de salto condicional más complicadas, las cuales evalúan más de un bit de bandera.

Como en la programación se utilizan números con signo y sin signo, y debido a que el orden de estos números es distinto, existen dos conjuntos de instrucciones de salto condicional para las comparaciones de magnitud. La figura 6-5 muestra el orden de los números de 8 bits con y sin signo. Los

TABLA 6-1 Instrucciones de salto condicional.

Lenguaje ensamblador	Condición a evaluar	Operación
JA	Z = 0 y C = 0	Salta si está por encima.
JAE	C = 0	Salta si está por encima o si es igual.
JB	C = 1	Salta si está por debajo.
JBE	Z = 1 o C = 1	Salta si está por debajo o si es igual.
JC	C = 1	Salta si hay acarreo.
JE o JZ	Z = 1	Salta si es igual o salta si es cero.
JG	Z = 0 y S = 0	Salta si es mayor que.
JGE	S = 0	Salta si es mayor o igual que.
JL	S != 0	Salta si es menor que.
JLE	Z = 1 o S != 0	Salta si es menor o igual que.
JNC	C = 0	Salta si no hay acarreo.
JNE o JNZ	Z = 0	Salta si no es igual o salta si no es cero.
JNO	O = 0	Salta si no hay desbordamiento.
JNS	S = 0	Salta si no hay signo (positivo).
JNP o JPO	P = 0	Salta si no hay paridad o si la paridad es impar.
JO	O = 1	Salta si hay desbordamiento.
JP o JPE	P = 1	Salta si hay paridad o si la paridad es par.
JS	S = 1	Salta si hay signo (negativo).
JCXZ	CX = 0	Salta si CX es igual a cero.
JECXZ	ECX = 0	Salta si ECX es igual a cero.

números de 16 y de 32 bits siguen el mismo orden que los números de 8 bits, sólo que son más grandes. Observe que un FFH (255) está por encima del 00H en el conjunto de números sin signo, pero un FFH (-1) es menor que 00H para los números con signo. Por lo tanto, un FFH sin signo está por encima de 00H, pero un FFH con signo es menor que 00H.

Cuando se comparan números con signo se utilizan las instrucciones JG, JL, JGE, JLE, JE y JNE. Los términos *mayor que* y *menor que* se refieren a los números con signo. Cuando se comparan números sin signo se utilizan las instrucciones JA, JB, JAB, JBE, JE y JNE. Los términos *por encima de* y *por debajo de* se refieren a los números sin signo.

El resto de los saltos condicionales evalúan bits de bandera individuales, como los de desbordamiento y de paridad. Observe que JE tiene un código de operación alternativo: JZ. Todas las instrucciones tienen códigos de operación alternativos, pero muchos no se utilizan en programación ya que por lo general no se adaptan a la condición que se está evaluando. (En el apéndice B aparecen los códigos

FIGURA 6-5 Los números con signo y sin signo siguen distintos órdenes.

Números sin signo		Números con signo	
255	FFH	+127	7FH
254	FEH	+126	7EH
132	84H	+2	02H
131	83H	+1	01H
130	82H	+0	00H
129	81H	-1	FFH
128	80H	-2	FEH
4	04H	-124	84H
3	03H	-125	83H
2	02H	-126	82H
1	01H	-127	81H
0	00H	-128	80H

de operación alternativos, con el listado del conjunto de instrucciones.) Por ejemplo, la instrucción JA (salta si está por encima) tiene la instrucción alternativa JNBE (salta si no está por debajo o si es igual). Una instrucción JA funciona de manera idéntica a una instrucción JNBE, sólo que esta última es inadecuada en muchos casos, si se le compara con JA.

Todas las instrucciones de salto condicional evalúan bits de bandera, excepto JCXZ (salta si CX = 0) y JECXZ (salta si ECX = 0). En vez de evaluar bits de bandera, JCXZ evalúa directamente el contenido del registro CX sin afectar a los bits de bandera, y JECXZ evalúa de manera directa el contenido del registro ECX. En la instrucción JCXZ, si CX = 0 se produce un salto, y si CX != 0 no se produce ningún salto. De igual forma en la instrucción JECXZ, si ECX = 0 se produce un salto; si CX != 0 no se produce ningún salto.

En el ejemplo 6-6 aparece un programa que utiliza la instrucción JCXZ. Aquí, la instrucción SCASB busca el valor 0AH en una tabla. Después de la búsqueda, una instrucción JCXZ evalúa CX para ver si el contador ha llegado a cero. Si es cero, esto significa que el valor 0AH no se encuentra en la tabla. La bandera de acarreo se utiliza en este ejemplo para pasar la condición de no encontrado de regreso al programa que hizo la llamada. La instrucción JNE es otro método que se utiliza para evaluar si los datos se encontraron o no. Si JNE sustituye a JCXZ, realiza la misma función. Después de que se ejecuta la instrucción SCASB, las banderas indican una condición desigual si los datos no se encontraron en la tabla.

EJEMPLO 6-6

```
;Instrucciones que buscan el valor 0AH en una tabla de 100H bytes
;Se supone que la dirección de desplazamiento de TABLA está en SI
;
0017 B9 0064      MOV   CX,100       ;carga el contador
001A B0 0A         MOV   AL,0AH       ;carga AL con 0AH
001C FC            CLD              ;autoincremento
001D F2/AE         REPNE SCASB    ;busca el valor 0AH
001F F9            STC              ;activa el acarreo si se encuentra
0020 E3 01         JCXZ NO_ENCONTRADO ;si no se encuentra
0022 NO_ENCONTRADO
```

Las instrucciones de ajuste condicional. Además de las instrucciones de salto condicional, los microprocesadores del 80386 al Pentium 4 también contienen instrucciones de ajuste condicional. Estas instrucciones ponen en funcionamiento las condiciones que evalúan los saltos condicionales. Las instrucciones de ajuste condicional activan un byte con 01H o borran un byte con 00H, dependiendo del resultado de la condición a evaluar. La tabla 6-2 lista las formas disponibles de las instrucciones de ajuste condicional.

Estas instrucciones son útiles cuando una condición debe evaluarse en un punto que está mucho después en un programa. Por ejemplo, un byte puede ajustarse para indicar que el acarreo se borra en cierto punto en el programa mediante el uso de la instrucción SETNC MEM. Esta instrucción coloca un 01H en la posición de memoria MEM si el acarreo es cero, y coloca un 00H en MEM si el acarreo es 1. El contenido de MEM puede evaluarse en un punto posterior en el programa para determinar si el acarreo era cero en el punto en el que se ejecutó la instrucción SETNC MEM.

LOOP

La instrucción LOOP es una combinación de un decremento de CX y el salto condicional JNZ. En los procesadores del 8086 al 80286, LOOP decremente a CX; si CX != 0, salta a la dirección indicada por la etiqueta. Si CX se hace 0, se ejecuta la siguiente instrucción secuencial. En los microprocesadores 80386 y superiores, LOOP decremente a CX o a ECX, dependiendo del modo de instrucciones. Si los microprocesadores 80386 y superiores operan en el modo de instrucciones de 16 bits, LOOP utiliza CX; si operan en el modo de 32 bits, LOOP usa ECX. Este valor predeterminado se cambia mediante las instrucciones LOOPW (que utiliza CX) y LOOPD (que usa ECX) en los microprocesadores del 80386 al Pentium 4.

El ejemplo 6-7 muestra cómo los datos en un bloque de memoria (BLOQUE1) se suman a los datos en un segundo bloque de memoria (BLOQUE2), mediante el uso de LOOP para controlar cuántos

TABLA 6-2 Instrucciones de ajuste condicional.

Lenguaje ensamblador	Condición a evaluar	Operación
SETA	Z = 0 y C = 0	Se activa si está por encima.
SETAE	C = 0	Se activa si está por encima o si es igual.
SETB	C = 1	Se activa si está por debajo.
SETBE	Z = 1 o C = 1	Se activa si está por debajo o si es igual.
SETC	C = 1	Se activa si hay acarreo.
SETE o SETZ	Z = 1	Se activa si es igual o cero.
SETG	Z = 0 y S = 0	Se activa si es mayor que.
SETGE	S = 0	Se activa si es mayor o igual que.
SETL	S != 0	Se activa si es menor que.
SETLE	Z = 1 o S != 0	Se activa si es menor o igual que.
SETNC	C = 0	Se activa si no hay acarreo.
SETNE o SETNZ	Z = 0	Se activa si no es igual o si no es cero.
SETNO	O = 0	Se activa si no hay desbordamiento.
SETNS	S = 0	Se activa si no hay signo (positivo).
SETNP o SETPO	P = 0	Se activa si no hay paridad o si la paridad es impar.
SETO	O = 1	Se activa si hay desbordamiento.
SETP o SETPE	P = 1	Se activa si hay paridad o si la paridad es par.
SETS	S = 1	Se activa si hay signo (negativo).

números se van a sumar. Las instrucciones LODSW y STOSW acceden a los datos en BLOQUE1 y BLOQUE2. La instrucción ADD AX,ES:[DI] accede a los datos en el BLOQUE2, que se ubica en el segmento extra. La única razón por la que BLOQUE2 está en el segmento extra es que DI direcciona los datos del segmento extra para la instrucción STOSW. La directiva .STARTUP sólo carga DS con la dirección del segmento de datos. En este ejemplo, el segmento extra también direcciona datos en el segmento de datos, por lo que el contenido de DS se copia a ES por medio del acumulador. Por desgracia no existe una instrucción de movimiento directo de registro de segmento a registro de segmento.

EJEMPLO 6-7

```
;Un programa que suma el contenido de BLOQUE1 y BLOQUE2
;y almacena los resultados por encima de los datos en el
;BLOQUE2.
;
;      .MODEL SMALL                      ;selecciona el modelo SMALL
0000          .DATA                         ;inicio del segmento de datos
0000 0064  [        BLOQUE1 DW    100 DUP(?)   ;100 palabras para BLOQUE1
0000
]
00C8 0064  [        BLOQUE2 DW    100 DUP(?)   ;100 palabras para BLOQUE2
0000
]
0000          .CODE                         ;inicio del segmento de código
          .STARTUP                      ;inicio del programa
0017 8C D8          MOV    AX,DS           ;traslapa DS y ES
0019 8E C0          MOV    ES,AX
001B FC            CLD
001C B9 0064        MOV    CX,100          ;carga el contador
001F BE 0000 R      MOV    SI,OFFSET BLOQUE1 ;direcciona el BLOQUE1
0022 BF 00C8 R      MOV    DI,OFFSET BLOQUE2 ;direcciona el BLOQUE2
0025 AD            L1:    LODSW             ;carga AX con el BLOQUE1
0026 26:03 05       ADD    AX,ES:[DI]       ;suma el BLOQUE2
0029 AB            STOSW             ;almacena la respuesta
002A E2 F9          LOOP   L1              ;repite 100 veces
          .EXIT
          END
```

Instrucción LOOP condicional. Al igual que REP, la instrucción LOOP también tiene formas condicionales: LOOPE y LOOPNE. La instrucción LOOPE (**ejecuta ciclo mientras sea igual**) salta si CX != 0 mientras exista una condición de igualdad. Se sale del ciclo si la condición no es igual o si el registro CX se decrementa hasta llegar a 0. La instrucción LOOPNE (**ejecuta ciclo mientras no sea igual**) salta si CX != 0 mientras exista una condición desigual. Se sale del ciclo si la condición es igual o si el registro CX se decrementa hasta llegar a cero. En los microprocesadores del 80386 al Pentium 4, la instrucción LOOP condicional puede utilizar a CX o a ECX como el contador. Las instrucciones LOOPEW/LOOPED o LOOPNEW/LOOPNED redefinen el modo de instrucciones en caso de ser necesario.

Al igual que con las instrucciones de repetición condicional, existen instrucciones alternativas para LOOPE y LOOPNE. La instrucción LOOPE es igual que LOOPZ, y la instrucción LOOPNE es igual que LOOPNZ. En la mayoría de los programas sólo se aplican LOOPE y LOOPNE.

6-2

CONTROL DEL FLUJO DEL PROGRAMA

Es mucho más fácil usar las instrucciones en lenguaje ensamblador .IF, .ELSE, .ELSEIF y .ENDIF para controlar el flujo del programa que utilizar la instrucción de salto condicional correcta. Estas instrucciones siempre indican un comando especial en lenguaje ensamblador para MASM. Observe que las instrucciones en lenguaje ensamblador de control de flujo que empiezan con un punto sólo están disponibles para MASM versión 6.xx y no para las versiones anteriores del ensamblador, como la 5.10. Otras instrucciones que se desarrollarán en este capítulo son .REPEAT-.UNTIL y .WHILE-.END. Estas instrucciones (**los comandos punto**) no funcionan si se utiliza el ensamblador en línea de Visual C++.

El ejemplo 6-8(a) muestra cómo se utilizan estas instrucciones para controlar el flujo de un programa, al evaluar AL para ver si contiene las letras ASCII de la A a la F. Si el contenido de AL está entre estas letras, se resta 7 a AL.

Para realizar la misma tarea mediante el uso del ensamblador en línea de Visual C++ se requiere por lo general de C++ en vez de lenguaje ensamblador. El ejemplo 6-8(b) muestra la misma tarea mediante el uso del ensamblador en línea de Visual C++ y los saltos condicionales en lenguaje ensamblador. También muestra cómo utilizar una etiqueta en un bloque de ensamblador en Visual C++. Esto nos muestra que es más difícil realizar la misma tarea sin los comandos punto. Nunca debe utilizar mayúsculas para los comandos en lenguaje ensamblador con el ensamblador en línea, ya que algunos de ellos están reservados por C++ y ocasionarán problemas.

EJEMPLO 6-8(A)

```
.IF AL >= 'A' && AL <= 'F'
    SUB     AL, 7
.ENDIF
    SUB     AL, 30H
```

EJEMPLO 6-8(B)

```
char temp;
_asm{
    mov     al,temp
    cmp     al,41h
    jb      Despues
    cmp     al,46h
    ja      Despues
    sub     al,7
Despues:
    sub     al,30h
    mov     temp,al
}
```

TABLA 6-3 Los operadores relacionales utilizados con la instrucción .IF en lenguaje ensamblador.

Lenguaje ensamblador	Función
==	Igual o lo mismo que
!=	Desigual
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
&	Prueba de bit
!	Inversión lógica
&&	AND lógico
	OR lógico
	OR

En el ejemplo 6-8(a) observe cómo el símbolo && representa la función AND en la instrucción .IF. No hay un .IF en el ejemplo 6-8(b) ya que la misma operación se realiza mediante el uso de unas cuantas instrucciones de comparación (CMP) para realizar la misma tarea. En la tabla 6-3 podrá consultar una lista completa de los operadores relacionales que se utilizan con la instrucción .IF. Observe que muchas de estas condiciones (como &&) también se utilizan en muchos lenguajes de alto nivel, como C/C++.

El ejemplo 6-9 muestra otro ejemplo de la directiva .IF condicional que convierte todas las letras en código ASCII a mayúsculas. Primero se lee el teclado sin eco mediante el uso de la función 06H de la instrucción INT 21H del DOS y después la instrucción .IF convierte el carácter en mayúsculas, si es necesario. En este ejemplo se utiliza la función lógica AND (&&) para determinar si el carácter está en minúsculas. Si está en minúsculas se resta un 20H, con lo que se convierte a mayúsculas. Este programa lee una tecla desde el teclado y la convierte en mayúsculas antes de mostrarla. Observe también cómo el programa termina cuando se escribe la tecla de control C (ASCII = 03H). La directiva .LISTALL hace que se listen todas las instrucciones generadas por el ensamblador, incluyendo la etiqueta @Startup generada por la directiva .STARTUP. La directiva .EXIT también se expande mediante .LISTALL para mostrar el uso de la función 4CH de la instrucción INT 21H del DOS, la cual devuelve el control al DOS.

EJEMPLO 6-9

```

;Un programa de DOS que lee el teclado y convierte todos
;los datos en minúsculas a mayúsculas antes de mostrarlos.
;
;Este programa se termina con control-C
;
; .MODEL TINY           ;selecciona el modelo diminuto
; .LISTALL              ;lista todas las instrucciones
0000          .CODE            ;inicio del segmento de código
                    .STARTUP        ;inicio del programa
0100          * @Startup
0100  B4  06    MAIN1:   MOV     AH,6      ;lee tecla sin eco
0102  B2  FF    MOV     DL,0FFH
0104  CD  21    INT     21H
0106  74  F8    JE      MAIN1        ;si no hay tecla
0108  3C  03    CMP     AL,3      ;evalúa control-C
010A  74  10    JE      MAIN2        ;si se escribe control-C
010C  3C  61    *                   cmp     al,'a'
010E  72  06    *                   jb     @C0001
0110  3C  7A    *                   cmp     al,'z'
0112  77  02    *                   ja     @C0001
0114  2C  29    SUB     AL,20H
0116  4C  00    .IF     AL >= 'a' && AL <= 'z'
0118  4C  00    .ENDIF
011A  4C  00    .EXIT

```

```

0116          *  @C0001:
0116  8A  D0          MOV      DL,AL      ;hace eco en carácter a mostrar
0118  CD  21          INT      21H
011A  EB  E4          JMP      MAIN1     ;repetición
011C          MAIN2:
011C          .EXIT
011C  B4  4C  *        MOV      AH,4CH
011E  CD  21  *        INT      21H
011F          END

```

En este programa, una letra en minúscula se convierte a mayúscula mediante el uso de la instrucción .IF AL \geq ‘a’ && AL \leq ‘z’. Si AL contiene un valor que sea mayor o igual que la letra a minúscula, y que sea menor o igual que la letra z minúscula (un valor entre a y z), se ejecutan la instrucción que está entre .IF y .ENDIF. Esta instrucción (SUB AL,20H) resta 20H de la letra minúscula para cambiarla a letra mayúscula. Observe cómo el programa ensamblador implementa la instrucción .IF (vea las líneas que empiezan con *). El ensamblador genera la etiqueta @C0001, para que luego la utilicen las instrucciones de salto condicional que se colocan en el programa mediante la instrucción .IF.

En el ejemplo 6-10 también se utiliza la instrucción .IF condicional. Este programa lee una tecla del teclado y luego la convierte en código hexadecimal. Este programa no se muestra en su forma expandida.

En este ejemplo, la instrucción .IF AL \geq ‘a’ && AL \leq ‘f’ hace que se ejecute la siguiente instrucción (SUB AL,57H) si AL contiene una letra entre la a y la f, para después convertirla en hexadecimal. Si AL no contiene una de esas letras, la siguiente instrucción .ELSEIF evalúa si contiene una letra entre la A y la F. Si esta condición se cumple, se resta un 37H de AL. Si ninguna condición se cumple, se resta un 30H de AL antes de almacenar su valor en la posición de memoria TEMP del segmento de datos. La misma conversión puede realizarse en una función en C++, como se muestra en el fragmento de un programa en el ejemplo 6-10(b).

EJEMPLO 6-10(a)

```

;Un programa de DOS que lee una tecla y almacena su valor
;hexadecimal
;en la posición de memoria TEMP
;
0000          .MODEL SMALL           ;selecciona el modelo pequeño
0000  00          .DATA             ;inicio del segmento de datos
0000          TEMP  DB   ?           ;define TEMP
0000          .CODE             ;inicio del segmento de código
0000          .STARTUP          ;inicio del programa
0017  B4  01          MOV    AH,1    ;lee el teclado
0019  CD  21          INT    21H
0023  2C  57          .IF    AL >= 'a'  && AL <= 'f'   ;si es minúscula
0023          SUB    AL,57H
002F  2C  37          .ELSEIF .IF AL >= 'A'  && AL <= 'F'   ;si es mayúscula
002F          SUB    AL,37H
0033  2C  30          .ELSE
0033          SUB    AL,30H   ;en cualquier otro caso
0035  A2  0000  R          .ENDIF
0035          MOV    TEMP,AL   ;se almacena en TEMP
0036          .EXIT
0037          END

```

EJEMPLO 6-10(b)

```

Char Convierte(char temp)
{
    if ( temp >= 'a'  &&  temp <= 'f' )
        temp -= 0x57;
    else if ( temp >= 'A'  &&  temp <= 'F' )
        temp -= 0x37;
    else
        temp -= 0x30;
    return temp;
}

```

Ciclos WHILE

Al igual que en la mayoría de los lenguajes de alto nivel, el ensamblador también cuenta con la instrucción de ciclo WHILE, disponible para MASM versión 6.x. La instrucción .WHILE se utiliza con una condición para empezar el ciclo, y la instrucción .ENDW termina el ciclo.

El ejemplo 6-11 muestra cómo se utiliza la instrucción .WHILE para leer datos desde el teclado y almacenarlos en un arreglo llamado BUF hasta que se oprime la tecla Intro (0DH). Este programa supone que BUF se almacena en el segmento extra, ya que se utiliza la instrucción STOSB para almacenar los datos del teclado en la memoria. Observe que la porción del programa correspondiente al ciclo .WHILE se muestra en formato expandido, para que puedan estudiarse las instrucciones que inserta el ensamblador (las cuales empiezan con *). Al oprimir la tecla Intro (0DH), a la cadena se le adjunta el signo \$ para que pueda mostrarse con la función número 9 de la instrucción INT 21H del DOS.

EJEMPLO 6-11

```
;Un programa de DOS que lee una cadena de caracteres
;desde el teclado y luego los vuelve a mostrar.
;
0000          .MODEL SMALL           ;selecciona el modelo pequeño
0000 0D 0A      .DATA              ;inicio del segmento de datos
0002 0100 [      MENS DB 13,10    ;retorno y salto de línea
0000          ]      BUF DB 256 DUP(?) ;búfer de la cadena de caracteres
00
0017 8C D8      .CODE              ;inicio del segmento de código
0019 8C C0      .STARTUP          ;inicio del programa
001B FC          MOV AX,DX        ;traslape DS con ES
001C BF 0002 R   MOV ES,AX
001F EB 05      CLD               ;selecciona autoincremento
0021 B4 01      MOV DI,OFFSET BUF ;direcciona el búfer
0023 CD 21      .WHILE AL != 0DH ;hace ciclo mientras no se oprima
0025 AA          INT 21H           Intro
0026 3C 0D      *     jmp @C0001
0028 75 F7      *     @C0002:
002A C6 45 FF 24  MOV AH,1          ;lee tecla
002E BA 0000 R   INT 21H
002F AA          STOSB             ;almacena el código de la tecla
0030 AA          .ENDW
0026 3C 0D      *     @C0001:
0028 75 F7      *     cmp al,0dh
002A C6 45 FF 24  MOV BYTE PTR[DI-1],'&
002E BA 0000 R   MOV DX,OFFSET MENS
0031 B4 09      MOV AH,9
0033 CD 21      INT 21H           ;muestra MENS
0034 AA          .EXIT
0035 AA          END
```

El programa del ejemplo 6-11 funciona en forma perfecta, siempre y cuando AL contenga algún otro valor distinto de 0DH cuando lleguemos a la instrucción .WHILE. Para corregir esto, podemos agregar una instrucción MOV AL,0DH antes de la instrucción .WHILE. Aunque no se muestran en un ejemplo, las instrucciones .BREAK y .CONTINUE pueden usarse con el ciclo WHILE. A menudo la instrucción .BREAK va seguida de la instrucción .IF para seleccionar la condición con que se va a interrumpir el flujo del programa, como en .BREAK .IF AL == 0DH. La instrucción .CONTINUE, que puede usarse para permitir que el ciclo DO—.WHILE continúe si se cumple cierta condición, puede usarse con .BREAK. Por ejemplo, la instrucción .CONTINUE .IF AL == 15 permite que el ciclo continúe si AL es igual a 15. Observe que los comandos .BREAK y .CONTINUE funcionan de la misma forma en un programa en C++.

Ciclos REPEAT-UNTIL

La instrucción REPEAT-UNTIL también está disponible para el ensamblador. Se repite una serie de instrucciones hasta que ocurre una condición. La instrucción .REPEAT define el inicio del ciclo; el final se define con la instrucción .UNTIL, que contiene una condición. Estas instrucciones están disponibles para MASM versión 6.x.

Si el ejemplo 6-11 se rediseña de manera que se utilice la instrucción REPEAT-UNTIL, ésta parece ser la mejor solución. En el ejemplo 6-12 podrá ver el programa que lee teclas desde el teclado y almacena los datos en el arreglo BUF del segmento extra hasta que se oprime Intro. Este programa también llena el búfer con datos del teclado hasta que se oprime la tecla Intro (0DH). Al oprimir Intro, el programa muestra la cadena de caracteres mediante el uso de la función número 9 de la instrucción INT 21H del DOS, después de anexar el signo de dólar requerido a los datos del búfer. Observe cómo la instrucción .UNTIL AL == 0DH genera código (las instrucciones que empiezan con *) para evaluar si se oprime la tecla Intro.

EJEMPLO 6-12

```
;Un programa de DOS que lee una cadena de caracteres
;desde el
;teclado y luego los vuelve a mostrar.
;
0000          .MODEL SMALL           ;selecciona el modelo pequeño
0000  0D  0A    .DATA              ;inicio del segmento de datos
0002  0100  [      MENS   DB  13,10  ;retorno y salto de línea
                  BUF    DB  256 DUP(?);búfer de la cadena de caracteres
                  00
                ]
0000          .CODE               ;inicio del segmento de código
0000          .STARTUP            ;inicio del programa
0017  8C  D8    MOV   AX,DX      ;traslape DS con ES
0019  8C  C0    MOV   ES,AX
001B  FC          CLD
001C  BF  0002  R    MOV   DI,OFFSET BUF ;direcciona el búfer
0000          .REPEAT             ;se repite hasta que se oprima Intro
001F          *  @C0001:
001F  B4  01    MOV   AH,1        ;lee tecla
0021  CD  21    INT   21H
0023  AA          STOSB           ;almacena el código de la tecla
0000          .UNTIL   AL  ==  0DH
0025  3C  0D      *  cmp   al,0dh
0027  75  F7      *  jne   @C0001
0028  C6  45  FF  24  MOV   BYTE PTR[DI-1],&
002C  BA  0000  R    MOV   DX,OFFSET MENS
002E  B4  09    MOV   AH,9
0031  CD  21    INT   21H       ;muestra MENS
0000          .EXIT
0000          END
```

También hay una instrucción .UNTILCXZ disponible, la cual utiliza la instrucción LOOP para revisar el valor de CX para un ciclo REPEAT. La instrucción .UNTILCXZ utiliza el registro CX como contador para repetir un ciclo un número fijo de veces. El ejemplo 6-13 muestra una secuencia de instrucciones que utilizan la instrucción .UNTILCXZ para sumar el contenido del arreglo tipo byte UNO al arreglo tipo byte DOS. Las sumas se almacenan en el arreglo TRES. Cada arreglo contiene 100 bytes de datos, de manera que el ciclo se repite 100 veces. Este ejemplo supone que el arreglo TRES está en el segmento extra, y que los arreglos UNO y DOS están en el segmento de datos. Observe cómo se inserta la instrucción LOOP para .UNTILCXZ.

EJEMPLO 6-13

```

012C  B9  0064      MOV    CX,100          ;establece el contador
012F  BF  00C8  R   MOV    DI,OFFSET TRES  ;direcciona los arreglos
0132  BE  0000  R   MOV    SI,OFFSET UNO
0135  BB  0064  R   MOV    BX,OFFSET DOS

.REPEAT

0138      * @C0001:
0138  AC      LODSB
0139  02  07      ADD    AL,[BX]
013B  AA      STOSB
013C  43      INC    BX

.UNTILCXZ

013D  E2  F9      *      LOOP   @C0001

```

6-3**PROCEDIMIENTOS**

El procedimiento (subrutina o **función**) es una parte importante de la arquitectura de cualquier sistema computacional. Un procedimiento es un grupo de instrucciones que por lo general desempeñan una tarea. Un procedimiento es una sección reutilizable del software que se almacena en memoria una vez, pero se utiliza tantas veces como sea necesario. Esto ahorra espacio en memoria y facilita el desarrollo del software. La única desventaja de un procedimiento es que la computadora requiere de una pequeña cantidad de tiempo para enlazarse al procedimiento y regresar de él. La instrucción CALL enlaza el procedimiento y la instrucción RET (**retorno**) regresa del procedimiento.

La pila almacena la dirección de retorno siempre que se llama a un procedimiento durante la ejecución de un programa. La instrucción CALL mete en la pila la dirección de la instrucción que va después de CALL (**dirección de retorno**). La instrucción RET saca una dirección de la pila, para que el programa regrese a la instrucción que va después de CALL.

En el ensamblador hay reglas específicas para almacenar procedimientos. Un procedimiento empieza con la directiva PROC y termina con la directiva ENDP. Cada directiva aparece con el nombre del procedimiento. Esta estructura de programación facilita la localización del procedimiento en un listado de programa. La directiva PROC va seguida del tipo de procedimiento: NEAR o FAR. El ejemplo 6-14 muestra cómo el ensamblador utiliza la definición de un procedimiento cercano (NEAR, intrasegmento) y de un procedimiento lejano (FAR, intersegmento). En la versión 6.x de MASM, el tipo NEAR o FAR puede ir seguido de la instrucción USES. Esta instrucción permite que se meta cualquier número de registros en forma automática a la pila, y que se saquen de la pila dentro del procedimiento. En el ejemplo 6-14 también se muestra el uso de la instrucción USES.

EJEMPLO 6-14

```

0000      SUMAS     PROC NEAR
0000  03  C3      ADD  AX,BX
0002  03  C1      ADD  AX,CX
0004  03  C2      ADD  AX,DX
0006  C3          RET
0007      SUMAS     ENDP

0007      SUMAS1    PROC FAR
0007  03  C3      ADD  AX,BX
0009  03  C1      ADD  AX,CX
000B  03  C2      ADD  AX,DX
000D  CB          RET
000E      SUMAS1    ENDP

```

```

000E          SUMAS3    PROC NEAR USE  BX  CX  DX
0011  03  C3          ADD  AX,BX
0013  03  C1          ADD  AX,CX
0015  03  C2          ADD  AX,DX
                  RET
001B          SUMAS3    ENDP

```

Cuando se comparan los primeros dos procedimientos, la única diferencia es el código de operación de la instrucción de retorno. La instrucción de retorno del procedimiento NEAR utiliza el código de operación C3H y la dirección de retorno del procedimiento FAR utiliza el código de operación CBH. El retorno del procedimiento NEAR saca un número de 16 bits de la pila y lo coloca en el apuntador de instrucciones para regresar del procedimiento en el segmento de código actual. El retorno del procedimiento FAR saca un número de 32 bits de la pila y lo coloca en IP y CS para regresar desde el procedimiento a cualquier posición de memoria.

Los procedimientos que se van a utilizar en todo el software (**globales**) deben escribirse como procedimientos FAR. Los procedimientos que se utilizan en una tarea dada (**locales**) por lo general se definen como procedimientos NEAR. La mayoría de los procedimientos son NEAR.

CALL

La instrucción CALL transfiere el flujo del programa al procedimiento. Esta instrucción es distinta a la instrucción de salto, ya que CALL almacena una dirección de retorno en la pila. La dirección de retorno devuelve el control a la instrucción que sigue inmediatamente después de CALL en un programa en donde se ejecuta una instrucción RET.

CALL cercana. La instrucción CALL cercana es de tres bytes; el primer byte contiene el código de operación y los bytes segundo y tercero contienen el desplazamiento o distancia de ± 32 Kbytes en los microprocesadores del 8086 al 80286. Su formato es idéntico al de la instrucción de salto cercano. Los microprocesadores 80386 y superiores utilizan un desplazamiento de 32 bits cuando operan en modo protegido, lo que permite una distancia de ± 2 Gbytes. Cuando se ejecuta la instrucción CALL, primero mete en la pila la dirección de desplazamiento de la siguiente instrucción. La dirección de desplazamiento de la siguiente instrucción aparece en el apuntador de instrucciones (IP o EIP). Después de almacenar esta dirección de retorno, suma el desplazamiento de los bytes 2 y 3 al registro IP para transferir el control al procedimiento. No hay una instrucción CALL corta. Existe una variación en el código de operación llamada CALLN, pero debe evitarse. Es mejor utilizar la instrucción PROC y definir la instrucción CALL como cercana.

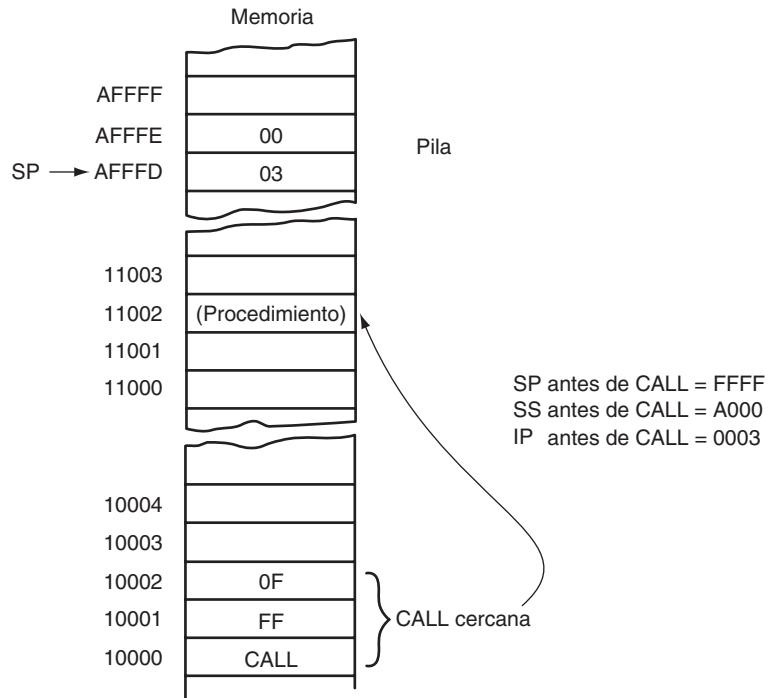
¿Por qué se almacena el registro IP o EIP en la pila? El apuntador de instrucciones siempre apunta a la siguiente instrucción en el programa. Para la instrucción CALL, el contenido de IP/EIP se mete en la pila, de manera que el control del programa pase a la instrucción que va después de CALL, una vez que termine el procedimiento. La figura 6-6 muestra la dirección de retorno (IP) almacenada en la pila y la llamada al procedimiento.

CALL lejana. La instrucción CALL lejana es como un salto lejano, ya que puede llamar a un procedimiento almacenado en cualquier posición de memoria en el sistema. La instrucción CALL lejana es de cinco bytes y contiene un código de operación que va seguido del siguiente valor para los registros IP y CS. Los bytes 2 y 3 almacenan el nuevo contenido del registro IP, y los bytes 4 y 5 almacenan el nuevo contenido de CS.

La instrucción CALL lejana coloca el contenido de IP y CS en la pila antes de saltar a la dirección indicada por los bytes 2-5 de la instrucción. Esto permite que la instrucción CALL lejana llame a un procedimiento ubicado en cualquier parte de la memoria y que regrese de ese procedimiento.

La figura 6-7 muestra cómo la instrucción CALL lejana llama a un procedimiento lejano. Aquí, el contenido de IP y CS se mete en la pila. A continuación, el programa se ramifica hacia el procedimiento. Existe una variación de la instrucción CALL lejana llamada CALLF, pero debe evitarse. Es mejor definir el tipo de la instrucción CALL mediante la instrucción PROC.

FIGURA 6-6 El efecto de una instrucción CALL cercana en la pila y en el apuntador de instrucciones.



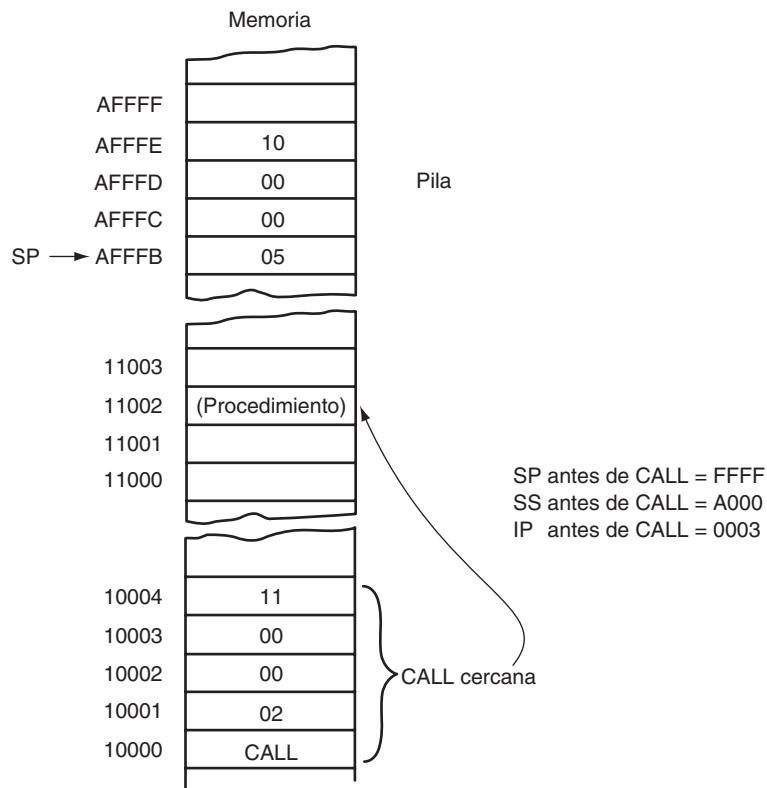
Instrucciones CALL con registros como operandos. Al igual que las instrucciones de salto, las instrucciones de llamada también pueden contener un registro como operando. Un ejemplo es la instrucción CALL BX, que mete el contenido de IP en la pila y después salta hacia la dirección de desplazamiento (ubicada en el registro BX) en el segmento de código actual. Este tipo de instrucción CALL siempre utiliza una dirección de desplazamiento de 16 bits, que se puede almacenar en cualquier registro de 16 bits excepto los registros de segmento.

El ejemplo 6-15 ilustra el uso de la instrucción CALL con registros para llamar a un procedimiento que empieza en la dirección de desplazamiento DESP. (Esta llamada también podría llamar al procedimiento en forma directa, mediante el uso de la instrucción CALL DESP.) La dirección OFFSET llamada DESP se coloca en el registro BX y después la instrucción CALL BX llama al procedimiento que empieza en la dirección DESP. Este programa muestra la cadena de caracteres “OK” en la pantalla del monitor.

EJEMPLO 6-15

```
;Un programa de DOS que muestra OK mediante el procedimiento
;DESP.
;
0000          .MODEL TINY           ;selecciona el modelo diminuto
;                                ;inicio del segmento de código
;                                ;inicio del programa
0100  BB  0110  R      MOV     BX,OFFSET DESP ;carga BX con el desplazamiento
;                                ;DESP
0103  B2  4F          MOV     DL, 'O'    ;muestra una O
0105  FF  D3          CALL    BX
0107  B2  4B          MOV     DL, 'K'    ;muestra una K
0109  FF  D3          CALL    BX
.EXIT
;
;Procedimiento que muestra el carácter ASCII que está en DL
;
```

FIGURA 6-7 El efecto de una instrucción CALL cercana.



```

0110      DESP    PROC    NEAR
0110  B4 02      MOV     AH, 2    ; selecciona la función 2
0112  CD 21      INT     21H    ; ejecuta la función 2 del DOS
0114  C3          RET
0115          DESP    ENDP
                  END
  
```

Instrucciones CALL con direcciones de memoria indirectas. Una instrucción CALL con una dirección de memoria indirecta es muy útil siempre que se necesitan seleccionar distintas subrutinas en un programa. Este proceso de selección se define a menudo mediante un número que direcciona la dirección a una instrucción CALL en una tabla de búsqueda. En esencia, esto es lo mismo que el salto indirecto en el que se utiliza una tabla de búsqueda para una dirección de salto, el cual vimos anteriormente en este capítulo.

El ejemplo 6-16 muestra cómo acceder a una tabla de direcciones mediante una instrucción CALL indirecta. La tabla que se muestra en el ejemplo contiene tres direcciones de subrutinas separadas, a las que se hace referencia mediante los números 0, 1 y 2. Este ejemplo utiliza el modo de direccionamiento de índice escalado para multiplicar el número en EBX por 2, de manera que acceda a la entrada correcta en la tabla de búsqueda.

EJEMPLO 6-16

```

; Instrucción que llama al procedimiento CERO, UNO o DOS
; dependiendo del valor que esté en EBX
;
TABLA DW      CERO      ;dirección del procedimiento CERO
                      DW      UNO       ;dirección del procedimiento UNO
                      DW      DOS       ;dirección del procedimiento DOS
;
CALL TABLA[2*EBX]
  
```

La instrucción CALL también puede hacer referencia a apunadores lejanos si la instrucción aparece como CALL FAR PTR [4*EBX] o como CALL TABLA[4*EBX], si los datos en la tabla se definen como datos tipo doble palabra mediante la directiva DD. Estas instrucciones recuperan una dirección de 32 bits (cuatro bytes) de la posición de memoria del segmento de datos direccionada por EBX y la utilizan como dirección de un procedimiento lejano.

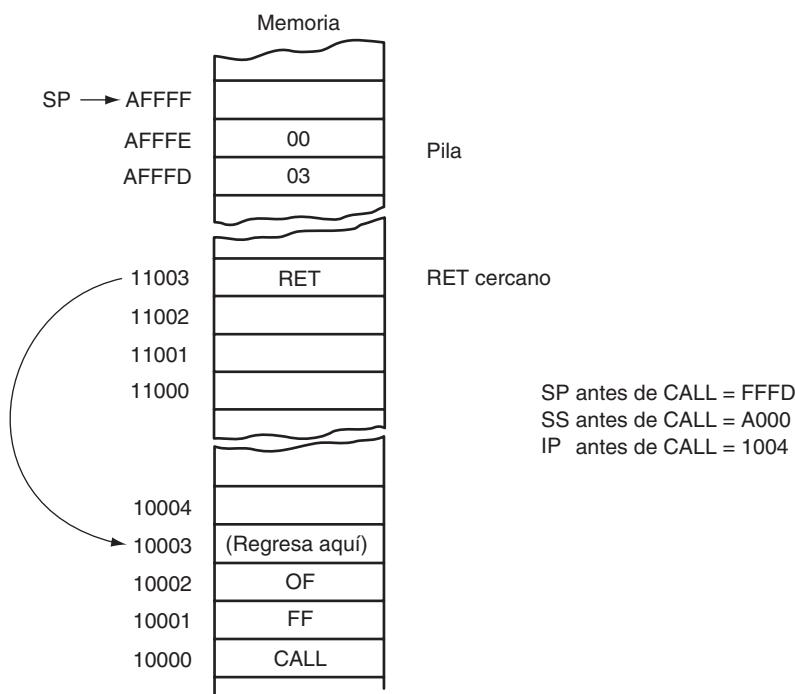
RET

La instrucción de retorno (RET) saca un número de 16 bits (**retorno cercano**) de la pila y la coloca en IP, o saca un número de 32 bits (**retorno lejano**) y lo coloca en IP y CS. Las instrucciones de retorno lejano y cercano se definen en la directiva PROC del procedimiento, que selecciona de manera automática la instrucción de retorno apropiada. Cuando los microprocesadores del 80386 al Pentium 4 operan en modo protegido, el retorno lejano saca seis bytes de la pila. Los primeros cuatro bytes contienen el nuevo valor para EIP y los últimos dos contienen el nuevo valor para CS. En los microprocesadores 80386 y superiores, un retorno cercano en modo protegido saca cuatro bytes de la pila y los coloca en EIP.

Cuando se modifican los registros IP/EIP o IP/EIP y CS, la dirección de la siguiente instrucción está en una nueva posición de memoria. Esta nueva posición es la dirección de la instrucción que sigue inmediatamente después de la instrucción CALL más reciente que llama a un procedimiento. La figura 6-8 muestra cómo se enlaza la instrucción CALL a un procedimiento y cómo regresa la instrucción RET en los microprocesadores del 8086 al Pentium 4 que operan en modo real.

Existe otra forma de la instrucción de retorno, la cual suma un número al contenido del apuntador de la pila (SP) después de sacar la dirección de retorno de la pila. Un retorno que utiliza un operando inmediato es ideal para usarse en un sistema que utilice las convenciones de llamadas de C/C++ o PASCAL. (Esto es cierto aún y cuando las convenciones de llamadas de C/C++ y PASCAL requieren que el procedimiento que hace la llamada saque los datos de la pila para muchas funciones.) Estas convenciones meten los parámetros en la pila antes de llamar a un procedimiento. Si los parámetros se van a descartar al momento de regresar, la instrucción de retorno contiene un número que representa el número de bytes que se meten en la pila como parámetros.

FIGURA 6-8 El efecto de una instrucción de retorno cercano sobre la pila y el apuntador de instrucciones.



El ejemplo 6-17 muestra cómo este tipo de retorno borra los datos que se colocan en la pila, mediante unas cuantas operaciones PUSH. La instrucción RET 4 suma un 4 a SP después de sacar la dirección de retorno de la pila. Como las instrucciones PUSH AX y PUSH BX en conjunto colocan cuatro bytes de datos en la pila, este retorno elimina en efecto AX y BX de la pila. Este tipo de retorno aparece pocas veces en los programas en lenguaje ensamblador, pero se utiliza en programas de alto nivel para borrar los datos de la pila después de un procedimiento. Observe cómo se direccionan los parámetros en la pila mediante el uso del registro BP, que de manera predeterminada direcciona el segmento de pila. La acción de apilar parámetros es común en los procedimientos escritos para C++ o PASCAL, mediante el uso de las convenciones llamadas de C++ o de PASCAL.

EJEMPLO 6-17

0000 B8 001E	MOV	AX, 30	
0003 BB 0028	MOV	BX, 40	
0006 50	PUSH	AX	;parámetro 1 de la pila
0007 53	PUSH	BX	;parámetro 2 de la pila
0008 E8 0066	CALL	SUMAM	;suma los parámetros de la pila
0071	SUMAM	PROC	NEAR
0071 55	PUSH	BP	;almacena BP
0072 8B EC	MOV	BP, SP	;direcciona la pila con BP
0074 8B 46 04	MOV	AX, [BP+4]	;obtiene el parámetro 1
0077 03 46 06	ADD	AX, [BP+6]	;suma el parámetro 2
007A 5D	POP	BP	;restaura BP
007B C2 0004	RET	4	;regresa, vacía parámetros
007E	SUMAM	ENDP	

Al igual que con las instrucciones CALLN y CALLF, también hay variaciones de la instrucción de retorno: RETN y RETF. De igual forma que con las instrucciones CALLN y CALLF, hay que evitar estas variaciones y utilizar mejor la instrucción PROC para definir el tipo de llamada y de retorno.

6-4

INTRODUCCIÓN A LAS INTERRUPCIONES

Una interrupción puede ser una instrucción **CALL generada por hardware** (derivada en forma externa de una señal de hardware) o una instrucción **CALL generada por software** (derivada en forma interna de la ejecución de una instrucción o por algún otro evento interno). Algunas veces, a una interrupción interna se le llama *excepción*. Cualquiera de los dos tipos interrumpe el programa mediante una llamada a un **procedimiento de servicio de interrupciones** (ISP) o manejador de interrupciones.

En esta sección se hablará sobre las interrupciones de software, que son tipos especiales de instrucciones CALL. Se describirán los tres tipos de instrucciones de interrupción de software (INT, INTO e INT 3), se proporcionará un mapa de los vectores de interrupción y se explicará el propósito de la instrucción especial de retorno de interrupción (IRET).

Vectores de interrupción

Un **vector de interrupción** es un número de cuatro bytes que se almacena en los primeros 1024 bytes de la memoria (00000H-003FFH) cuando el microprocesador opera en modo real. En el modo protegido, la tabla de vectores se sustituye por una tabla de descriptores de interrupción, que utiliza descriptores de ocho bytes para describir cada una de las interrupciones. Hay 256 vectores de interrupción distintos, y cada vector contiene la dirección de un procedimiento de servicio de interrupciones. La tabla 6-4 lista los vectores de interrupción con una breve descripción y la posición de memoria de cada vector para el modo real. Cada vector contiene un valor para IP y CS que forma la dirección del procedimiento

TABLA 6-4 Los vectores de interrupción definidos por Intel.

Número	Dirección	Microprocesador	Función
0	0H-3H	Todos	Error de división.
1	4H-7H	Todos	Un solo paso.
2	8-BH	Todos	Terminal NMI.
3	CH-FH	Todos	Punto de ruptura (breakpoint).
4	10H-13H	Todos	Interrupción por desbordamiento.
5	14H-17H	80186-Pentium 4	Instrucción delimitada.
6	18H-1BH	80186-Pentium 4	Código de operación inválido.
7	1CH-1FH	80186-Pentium 4	Emulación del coprocesador.
8	20H-23H	80386-Pentium 4	Fallo doble.
9	24H-27H	80386	El coprocesador rebasó el segmento.
A	28H-2BH	80386-Pentium 4	Segmento de estado de tarea no válido.
B	2CH-2FH	80386-Pentium 4	El segmento no está presente.
C	30H-33H	80386-Pentium 4	Fallo en la pila.
D	34H-37H	80386-Pentium 4	Fallo de protección general (GPF).
E	38H-3BH	80386-Pentium 4	Fallo de página.
F	3CH-3FH	—	Reservado.
10	40H-43H	80286-Pentium 4	Error de punto flotante.
11	44H-47H	80486SX	Interrupción de comprobación de alineación.
12	48H-4BH	Pentium-Pentium 4	Excepción de comprobación del equipo.
13-1F	4CH-7FH	—	Reservado.
20-FF	80H-3FFH	—	Interrupciones del usuario.

del servicio de interrupciones. Los primeros dos bytes contienen el valor de IP y los últimos dos bytes contienen el valor de CS.

Intel reserva los primeros 32 vectores de interrupción para los productos de microprocesadores presentes y futuros. El resto de los vectores de interrupción (32-255) están disponibles para el usuario. Algunos de los vectores reservados son para errores que ocurren durante la ejecución de software, como la interrupción por error de división. Algunos vectores están reservados para el coprocesador. Otros se utilizan en eventos normales en el sistema. En una computadora personal, los vectores reservados se utilizan para las funciones del sistema, como veremos más adelante en esta sección. Los vectores 1-6, 7, 9, 16 y 17 funcionan en modo real y en modo protegido; el resto de los vectores sólo funcionan en modo protegido.

Instrucciones de interrupción

El microprocesador cuenta con tres instrucciones de interrupción distintas, que están disponibles para el programador: INT, INTO e INT 3. En el modo real, cada una de estas instrucciones obtiene un vector de la tabla de vectores y después llama al procedimiento almacenado en la posición direcciónada por el vector. En el modo protegido, cada una de estas instrucciones obtiene un descriptor de interrupción de la tabla de descriptores de interrupción. El descriptor especifica la dirección del procedimiento de servicio de interrupciones. La llamada a la interrupción es similar a una instrucción CALL lejana, ya que coloca la dirección de retorno (IP/EIP y CS) en la pila.

Instrucciones INT. Existen 256 instrucciones de interrupción de software (INTs) distintas, disponibles para el programador. Cada instrucción INT tiene un operando numérico cuyo rango es de 0 a 255

(00H-FFH). Por ejemplo, la instrucción INT 100 utiliza el vector de interrupción 100, que aparece en la dirección de memoria 190H-193H. La dirección del vector de interrupción se determina mediante la multiplicación del número de tipo de interrupción por 4. Por ejemplo, la instrucción INT 10H llama al procedimiento de servicio de interrupciones cuya dirección se almacena empezando en la posición de memoria 40H ($10H \times 4$) en el modo real. En el modo protegido, el descriptor de interrupción se localiza mediante la multiplicación del número de tipo por 8 en vez de 4, ya que cada descriptor es de 8 bytes.

Cada instrucción INT es de dos bytes. El primer byte contiene el código de operación y el segundo byte contiene el número de tipo de vector. La única excepción a esto es INT 3, una interrupción de software especial de un byte que se utiliza para los puntos de ruptura.

Cada vez que se ejecuta una instrucción de interrupción de software, (1) mete las banderas en la pila (2) borra los bits de las banderas T e I, (3) mete CS en la pila, (4) obtiene el nuevo valor para CS del vector de interrupción, (5) mete IP/EIP en la pila, (6) obtiene el nuevo valor para IP/EIP del vector y (7) salta a la nueva posición direccionalizada por CS e IP/EIP.

La instrucción INT funciona como una instrucción CALL lejana, con la excepción de que no sólo mete los registros CS e IP en la pila, sino que también mete las banderas. La instrucción INT realiza la operación de una instrucción PUSHF, seguida de una instrucción CALL lejana.

Cuando la instrucción INT se ejecuta, borra la bandera de interrupción (I) que controla la terminal INTR de entrada de interrupción de hardware externa (petición de interrupción). Cuando I = 0, el microprocesador deshabilita la terminal INTR; cuando I = 1, el microprocesador la habilita.

Las interrupciones de software se utilizan con más frecuencia para llamar procedimientos del sistema, ya que no necesita conocerse la dirección de la función del sistema. Los procedimientos del sistema son comunes para todo el software de sistema y de aplicación. A menudo las interrupciones controlan impresoras, pantallas de vídeo y unidades de disco. Además de liberar al programa de tener que recordar la dirección de la llamada del sistema, la instrucción INT sustituye a la instrucción CALL lejana que tendría que utilizarse para llamar a una función del sistema. La instrucción INT es de dos bytes, mientras que la instrucción CALL lejana es de cinco bytes. Cada vez que la instrucción INT sustituye a una instrucción CALL lejana, se ahorran tres bytes de memoria en un programa. Este ahorro puede llegar a ser considerable si la instrucción INT aparece con frecuencia en un programa, de igual forma que en las llamadas al sistema.

IRET/IROTD. La instrucción de retorno de interrupción (IRET) se utiliza sólo con los procedimientos de servicio de interrupciones de software o de hardware. A diferencia de una simple instrucción de retorno (RET), la instrucción IRET (1) saca los datos de la pila y los coloca de nuevo en el registro IP, (2) saca los datos de la pila y los coloca de nuevo en CS y (3) saca los datos de la pila y restaura el registro de banderas. La instrucción IRET realiza las mismas tareas que la instrucción POPF seguida de una instrucción RET lejana.

Cada vez que se ejecuta una instrucción IRET, se restaura el contenido de las banderas I y T de la pila. Esto es importante porque se preserva el estado de estos bits de bandera. Si se habilitaron las interrupciones antes de un procedimiento de servicio de interrupciones, se rehabilitan automáticamente mediante la instrucción IRET, ya que restaura el registro de banderas.

En los microprocesadores del 80386 al Pentium 4, la instrucción IROTD se utiliza para regresar de un procedimiento de servicio de interrupción que se llama en el modo protegido. Difiere de la instrucción IRET ya que saca un apuntador de instrucciones de 32 bits (EIP) de la pila. La instrucción IRET se utiliza en modo real y la instrucción IROTD se utiliza en modo protegido.

INT 3. Una instrucción INT 3 es una interrupción especial de software diseñada para funcionar como un **punto de ruptura** para interrumpir o romper el flujo del software. La diferencia entre esta interrupción y las demás interrupciones de software es que INT 3 es una instrucción de un byte, mientras que las demás son instrucciones de dos bytes.

Un punto de ruptura puede ocurrir para cualquier interrupción de software, pero como INT 3 es de un byte, es más fácil de usar para esta función. Los puntos de ruptura también ayudan a depurar las fallas en el software.

INTO. La interrupción por desbordamiento (INTO) es una interrupción condicional de software que evalúa la bandera de desbordamiento (O). Si O = 0, la instrucción INTO no realiza ninguna operación;

si O = 1 y se ejecuta una instrucción INTO, se produce una interrupción mediante el número de tipo de vector 4.

La instrucción INTO aparece en el software que suma o resta números binarios con signos. Con estas operaciones es posible tener un desbordamiento. La condición de desbordamiento se detecta mediante la instrucción JO o la instrucción INTO.

Un procedimiento de servicio de interrupciones. Suponga que en cierto sistema se requiere un procedimiento para sumar el contenido de DI, SI, BP y BX, para después almacenar la suma en AX. Como esta tarea es común en este sistema, algunas veces puede ser conveniente desarrollar la tarea como una interrupción de software. Aunque por lo general las interrupciones se reservan para los eventos del sistema, este ejemplo muestra cómo aparece un procedimiento de servicio de interrupciones. El ejemplo 6-18 muestra esta interrupción de software. La principal diferencia entre este procedimiento y un procedimiento lejano normal es que termina con la instrucción IRET en vez de la instrucción RET, y el contenido del registro de banderas se almacena en la pila durante su ejecución. También es importante almacenar todos los registros que modifica el procedimiento mediante USES.

EJEMPLO 6-18

0000	INTS	PROC FAR USES AX
0000 03 C3		ADD AX, BX
0002 03 05		ADD AX, BP
0004 03 C7		ADD AX, DI
0006 03 C6		ADD AX, SI
0008 CF		IRET
0009	INTS	ENDP

Control de interrupciones

Aunque en esta sección no se explicará el funcionamiento de las interrupciones de hardware, presentaremos dos instrucciones que controlan la terminal INTR. La instrucción **activa bandera de interrupción** (STI) coloca un 1 en el bit de la bandera I, el cual habilita la terminal INTR. La instrucción **borra bandera de interrupción** (CLI) coloca un 0 en el bit de la bandera I, el cual deshabilita la terminal INTR. La instrucción STI habilita INTR y la instrucción CLI la deshabilita. En un procedimiento de servicio de interrupciones de software, las interrupciones de hardware se habilitan como uno de los primeros pasos. Esto se logra mediante la instrucción STI. La razón por la que deben habilitarse las interrupciones lo antes posible en un procedimiento de servicio de interrupciones es que casi todos los dispositivos de E/S en la computadora personal se procesan mediante interrupciones. Si se deshabilitan las interrupciones por mucho tiempo, se producen problemas severos en el sistema.

Interrupciones en la computadora personal

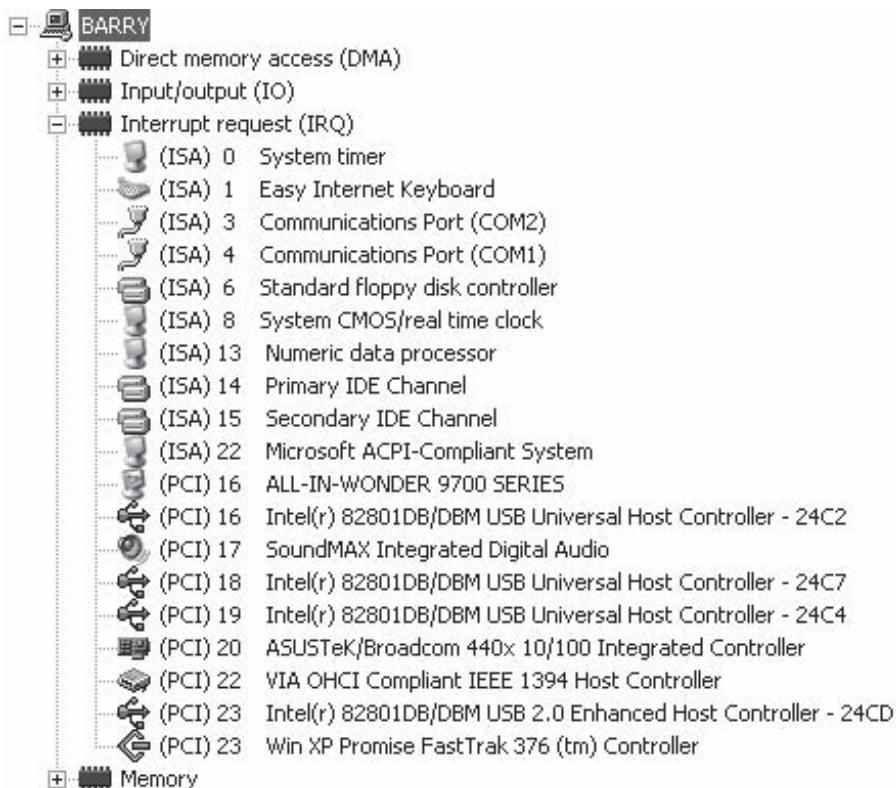
Las interrupciones que se encuentran en la computadora personal difieren en cierta forma de las que se presentan en la tabla 6-4. La razón es que las computadoras personales originales eran sistemas basados en el microprocesador 8086/8088. Esto significa que sólo contenían las interrupciones 0-4 especificadas por Intel. Este diseño se ha ampliado de manera que los sistemas más recientes son compatibles con las primeras computadoras personales.

El acceso a la estructura de interrupciones en modo protegido que utiliza Windows se realiza mediante funciones del núcleo (kernel) que proporciona Microsoft y no pueden direccionarse en forma directa. Las interrupciones en modo protegido utilizan una tabla de descriptores de interrupción, un tema que está más allá del alcance del libro en este punto. En capítulos posteriores se hablará sobre todo lo relacionado con las interrupciones en modo protegido.

La figura 6-9 ilustra las interrupciones disponibles en la computadora del autor. Las asignaciones de interrupciones pueden verse en el panel de control de Windows, en Rendimiento y Mantenimiento. Haga clic en Sistema, seleccione la ficha Hardware y después haga clic en el botón Administrador de dispositivos. Luego haga clic en Ver | Recursos por tipo, y finalmente seleccione la opción Solicitud de interrupciones (IRQ).

FIGURA 6-9

Las interrupciones en una computadora personal típica.



6-5

INSTRUCCIONES VARIAS Y DE CONTROL DE LA MÁQUINA

La última categoría de instrucciones en modo real que existen en el microprocesador corresponden al grupo de instrucciones de control de la máquina e instrucciones varias. Estas instrucciones proporcionan el control del bit de acarreo, muestran la terminal BUSY/TEST y realizan otras funciones diversas. Como muchas de estas instrucciones se utilizan en el control del hardware, en esta sección sólo las explicaremos en forma breve.

Control del bit de bandera de acarreo

La bandera de acarreo (C) propaga el acarreo de suma o de resta en las operaciones de suma y resta de varias palabras/dobles palabras. También puede indicar errores en los procedimientos en lenguaje ensamblador. Hay tres instrucciones que controlan el contenido de la bandera de acarreo: STC (activa el acarreo), CLC (borra el acarreo) y CMC (complementa el acarreo).

Como la bandera de acarreo se utiliza raras veces excepto en la suma y resta con varias palabras, está disponible para otros usos. La tarea más común de la bandera de acarreo es indicar un error al momento de regresar de un procedimiento. Suponga que un procedimiento lee datos de un archivo en la memoria en disco. Esta operación puede tener éxito o puede ocurrir un error tal como el de archivo no encontrado. Al regresar de este procedimiento, si C = 1 significa que ha ocurrido un error; si C = 0 significa que no ocurrió ningún error. La mayoría de los procedimientos del DOS y del BIOS utilizan la bandera de acarreo para indicar condiciones de error. Esta bandera no está disponible en Visual C/C++, por lo que no se puede utilizar en C++.

WAIT

La instrucción WAIT monitorea la terminal de hardware *BUSY* en los microprocesadores 80286 y 80386, y la terminal *TEST* en el 8086/8088. El nombre de esta terminal se cambió de *TEST* a *BUSY*.

desde el microprocesador 80286. Si la instrucción WAIT se ejecuta mientras la terminal $\overline{BUSY} = 1$, no ocurre nada y se ejecuta la siguiente instrucción. Si la terminal $\overline{BUSY} = 0$ cuando se ejecuta la instrucción WAIT, el microprocesador espera a que la terminal \overline{BUSY} regrese a un 1 lógico. Esta terminal introduce una condición de “ocupado” cuando está en el nivel de 0 lógico.

Por lo general, la terminal $\overline{BUSY/TEST}$ del microprocesador se conecta a la terminal \overline{BUSY} de los coprocesadores numéricos del 8087 al 80387. Esta conexión permite al microprocesador esperar hasta que el coprocesador termine una tarea. Como el coprocesador está dentro de los microprocesadores del 80486 al Pentium 4, la terminal \overline{BUSY} no está presente en ellos.

HLT

La instrucción de alto (HLT) detiene la ejecución del software. Hay tres formas de salir de un alto: mediante una interrupción, un reinicio de hardware o durante una operación de DMA. Esta instrucción por lo general aparece en un programa para esperar a una interrupción. A menudo sincroniza las interrupciones de hardware externas con el sistema de software. Tanto el DOS como Windows utilizan las interrupciones en forma exclusiva, por lo que HLT no detendrá la computadora cuando se opere bajo estos sistemas operativos.

NOP

Cuando el microprocesador encuentra una instrucción de no operación (NOP), se tarda un periodo corto de tiempo en ejecutarla. En los primeros años, antes de que hubiera herramientas de desarrollo de software disponibles, se utilizaba con frecuencia una instrucción NOP (que no realiza ninguna operación) para dejar espacio en el software para futuras instrucciones en lenguaje máquina. Si usted desarrolla programas en lenguaje máquina, que son en extremo raros, se recomienda que coloque 10 o más NOPs en su programa, en intervalos de 50 bytes. Esto se hace en caso de que necesite agregar instrucciones en cierto punto en el futuro. También se puede aplicar una instrucción NOP para crear retrasos de tiempo. Sin embargo, no es muy preciso utilizar una NOP para medir tiempos, debido a la caché y las canalizaciones en los microprocesadores modernos.

Prefijo LOCK

Este prefijo se coloca antes de una instrucción y hace que la terminal \overline{LOCK} sea un 0 lógico. La terminal \overline{LOCK} a menudo deshabilita los componentes maestros del bus u otros componentes del sistema. El prefijo LOCK hace que la terminal \overline{LOCK} se active sólo para la duración de una instrucción bloqueada. Si se bloquea más de una instrucción secuencial, la terminal \overline{LOCK} se mantiene en un 0 lógico durante toda la secuencia de instrucciones bloqueadas. La instrucción `LOCK:MOV AL,[SI]` es un ejemplo de una instrucción bloqueada.

ESC

La instrucción de escape (ESC) pasa instrucciones desde el microprocesador al coprocesador de punto flotante. Cada vez que se ejecuta una instrucción ESC, el microprocesador proporciona la dirección de memoria en caso de que se requiera; en caso contrario, ejecuta una instrucción NOP. Seis bits de la instrucción ESC proporcionan el código de operación al coprocesador y empiezan la ejecución de una instrucción del coprocesador.

El código de operación ESC nunca aparece en un programa como ESC, ya que en sí se considera obsoleto como código de operación. En su lugar existe un conjunto de instrucciones del coprocesador (FLD, FST, FMUL, etc.) que se ensamblan como instrucciones ESC para el coprocesador. En el capítulo 13 veremos esto con más detalle, ya que hablaremos sobre los coprocesadores numéricos del 8087 al Pentium 4.

BOUND

Esta instrucción, que estuvo disponible por primera vez en el microprocesador 80186, es una instrucción de comparación que puede producir una interrupción (número de tipo de vector 5). Esta instrucción

compara el contenido de cualquier registro de 16 o de 32 bits con el contenido de dos palabras o dobles palabras de memoria: un límite superior y uno inferior. Si el valor en el registro que se compara con la memoria no está dentro de los límites superior e inferior, se produce una interrupción tipo 5. Si está dentro de los límites, se ejecuta la siguiente instrucción en el programa.

Por ejemplo, si se ejecuta la instrucción BOUND SI,DATOS, la posición tipo palabra DATOS contiene el límite inferior y la posición tipo palabra DATOS+2 contiene el límite superior. Si el número que contiene SI es menor que la posición de memoria DATOS o mayor que la posición de memoria DATOS +2 bytes, se produce una interrupción tipo 5. Cuando ocurre esta interrupción, la dirección de retorno apunta a la instrucción BOUND y no a la instrucción que va después de BOUND. Esto difiere de una interrupción normal, en donde la dirección de retorno apunta a la siguiente instrucción en el programa.

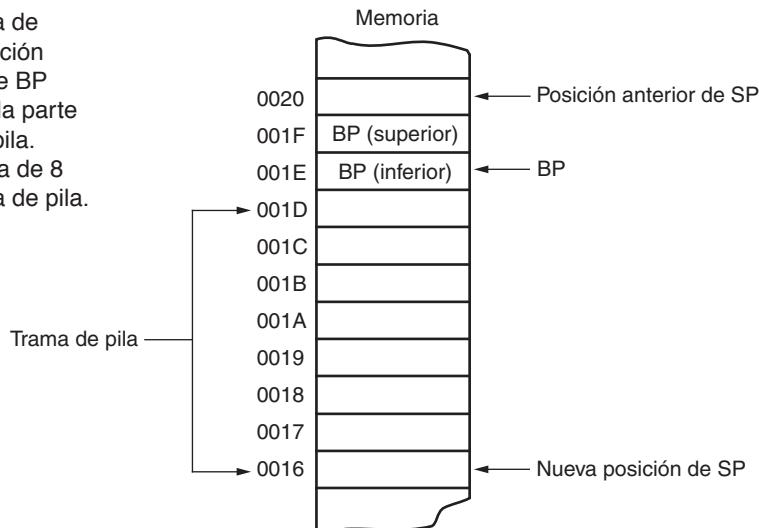
ENTER y LEAVE

Estas instrucciones, que estuvieron disponibles por primera vez en el microprocesador 80186, se utilizan con las tramas de pila, que son mecanismos utilizados para pasar parámetros a un procedimiento a través de la memoria de la pila. La trama de pila también almacena las variables de memoria locales para el procedimiento. Las tramas de pila proporcionan áreas dinámicas de memoria para los procedimientos, en entornos multiusuario.

La instrucción ENTER crea una trama de pila al meter BP en la pila y después cargar BP con la dirección más alta de la trama de pila. Esto permite el acceso a las variables de la trama de pila mediante el registro BP. La instrucción ENTER contiene dos operandos. El primer operando especifica el número de bytes a reservar para las variables en la trama de pila y el segundo especifica el nivel del procedimiento.

Suponga que se ejecuta la instrucción ENTER 8,0. Esta instrucción reserva ocho bytes de memoria para la trama de pila y el cero especifica el nivel 0. La figura 6-10 muestra la trama de pila que se establece mediante esta instrucción. Observe que esta instrucción almacena BP en la parte superior de la pila. Después resta 8 del apuntador de la pila, dejando ocho bytes de espacio en memoria para el almacenamiento temporal de datos. La posición más alta de esta área de almacenamiento temporal de ocho bytes se direcciona mediante BP. La instrucción LEAVE invierte este proceso al recargar tanto SP como BP con sus valores anteriores. Las instrucciones ENTER y LEAVE se utilizaban para llamar funciones de C++ en Windows 3.1, pero desde entonces se utiliza CALL en las versiones modernas de Windows para las funciones de C++.

FIGURA 6-10 La trama de pila creada por la instrucción ENTER 8,0. Observe que BP se almacena a partir de la parte superior de la trama de pila. Después le sigue un área de 8 bytes que se llama trama de pila.



6-6**RESUMEN**

1. Existen tres tipos de instrucciones de salto incondicional: corto, cercano y lejano. El salto corto permite una ramificación hacia un rango de entre +127 y -128 bytes. El salto cercano (mediante el uso de un desplazamiento de $\pm 32\text{ K}$) permite un salto hacia cualquier posición en el segmento de código actual (inrasegmento). El salto lejano permite un salto hacia cualquier posición en el sistema de memoria (intersegmento). El salto cercano en un microprocesador del 80386 al Pentium 4 está dentro de un rango de $\pm 2\text{ Gbytes}$, ya que estos microprocesadores pueden usar un desplazamiento con signo de 32 bits.
2. Siempre que aparece una etiqueta con una instrucción JMP o salto condicional, la etiqueta (que se ubica en el campo de etiquetas) debe ir seguida de un signo de dos puntos (ETIQUETA:). Por ejemplo, la instrucción JMP PERRITO salta hacia la posición de memoria PERRITO:.
3. El desplazamiento que va después de un salto corto o cercano es la distancia desde la siguiente instrucción hasta la posición de salto.
4. Los saltos indirectos están disponibles en dos formas (1) salto hacia la posición almacenada en un registro y (2) salto hacia la posición almacenada en una palabra de memoria (cercano indirecto) o en una doble palabra (lejano indirecto).
5. Los saltos condicionales son todos saltos cortos que evalúan una o más de los siguientes bits de bandera: C, Z, O, P o S. Si la condición es verdadera se produce un salto; si la condición es falsa se ejecuta la siguiente instrucción secuencial. Tenga en cuenta que los microprocesadores 80386 y superiores permiten un desplazamiento con signo de 16 bits para las instrucciones de salto condicional.
6. Una instrucción especial de salto condicional (LOOP) decremente CX y salta hacia la etiqueta cuando CX es distinto de 0. Otras formas del ciclo son: LOOPE, LOOPNE, LOOPZ y LOOPNZ. La instrucción LOOPE salta si CX es distinto de 0 y si existe una condición de igualdad. En los microprocesadores del 80386 al Pentium 4, las instrucciones LOOPD, LOOPED y LOOPND también utilizan el registro ECX como contador.
7. Los microprocesadores del 80386 al Pentium 4 contienen conjuntos de instrucciones que activan un byte con 01H o lo borran con 00H. Si la condición que se evalúa es verdadera, el byte del operando se hace 01H; si la condición que se evalúa es falsa, el byte del operando se hace 00H.
8. Las instrucciones .IF y .ENDIF son útiles en el lenguaje ensamblador para tomar decisiones. Estas instrucciones hacen que el ensamblador genere instrucciones de salto condicional, las cuales modifican el flujo del programa.
9. Las instrucciones .WHILE y .ENDW permiten que un programa en lenguaje ensamblador utilice la construcción WHILE, y las instrucciones .REPEAT y UNTIL permiten que un lenguaje ensamblador utilice la construcción REPEAT-UNTIL.
10. Los procedimientos son grupos de instrucciones que realizan una tarea y se utilizan desde cualquier punto en un programa. La instrucción CALL se enlaza a un procedimiento y la instrucción RET regresa de un procedimiento. En lenguaje ensamblador, la directiva PROC define el nombre y el tipo del procedimiento. La directiva ENDP declara el final del procedimiento.
11. La instrucción CALL es una combinación de las instrucciones PUSH y JMP. Cuando CALL se ejecuta, mete la dirección de retorno en la pila y después salta hacia el procedimiento. Una instrucción CALL cercana coloca el contenido de IP en la pila y una instrucción CALL lejana coloca a IP y a CS en la pila.
12. Para regresar de un procedimiento, la instrucción RET saca la dirección de retorno de la pila y la coloca en IP (retorno cercano), o en IP y CS (retorno lejano).
13. Las interrupciones son instrucciones de software similares a CALL, o señales de hardware que se utilizan para llamar procedimientos. Este proceso interrumpe el programa actual y llama a un procedimiento. Después de que el procedimiento se ejecuta, una instrucción IRET especial regresa el control al software interrumpido.
14. Los vectores de interrupción en modo real son de cuatro bytes y contienen la dirección (IP y CS) del procedimiento de servicio de interrupciones. El microprocesador contiene 256 vectores de interrupción en el primer 1 Kbyte de memoria. Intel define los primeros 32 vectores; los 224 restantes son interrupciones del usuario. En la operación en modo protegido, el vector de interrupción

- es de ocho bytes de largo y la tabla de vectores de interrupción puede reubicarse en cualquier sección del sistema de memoria.
15. Cada vez que el microprocesador acepta una interrupción se meten las banderas, IP y CS en la pila. Además de meter las banderas, se borran los bits de las banderas T e I para deshabilitar la función de rastreo y la terminal INTR. El evento final que ocurre para la interrupción es que el vector de interrupción se obtiene de la tabla de vectores y se produce un salto hacia el procedimiento de servicio de interrupciones.
 16. A menudo, las instrucciones de interrupción de software (INT) sustituyen las llamadas al sistema. Las interrupciones de software ahorran tres bytes de memoria cada vez que sustituyen a las instrucciones CALL.
 17. Debe utilizarse una instrucción especial de retorno (IRET) para regresar de un procedimiento de servicio de interrupciones. La instrucción IRET no sólo saca IP y CS de la pila, sino también las banderas.
 18. La interrupción por desbordamiento (INTO) es una interrupción condicional que llama a un procedimiento de servicio de interrupciones si la bandera de desbordamiento (O) = 1.
 19. La bandera de habilitación de interrupciones (I) controla la conexión de la terminal INTR en el microprocesador. Si se ejecuta la instrucción STI, se activa I para habilitar la terminal INTR. Si se ejecuta la instrucción CLI, se borra I para deshabilitar la terminal INTR.
 20. El bit de la bandera de acarreo (C) puede borrarse, activarse y complementarse mediante las instrucciones CLC, STC y CMC.
 21. La instrucción WAIT evalúa la condición de la terminal \overline{BUSY} o \overline{TEST} en el microprocesador. Si \overline{BUSY} o $\overline{TEST} = 1$, WAIT no espera; pero si \overline{BUSY} o $\overline{TEST} = 0$, WAIT continúa evaluando la terminal \overline{BUSY} o \overline{TEST} hasta que se convierta en un 1 lógico. Tenga en cuenta que el microprocesador 8086/8088 contiene la terminal \overline{TEST} , mientras que los microprocesadores 80286-80386 contienen la terminal \overline{BUSY} . Los microprocesadores del 80486 al Pentium 4 no contienen una terminal \overline{BUSY} o \overline{TEST} .
 22. El prefijo LOCK hace que la terminal \overline{LOCK} se convierta en un 0 lógico mientras dure la instrucción bloqueada. La instrucción ESC pasa la instrucción al coprocesador numérico.
 23. La instrucción BOUND compara el contenido de cualquier registro de 16 bits con el contenido de dos palabras de memoria; un límite superior y uno inferior. Si el valor en el registro que se compara con la memoria no está dentro de los límites superior e inferior, se produce una interrupción tipo 5.
 24. Las instrucciones ENTER y LEAVE se utilizan con las tramas de pila. Una trama de pila es un mecanismo que se utiliza para pasar parámetros a un procedimiento a través de la memoria de la pila. La trama de pila también almacena las variables de memoria locales para el procedimiento. La instrucción ENTER crea la trama de pila y la instrucción LEAVE saca la trama de pila de la pila. El registro BP direcciona los datos de la trama de pila.

6-7**PREGUNTAS Y PROBLEMAS**

1. ¿Qué es un salto (JMP) corto?
2. ¿Qué tipo de instrucción JMP se utiliza cuando se salta hacia cualquier posición dentro del segmento de código actual?
3. ¿Qué instrucción JMP permite que el programa continúe su ejecución en cualquier posición de memoria en el sistema?
4. ¿Qué instrucción JMP es de cinco bytes?
5. ¿Cuál es el rango de un salto cercano en los microprocesadores del 80386 al Pentium 4?
6. Indique el tipo de instrucción JMP (corto, cercano o lejano) que se ensambla para los siguientes casos:
 - (a) Si la distancia es de 0210H bytes.
 - (b) Si la distancia es de 0020H bytes.
 - (c) Si la distancia es de 10000H bytes.

7. ¿Qué puede decirse sobre una etiqueta que va seguida del signo de dos puntos?
8. ¿Cuál registro o registros cambia el salto cercano para modificar la dirección del programa?
9. ¿Cuál registro o registros cambia el salto lejano para modificar la dirección del programa?
10. Explique qué hace la instrucción JMP AX. Identifíquela como una instrucción de salto cercano o lejano.
11. Contraste la operación de una instrucción JMP DI con la de una instrucción JMP [DI].
12. Contraste la operación de una instrucción JMP [DI] con la de una instrucción JMP FAR PTR [DI].
13. Liste los cinco bits de bandera que se evalúan mediante las instrucciones de salto condicional.
14. Describa cómo opera la instrucción JA.
15. ¿Cuándo hace el salto la instrucción JO?
16. ¿Qué instrucciones de salto condicional van después de la comparación de números con signo?
17. ¿Qué instrucciones de salto condicional van después de la comparación de números sin signo?
18. ¿Qué instrucciones de salto condicional evalúan los bits de bandera Z y C?
19. ¿Cuándo hace el salto la instrucción JCXZ?
20. ¿Qué instrucción SET se utiliza para establecer AL si los bits de bandera indican una condición de cero?
21. La instrucción LOOP del 8086 decremente el registro _____ y lo evalúa para ver si es 0, para decidir si se produce un salto o no.
22. La instrucción LOOPD del Pentium 4 decremente el registro _____ y lo evalúa para ver si es 0, para decidir si se produce un salto o no.
23. Explique cómo opera la instrucción LOOPE.
24. Desarrolle una secuencia corta de instrucciones que almacene un 00H en 150H bytes de memoria, empezando en la posición de memoria DATAZ del segmento extra. Debe utilizar la instrucción LOOP para ayudar a realizar esta tarea.
25. Desarrolle una secuencia de instrucciones que realicen búsquedas a través de un bloque de 100H bytes de memoria. Este programa debe contar todos los números sin signo mayores de 42H y menores de 42H. La posición de memoria MAYORES tipo byte del segmento de datos debe almacenar la cuenta de los números mayores de 42H, y la posición MENORES del segmento de datos debe almacenar la cuenta de los números menores de 42H.
26. Muestre las instrucciones en lenguaje ensamblador que se generan mediante la siguiente secuencia:

```
.IF AL == 3
    ADD AL,2
.ENDIF
```

27. ¿Qué ocurre si se coloca la instrucción .WHILE 1 en un programa?
28. Desarrolle una secuencia corta de instrucciones que utilice la construcción REPEAT-UNTIL para copiar el contenido de la memoria tipo byte BLOQUEA en la memoria tipo byte BLOQUEB hasta que se mueva un 00H.
29. ¿Cuál es el propósito de la directiva .BREAK?
30. Utilice la construcción WHILE para desarrollar una secuencia de instrucciones que sumen el contenido tipo byte de BLOQUEA a BLOQUEB mientras la suma no sea 12H.
31. ¿Qué es un procedimiento?
32. Explique cómo funcionan las instrucciones CALL cercana y lejana.
33. ¿Cómo funciona la instrucción RET cercano?
34. La última instrucción ejecutable en un procedimiento debe ser _____.
35. ¿Qué directiva identifica el inicio de un procedimiento?
36. ¿Cómo se identifica un procedimiento como cercano o lejano?
37. Explique qué hace la instrucción RET 6.
38. Escriba un procedimiento cercano que eleve al cubo el contenido del registro CX. Este procedimiento no puede afectar a ningún registro excepto CX.
39. Escriba un procedimiento que multiplique DI por SI y que luego divida el resultado entre 100H. Asegúrese de que el resultado se deje en AX al momento de regresar del procedimiento. Este procedimiento no puede cambiar a ningún registro, excepto AX.

40. Escriba un procedimiento que sume EAX, EBX, ECX y EDX. Si se produce un acarreo, coloque un 1 lógico en EDI. Si no hay acarreo, coloque un 0 en EDI. La suma debe encontrarse en EAX después de ejecutar su procedimiento.
41. ¿Qué es una interrupción?
42. ¿Qué instrucciones de software llaman a un procedimiento de servicio de interrupciones?
43. ¿Cuántos tipos distintos de interrupciones hay disponibles en el microprocesador?
44. ¿Cuál es el propósito del vector de interrupción cuyo número de tipo es 0?
45. Ilustre el contenido de un vector de interrupción y explique el propósito de cada parte.
46. ¿En qué difiere la instrucción IRET de la instrucción RET?
47. ¿Cuál es la instrucción IRETD?
48. ¿Cuál es la única condición que hace que la instrucción INTO interrumpa el programa?
49. ¿En qué posiciones de memoria se almacena el vector de interrupción para una instrucción INT 40H?
50. ¿Qué instrucciones controlan la función de la terminal INTR?
51. ¿Qué instrucción evalúa la terminal BUSY?
52. ¿Cuándo interrumpe un programa la instrucción BOUND?
53. Una instrucción ENTER 16,0 crea una trama de pila que contiene _____ bytes.
54. ¿Qué registro se mueve a la pila cuando se ejecuta una instrucción ENTER?
55. ¿Qué instrucción pasa códigos de operación al coprocesador numérico?

CAPÍTULO 7

Uso del lenguaje ensamblador con C/C++

INTRODUCCIÓN

Actualmente es raro desarrollar un sistema completo sólo mediante el uso del lenguaje ensamblador. Con frecuencia se utiliza C/C++ con algo de lenguaje ensamblador para desarrollar un sistema. La porción correspondiente al lenguaje ensamblador casi siempre resuelve tareas (difíciles o poco eficientes como para desarrollarlas en C/C++) que a menudo incluyen software de control para interfaces de periféricos y programas controladores que utilizan interrupciones. Otra aplicación del lenguaje ensamblador en los programas de C/C++ es para las instrucciones MMX y SEC, las cuales forman parte de los microprocesadores de clase Pentium y que no se soportan en C/C++. Aunque C++ tiene macros para estos comandos, son más complicados de usar que el lenguaje ensamblador. En este capítulo desarrollaremos la idea de mezclar C/C++ y el lenguaje ensamblador. Muchas de las aplicaciones en capítulos posteriores también ilustran el uso de lenguaje ensamblador y C/C++ para realizar tareas para el microprocesador.

En este libro utilizaremos Microsoft Visual C/C++, pero es muy probable que los programas puedan adaptarse a cualquier versión de C/C++, siempre y cuando sea C/C++ en formato ANSI (Instituto Nacional Estadounidense de Estándares) estándar. Si lo desea, puede utilizar C/C++ para introducir y ejecutar todas las aplicaciones de programación en este texto. Las aplicaciones de 16 bits están escritas mediante Microsoft Visual C/C++ versión 1.52 o más reciente (disponible [CL.EXE] en forma gratuita como aplicación heredada en el Kit de desarrollo de controladores para Microsoft Windows [DDK]); las aplicaciones de 32 bits están escritas mediante Microsoft Visual C/C++ versión 6 o más reciente (de preferencia Microsoft Visual C/C++ versión .NET 2003). Los ejemplos en este libro están escritos en base a la suposición de que usted cuenta con la versión más reciente.

OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

1. Utilizar el lenguaje ensamblador en bloques _asm dentro de C/C++.
2. Aprender las reglas que se aplican al desarrollo de software con lenguajes mixtos.
3. Utilizar datos y estructuras comunes en C/C++ con el lenguaje ensamblador.
4. Utilizar la interfaz de 16 bits (DOS) y la interfaz de 32 bits (Windows) con código en lenguaje ensamblador.
5. Utilizar objetos del lenguaje ensamblador con programas en C/C++.

7-1

USO DEL LENGUAJE ENSAMBLADOR CON C++ PARA APLICACIONES DOS DE 16 BITS

En esta sección se mostrará cómo incorporar comandos en lenguaje ensamblador dentro de un programa en C/C++. Esto es importante, ya que el rendimiento de un programa depende a menudo de la incorporación de secuencias en lenguaje ensamblador para agilizar su ejecución. Como se mencionó en la introducción de este capítulo, el lenguaje ensamblador también se utiliza para las operaciones de E/S en sistemas integrados. En este libro asumiremos que cuenta con una versión del programa Microsoft C/C++, pero cualquier programa de C/C++ debe funcionar como se muestra si soporta los comandos del ensamblador en línea. El único cambio podría ser configurar el paquete de C/C++ para que funcione con el lenguaje ensamblador. En esta sección supondrá que va a crear aplicaciones de 16 bits para DOS. Asegúrese de que su software pueda crear aplicaciones de 16 bits antes de probar los programas de esta sección. Si crea una aplicación de 32 bits y trata de utilizar la función INT 21H del DOS, el programa fallará porque no se permiten las llamadas al DOS en forma directa. De hecho, son inefficientes si se utilizan en una aplicación de 32 bits.

Para crear una aplicación DOS de 16 bits necesitará el compilador de 16 bits heredado, que por lo general se encuentra en el directorio C:\WINDDK\2600.1106\bin\win_me\bin16 del DDK de Windows. (Puede obtener el **Kit de desarrollo de controladores para Windows** de Microsoft Corporation por un pequeño costo de envío.) El compilador es CL.EXE y el programa enlazador de 16 bits es LINK.EXE. Ambos se encuentran en el directorio o carpeta antes mencionada. Como es probable que la ruta en la computadora que utiliza apunte al programa enlazador de 32 bits, sería conveniente trabajar desde este directorio, para que se utilice el enlazador correcto cuando se enlacen los archivos objeto generados por el compilador. La compilación y el enlace deben realizarse desde la línea de comandos, ya que no hay una interfaz visual o un editor incluido con el compilador y el enlazador. Los programas se generan mediante el uso del Bloc de notas o del programa Edit del DOS.

Reglas básicas y programas simples

Antes de poder colocar código en lenguaje ensamblador en un programa de C/C++, hay que conocer ciertas reglas. El ejemplo 7-1 muestra cómo colocar código ensamblador en un bloque de lenguaje ensamblador dentro de un programa en C/C++. Observe que todo el código ensamblador en este ejemplo se coloca en el bloque **_asm**. Las etiquetas se utilizan según como se muestra en la etiqueta **grande**: en este ejemplo. También es importante utilizar caracteres en minúsculas para cualquier código de ensamblador en línea. Si utiliza mayúsculas descubrirá que algunos de los comandos y registros del lenguaje ensamblador son palabras reservadas o definidas en el lenguaje C/C++.

El ejemplo 7-1 no utiliza comandos de C/C++, excepto el procedimiento principal (main). Escriba el programa mediante el Bloc de notas o Edit. El programa lee un carácter del teclado de consola y luego lo filtra a través de lenguaje ensamblador, para que sólo los números del 0 al 9 se envíen de regreso a la pantalla de vídeo. Aunque este ejemplo de programación no hace mucho, sí muestra cómo establecer y utilizar algunas construcciones simples de programación en el entorno C/C++ y también muestra cómo utilizar el ensamblador en línea.

EJEMPLO 7-1

```
//Acepta y muestra un carácter del 1 al 9,
//todos los demás se ignoran.

void main(void)
{
    _asm
    {
        mov ah,8          ;lee tecla sin eco
        int 21h
        cmp al, '0'      ;filtra el código de la tecla
        jb grande
        cmp al, '9'
        ja grande
    }
}
```

```

        mov dl,al           ; eco del 0 al 9
        mov ah,2
        int 21h
    grande:
}
}

```

En el ejemplo 7-1 no se almacena el registro AX, pero el programa lo utiliza. Es muy importante tener en cuenta que los registros AX, BX, CX, DX y ES nunca se utilizan en Microsoft C/C++. (Más adelante en este capítulo se explicará la función de AX en el regreso de un procedimiento.) Estos registros, que podrían considerarse como registros **de apuntes**, están disponibles para usarse en lenguaje ensamblador. Si desea utilizar cualquiera de los otros registros, asegúrese de guardarlos con una instrucción PUSH antes de utilizarlos, y de restaurarlos con una instrucción POP después de usarlos. Si no guarda los registros que utiliza un programa, éste podría no funcionar en forma adecuada, pudiendo incluso hacer que la computadora falle. Si utiliza el procesador 80386 o superior como base para su programa, no necesita guardar EAX, EBX, ECX, EDX y ES.

Para compilar el programa, inicie el programa Símbolo del sistema que se encuentra en el menú Inicio | Todos los programas | Accesorios. Cambie la ruta a C:\WINDDK\2600.1106\bin\win_me\bin16, si es ahí donde tiene el DDK de Windows. También tendrá que ir al directorio C:\WINDDK\2600.1106\lib\win_me y copiar slibce.lib al directorio C:\WINDDK\2600.1106\bin\win_me\bin16. Asegúrese de guardar el programa en la misma ruta y utilizar la extensión .c en el nombre del archivo. Si utiliza el Bloc de notas, asegúrese de seleccionar la opción **Todos los archivos** de la lista **Tipo** en el cuadro de diálogo **Guardar como** cuando vaya a guardar el programa. Para compilarlo, escriba CL /G3 nombrearchivo.c>. Esto generará el archivo .exe (/G3 es el 80386) para el programa. (En la tabla 7-1 podrá consultar la lista de los modificadores de compilador /G.) Puede ignorar cualquier error que aparezca si oprime la tecla Intro. Estos errores generan advertencias que no ocasionarán problemas cuando se ejecute el programa. Después de la ejecución, lo único que verá es un número que se vuelve a mostrar con eco en la pantalla del DOS.

El ejemplo 7-2 muestra cómo utilizar variables desde C con un programa corto en lenguaje ensamblador. En este ejemplo se utiliza la variable tipo char (un byte en C) para ahorrar espacio para unos cuantos bytes de datos de ocho bits. El programa en sí realiza la operación X + Y = Z, en donde X e Y son dos números de un dígito y Z es el resultado. Como es de imaginar, se podría usar el ensamblador en línea en C para aprender lenguaje ensamblador y escribir muchos de los programas en este libro. El signo de punto y coma agrega comentarios al listado en el bloque _asm, justo igual que el ensamblador normal.

EJEMPLO 7-2

```

void main(void)
{
    char a, b;
    _asm
    {
        mov    ah, 1           ; lee el primer dígito
        int    21h
        mov    a, al
        mov    ah, 1           ; lee un signo +
}
}

```

TABLA 7-1
Las opciones G del compilador (16 bits).

Modificador del compilador	Función
/G1	Selecciona el 8088/8086
/G2	Selecciona el 80188/80186/80286
/G3	Selecciona el 80386
/G4	Selecciona el 80486
/G5	Selecciona el Pentium
/G6	Selecciona el Pentium Pro-Pentium 4

Nota: El compilador de C++ de 32 bits no reconoce /G1 o /G2.

```

int    21h
cmp   al, '+'
jne   fin1           ;si no es el signo de suma
mov   ah, 1
int   21h           ;lee el segundo número
mov   b, al
mov   ah, 2           ;muestra el =
mov   dl, '='
int   21h
mov   ah, 0           ;genera la suma
mov   al, a
add   al, b
aaa
add   ax, 3030h
cmp   ah, '0'
je    abajo
push  ax             ;muestra la posición de los 10's
mov   dl, ah
mov   ah, 2
int   21h
pop   ax
abajo:
mov   dl, al           ;muestra la posición de las unidades
mov   ah, 2
int   21h
fin1:
}
}

```

Lo que no puede usarse de MASM dentro de un bloque `_asm`

Aunque MASM contiene algunas buenas características, como los comandos condicionales (.IF, .WHILE, .REPEAT, etc.), el ensamblador en línea no incluye los comandos condicionales de MASM ni incluye la característica MACRO del ensamblador. En el ensamblador en línea, la asignación de datos se maneja mediante C en vez de utilizar DB, DW, DD, etcétera. Casi todas las demás características sí están soportadas, pero las que se omiten pueden producir algunos problemas, no muy graves, como veremos en las siguientes secciones de este capítulo.

Uso de cadenas de caracteres

El ejemplo 7-3 ilustra un programa simple que utiliza una cadena de caracteres definida con C y la muestra de forma que cada palabra se liste en una línea separada. Observe la mezcla de instrucciones en C y en lenguaje ensamblador. La instrucción WHILE repite los comandos de lenguaje ensamblador hasta que se descubre el valor nulo (00H) al final de la cadena de caracteres. Si no se descubre este valor, la instrucción de lenguaje ensamblador muestra un carácter de la cadena, a menos que se localice un espacio. Para cada espacio, el programa muestra una combinación de retorno de carro/avance de línea. Esto hace que cada palabra de la cadena se muestre en una línea separada.

EJEMPLO 7-3

```

//Ejemplo que muestra una cadena de caracteres, una palabra por línea

void main(void)
{
    char cadena1[] = "Ésta es mi primera aplicación de prueba que utiliza _asm.\n";
    int sc = -1;
    while (cadena1[sc++] != 0)
    {
        _asm
        {
            push    si
            mov     si, sc           ;obtiene el apuntador

```

```

        mov    dl, cadenal[si]      ;obtiene el carácter
        cmp    dl, ' '
        jne    siguiente
        mov    ah, 2                  ;muestra una nueva línea
        mov    dl, 10
        int    21h
        mov    dl, 13
siguiente: mov   ah, 2            ;muestra el carácter
           int   21h
           pop   si

}
}

```

Suponga que desea mostrar más de una cadena en un programa, pero sin dejar de usar lenguaje ensamblador para desarrollar el software para mostrar una cadena. El ejemplo 7-4 ilustra un programa que crea un procedimiento para mostrar una cadena de caracteres. Este procedimiento se llama cada vez que se muestra una cadena en el programa. Observe que este programa muestra una cadena en cada línea, a diferencia del ejemplo 7-3.

EJEMPLO 7-4

```

//Un programa que ilustra un procedimiento en lenguaje ensamblador que
//muestra cadenas de caracteres de lenguaje C

char cadena1[] = "Éste es mi primer programa de prueba que usa _asm.";
char cadena2[] = "Ésta es la segunda línea en este programa.";
char cadena3[] = "Ésta es la tercera.";

void main(void)
{
    Cad    (cadena1);
    Cad    (cadena2);
    Cad    (cadena3);
}

Cad (char *dir_cadena)
{
    _asm
    {
        mov     bx, dir_cadena          ;obtiene la dirección de la cadena
        mov     ah, 2

        sup:
        mov     dl, [bx]
        inc     bx
        cmp     al, 0                  ;si es nulo
        je      inf
        int    21h                     ;muestra el carácter
        jmp    sup

        inf:
        mov     dl,13                 ;muestra CR + LF
        int    21h
        mov     dl, 10
        int    21h
    }
}

```

Uso de estructuras de datos

En la mayoría de los programas, las estructuras de datos son una parte importante. En esta sección se mostrará cómo crear una interfaz entre una estructura de datos creada en C y una sección en lenguaje ensamblador que manipule los datos en la estructura. El ejemplo 7-5 ilustra un programa corto que utiliza

una estructura de datos para almacenar nombres, edades y salarios. Después el programa muestra cada una de las entradas mediante el uso de unos cuantos procedimientos en lenguaje ensamblador. Aunque el procedimiento de cadena muestra una cadena de caracteres, como se muestra en el ejemplo 7-4, no se muestra una combinación de retorno de carro/avance de línea; en vez de ello se muestra un espacio. El procedimiento Crlf muestra una combinación de retorno de carro/avance de línea. El procedimiento Nume muestra el entero.

EJEMPLO 7-5

```
//Programa que ilustra un procedimiento en lenguaje ensamblador que
//muestra el contenido de una estructura de datos en C.

//Una estructura de datos simple

typedef struct registros
{
    char primer_nombre[16];
    char apellido[16];
    int edad;
    int salario;
} REGISTRO;

//Llenado de algunos registros

REGISTRO registro[4] =
{ {"Bill", "Boyd", 56, 23000},
  {"Page", "Turner", 32, 34000},
  {"Bull", "Dozer", 39, 22000},
  {"Hy", "Society", 48, 62000}
};

//Programa

void main(void)
{
    int apunt = -1
    while (apunt++ < 3)
    {
        Cad(registro[apunt].apellido);
        Cad(registro[apunt].primer_nombre);
        Nume(registro[apunt].edad);
        Nume(registro[apunt].salario);
        Crlf;
    }
}

Cad (char *dir_cadena[])
{
    _asm
    {
        mov     bx, dir_cadena
        mov     ah, 2
        sup:
        mov     dl, [bx]
        inc     bx
        cmp     al, 0
        je      inf
        int    21h
        jmp     sup
        inf:
        mov     al, 20h
        int    21h
    }
}
```

```

Crlf()
{
    __asm
    {
        mov     ah, 2
        mov     dl, 13
        int     21h
        mov     dl, 10
        int     21h
    }
}

Nume (int temp)
{
    __asm
    {
        mov     ax, temp
        mov     bx, 10
        push   bx
L1:
        mov     dx, 0
        div     bx
        push   dx
        cmp     ax, 0
        jne     L1
L2:
        pop     dx
        cmp     dl, bl
        je      L3
        mov     ah, 2
        add     dl, 30h
        int     21h
        jmp     L2
L3:
        mov     dl, 20h
        int     21h
    }
}

```

Un ejemplo de un programa con lenguaje mixto

Para ver cómo se puede aplicar esta técnica a cualquier programa, el ejemplo 7-6 muestra cómo el programa puede realizar ciertas operaciones en lenguaje ensamblador y ciertas en C. Aquí, la única porción del programa en lenguaje ensamblador está representada por los procedimientos Muestranum, que muestra un entero, y Leenum, que lee un entero. El programa en el ejemplo 7-6 no realiza ningún intento por detectar o corregir errores. Además, el programa funciona en forma correcta sólo si el resultado es positivo y menor de 64 K. Observe que este ejemplo utiliza lenguaje ensamblador para realizar las operaciones de E/S; la porción en C realiza todas las demás operaciones para formar el intérprete de comandos del programa.

EJEMPLO 7-6

```

/*
Un programa que funciona como una calculadora simple para realizar operaciones de
suma, resta, multiplicación y división. El formato es X <oper> Y =.
*/
int temp;

void main(void)
{

```

```

int temp1, oper;
while (1)
{
    oper = Leenum();           //obtiene el primer número y la operación
    temp1 = temp;
    if ( Leenum() == '=' )    //obtiene el segundo número
    {
        switch (oper)
        {
            case '+':
                temp += temp1;
                break;
            case '-':
                temp = temp1 - temp;
                break;
            case '/':
                temp = temp1 / temp;
                break;
            case '*':
                temp *= temp1;
                break;
        }
        Muestranum(temp);      //muestra el resultado
    }
    else
        Break;
}
}

int Leenum()
{
    int a;
    temp = 0;
    _asm
    {
        Leenum1:
        mov     ah, 1
        int    21h
        cmp     al, 30h
        jb     Leenum2
        cmp     al, 39h
        ja     Leenum2
        sub     al, 30h
        shl     temp, 1
        mov     bx, temp
        shl     temp, 2
        add     temp, bx
        add     byte ptr temp, al
        adc     byte ptr temp+1, 0
        jmp     Leenum1
        Leenum2:
        mov     ah, 0
        mov     a, ax
    }
    return a;
}

Muestranum (int MuestranumTemp)
{
    _asm
    {
        mov     ax, MuestranumTemp
        mov     bx, 10
        push    bx
        Muestranum1:
        mov     dx, 0
    }
}

```

```

        div      bx
        push    dx
        cmp     ax, 0
        jne    Muestranum1
Muestranum2:
        pop     dx
        cmp     dl, bl
        je     Muestranum3
        add     dl, 30h
        mov     ah, 2
        int     21h
        jmp     Muestranum2
Muestranum3:
        mov     dl, 13
        int     21h
        mov     dl, 10
        int     21h
    }
}

```

7-2**USO DEL LENGUAJE ENSAMBLADOR CON VISUAL C/C++ PARA
APLICACIONES DE 32 BITS**

Existe una importante diferencia entre las aplicaciones de 16 y de 32 bits. Las aplicaciones de 32 bits se escriben mediante el uso de Microsoft Visual C/C++ para Windows y las aplicaciones de 16 bits se escriben mediante el uso de Microsoft C++ para DOS. La principal diferencia es que Visual C/C++ para Windows es más común actualmente, pero Visual C/C++ no puede llamar fácilmente a las funciones del DOS tales como INT 21H. Se recomienda que las aplicaciones integradas que no requieren de una interfaz visual se escriban en C o C++ de 16 bits, y que las aplicaciones que incorporan Windows o Windows CE (disponible para su uso en un dispositivo ROM o Flash¹ para aplicaciones integradas) utilicen Visual C/C++ de 32 bits para Windows.

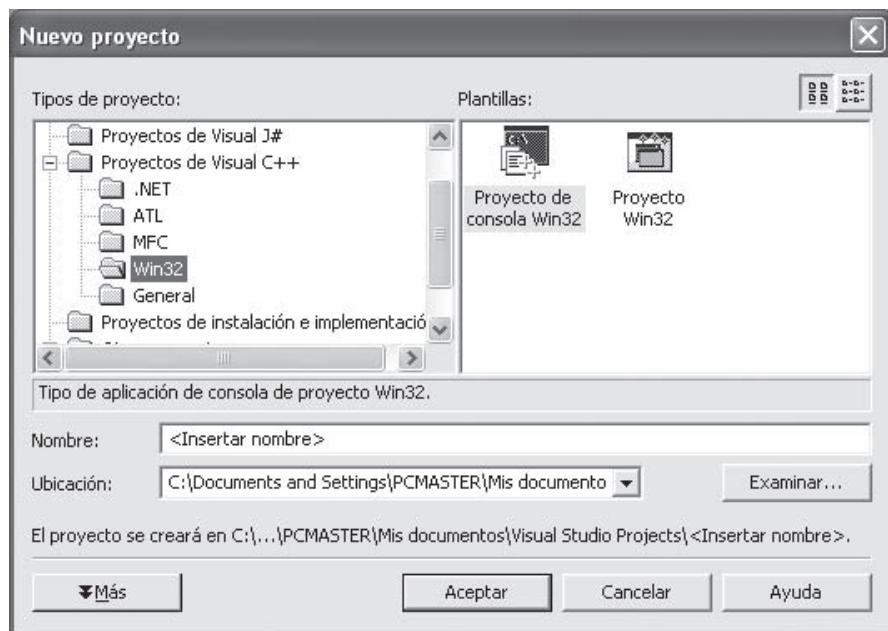
Una aplicación de 32 bits se escribe mediante el uso de cualquiera de los registros de 32 bits, y el espacio de memoria se limita en esencia a 2 Gbytes para Windows. La única diferencia es que no se pueden usar las llamadas a las funciones del DOS; en vez de ello se utilizan las funciones de consola getch() o getche() y putch del lenguaje C/C++, disponibles para su uso en aplicaciones de consola del DOS. Las aplicaciones integradas utilizan instrucciones directas en lenguaje ensamblador para acceder a los dispositivos de E/S en un sistema integrado. En la interfaz de Visual, todas las operaciones de E/S se manejan mediante la estructura de trabajo del sistema operativo Windows.

Un ejemplo que utiliza la E/S de consola para acceder al teclado y a la pantalla

El ejemplo 7-7 ilustra un ejemplo sencillo que utiliza los comandos de E/S de consola para leer y escribir datos de la consola. Para escribir esta aplicación (suponiendo que está disponible Visual Studio .NET 2003), selecciona una aplicación de consola WIN32 en la opción de nuevo proyecto (vea la figura 7-1). En vez de utilizar la biblioteca stdio.h como de costumbre, utilizaremos la biblioteca conio.h. Este programa de ejemplo muestra cualquier número entre 0 y 1000, en todas las bases numéricas desde la base 2 hasta la base 16. Observe que el programa principal no se llama main como en versiones anteriores de C/C++, sino _tmain en la versión actual de Visual C/C++ cuando se utiliza con una aplicación de consola. La variable argc es el contador de argumentos que se pasa al procedimiento _tmain desde la línea de comandos, y argv[] es un arreglo que contiene las cadenas de argumentos de la línea de comandos.

¹ Flash es una marca registrada de Intel Corporation.

FIGURA 7-1 Captura de pantalla del menú Nuevo proyecto de Visual Studio .NET 2003 para escribir aplicaciones de consola en Visual C/C++.



EJEMPLO 7-7

```
//Programa que muestra cualquier número en todas las bases numéricas
//entre la base 2 y la base 16.

#include "stdafx.h"
#include <conio.h>

char *bufer = "Escriba un número entre 0 y 1000: ";
char *bufer1 = "Base :";
int a, b = 0;

int _tmain(int argc, _TCHAR* argv[])
{
    int i;
    _cputs(bufer);
    a = _getche();
    while ( a >= '0' && a <= '9' )
    {
        _asm sub a, 30h;
        b = b * 10 + a;
        a = _getche();
    }
    _putch(10);
    _putch(13);
    for ( i = 2; i < 17; i++ )
    {
        _cputs(bufer1);
        muestra(10, i);
        _putch(' ');
        muestra(i, b);
        _putch(10);
        _putch(13);
    }
    getche(); //espera cualquier tecla
    return 0;
}
```

```

void muestra(int base, int datos)
{
    int temp;
    _asm
    {
        mov    eax, datos
        mov    ebx, base
        push   ebx
muestra1:
        mov    edx, 0
        div    ebx
        push   edx
        cmp    eax, 0
        jne    muestra1
muestra2:
        pop    edx
        cmp    ebx, edx
        je     muestra4
        add    dl, 30h
        cmp    dl, 39h
        jbe    muestra3
        add    dl, 7
muestra3:
        mov    temp, edx
    }
    _putch(temp);
    _asm    jmp    muestra2;
muestra4:;
}

```

Este ejemplo presenta una buena mezcla de comandos en lenguaje ensamblador y en lenguaje C/C++. El procedimiento muestra (base, datos) realiza la mayor parte del trabajo en este programa. Permite mostrar cualquier entero (sin signo) en cualquier base numérica, que puede estar entre la base 2 y la base 36. El límite superior se debe a que ya no hay letras para mostrar bases numéricas después de la letra Z. Si necesita realizar conversiones a bases numéricas más grandes, hay que desarrollar un nuevo esquema para las bases mayores de 36. Tal vez podrían utilizarse las letras de la a hasta la z para las bases 37 a 52. El ejemplo 7-7 muestra el número que recibe como entrada en una base numérica entre la base 2 y la base 16.

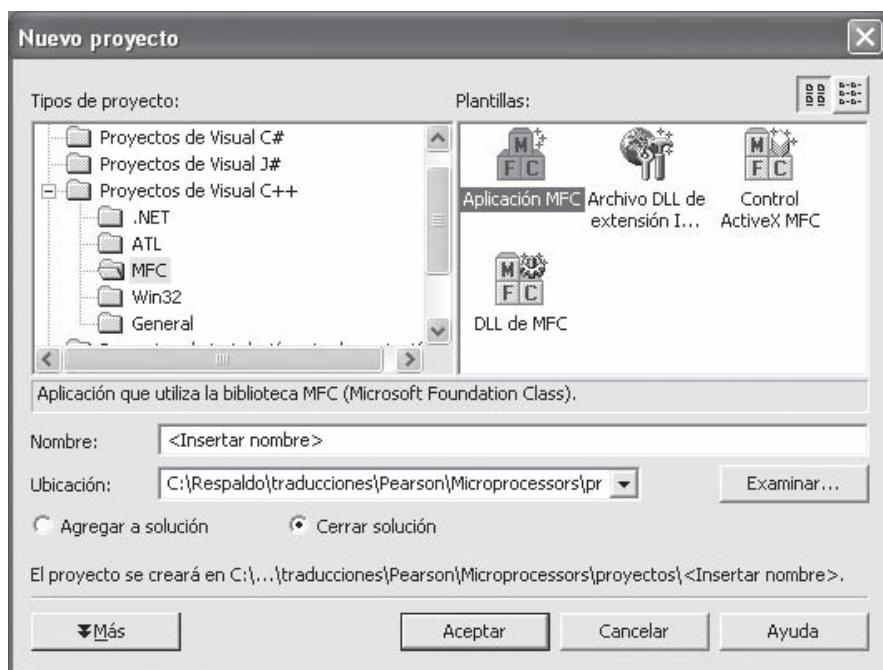
Direccionamiento directo de los puertos de E/S

Si se escribe un programa que deba tener acceso a un número de puerto real, podemos usar los comandos de E/S de consola tales como el comando `_inp(puerto)` para introducir datos tipo byte, y el comando `_outp(puerto,datos_byte)` para enviar datos tipo byte. Cuando se escribe software para la computadora personal es raro direccionar un puerto de E/S de forma directa, pero cuando se escribe software para un sistema integrado, esta operación es frecuente. El lenguaje ensamblador es una alternativa al uso de los comandos `_inp` y `_outp`, y es más eficiente en la mayoría de los casos. Tenga en cuenta que no se puede tener acceso a los puertos de E/S en el entorno Windows si se utiliza Windows NT, Windows 2000 o Windows XP. La única manera de acceder a los puertos de E/S en estos sistemas es mediante el desarrollo de un controlador del núcleo (kernel). En este punto del libro no sería práctico desarrollar un controlador de ese tipo. Si utiliza Windows 98 o incluso Windows 95, puede utilizar las instrucciones `inp` y `outp` para acceder a los puertos de E/S de forma directa.

Desarrollo de una aplicación para Windows en Visual C/C++

En esta sección se mostrará cómo utilizar Visual C++ para desarrollar una aplicación basada en un cuadro de diálogo para la biblioteca Microsoft Foundation Classes. Esta biblioteca (MFC) es una colección de clases que nos permite utilizar la interfaz de Windows sin mucha dificultad. La aplicación más fácil de desarrollar es la aplicación de cuadro de diálogo que presentamos aquí. Este tipo básico de aplicación puede usarse para programar y probar todos los ejemplos de software que se incluyen en este libro de texto, mediante el entorno de programación Visual C++.

FIGURA 7-2 Captura de pantalla del Asistente para nuevos proyectos de Visual Studio .NET 2003.



Para crear una aplicación de Visual C++ basada en un cuadro de diálogo, inicie Visual Studio .NET 2003. Haga clic en el menú Archivo | Nuevo | Proyecto. La figura 7-2 muestra lo que aparece cuando selecciona el tipo de aplicación MFC en los Proyectos de Visual C++. Escriba un nombre para el proyecto y seleccione una ruta apropiada para el mismo, después haga clic en Aceptar.

La figura 7-3 ilustra el Asistente para aplicaciones MFC, para un proyecto llamado Primero, que es el nombre que se seleccionó en el cuadro de diálogo Nuevo proyecto de la figura 7-2. El Asistente para aplicaciones tiene muchas características, pero para una aplicación de cuadro de diálogo todo lo que se necesita es hacer clic en Tipo de aplicación, seleccionar la opción Basada en cuadros de diálogo, y eso es todo. Ahora haga clic en Finalizar.

FIGURA 7-3 Captura de pantalla del Asistente para aplicaciones MFC, para el proyecto Primero.

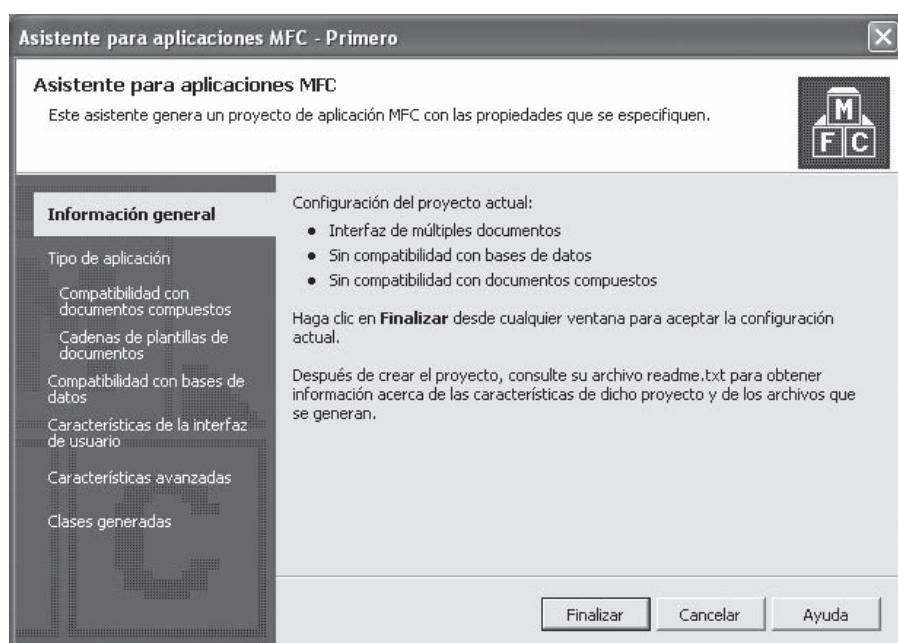
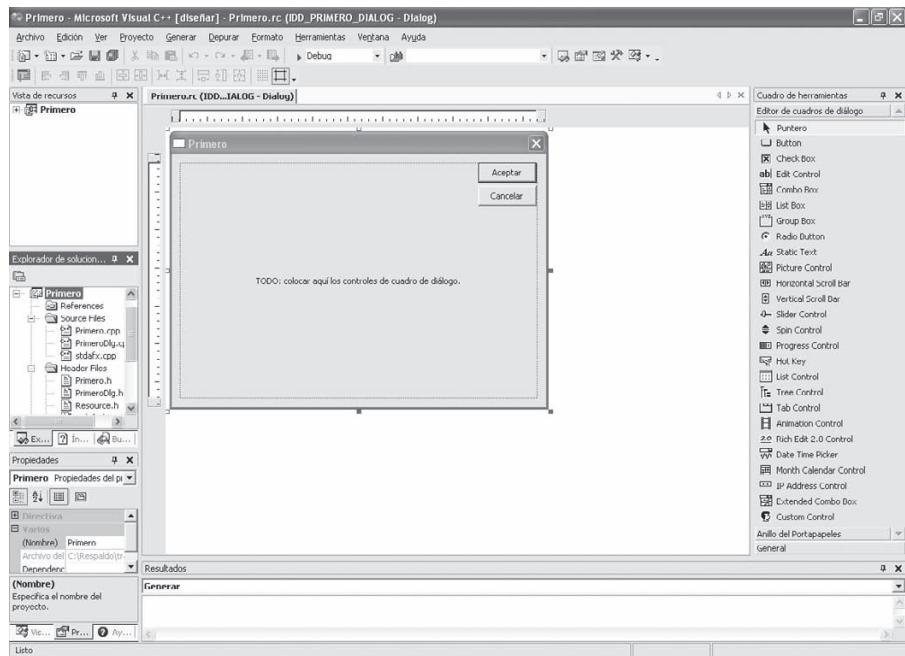


FIGURA 7-4 Captura de pantalla de la aplicación basada en cuadro de diálogo Primero.



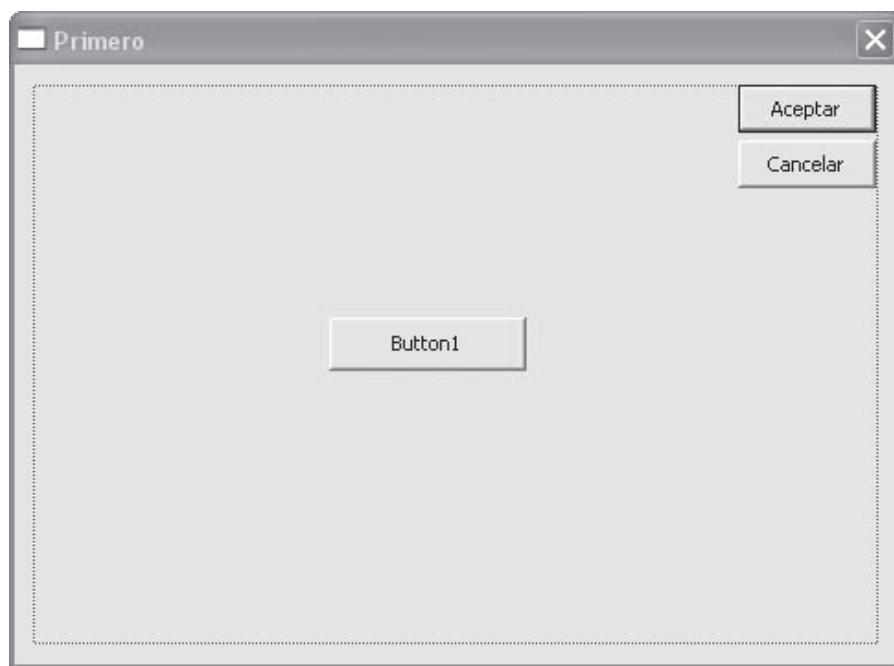
Después de unos momentos deberá aparecer la pantalla que se muestra en la figura 7-4. En la sección de en medio está el cuadro de diálogo creado por esta aplicación. Para probar la aplicación, sólo encuentre la flecha ▶ de color azul debajo del menú Herramientas que está en la parte superior de la pantalla, y haga clic en ella para compilar, enlazar y ejecutar la aplicación de cuadro de diálogo. (Responda que sí a la pregunta “Las configuraciones del proyecto no están actualizadas. ¿Le gustaría generarlas?”) Haga clic en el control Aceptar para salir de la aplicación. Con esto habrá creado y probado su primera aplicación de Visual C++ basada en un cuadro de diálogo.

En la figura 7-4 se muestran varios elementos que son importantes para la creación y el desarrollo de programas. El margen derecho de la pantalla contiene un cuadro de herramientas con objetos que pueden colocarse en la aplicación. El margen izquierdo contiene tres ventanas: Vista de recursos, Explorador de soluciones y Propiedades. Las fichas ubicadas en la parte inferior de la ventana Explorador de soluciones permiten mostrar las ventanas de solución, índice o búsqueda en esta área. Las fichas en la parte inferior de la ventana Propiedades permiten mostrar las clases, las propiedades, la ayuda dinámica o la salida en esta ventana. Su pantalla podría parecerse o no a la que se muestra en la figura 7-4, ya que puede modificarse mediante la selección de la ficha Perfiles cuando se inicia Visual Studio .NET 2003.

Para crear una aplicación simple para el cuadro de diálogo, primero haga clic en la etiqueta “TODO: colocar aquí los controles de cuadro de diálogo”, después haga clic en la tecla Supr del teclado para eliminarla. Windows es un sistema controlado por eventos, por lo que necesita crear algún objeto en la aplicación de cuadro de diálogo para iniciar un evento. Este objeto podría ser un botón o casi cualquier otro control que se pueda seleccionar del cuadro de herramientas. Haga clic en el control de botón cerca de la parte superior del cuadro de herramientas, con lo que se seleccionará el botón. Ahora mueva el puntero del ratón (no arrastre el botón) hacia la aplicación de cuadro de diálogo que está en medio de la pantalla y dibuje el botón cerca del centro (vea la figura 7-5).

Una vez colocado el botón en la pantalla hay que agregar un manejador de eventos a la aplicación, para que maneje el acto de oprimir o de hacer clic en el botón. Para seleccionar los manejadores de eventos, vaya a la ventana Propiedades que está en el margen izquierdo de la pantalla y haga clic en el ícono de rayo. Localice el objeto IDC_BUTTON1 que está debajo del ícono de rayo y haga clic en el signo positivo para expandir el listado y mostrar los eventos disponibles para Button1. Localice BN_CLICKED, que es el manejador de eventos que responde cuando se hace clic en el botón. A la

FIGURA 7-5 Captura de pantalla del botón colocado en el centro de la aplicación de cuadro de diálogo.



derecha de BN_CLICKED podrá ver una flecha. Haga clic en la flecha y seleccione <Agregar>OnBnClickedButton1 para agregar un manejador para el evento del botón oprimido. Al hacer esto, la pantalla mostrará el procedimiento OnBnClickedButton1 para el evento del botón oprimido. En el ejemplo 7-8 se muestra este procedimiento. Para probar el botón, cambie el software del ejemplo 7-8 por el software del ejemplo 7-9(a). Haga clic en la flecha azul para compilar, enlazar y ejecutar la aplicación de cuadro de diálogo. Después haga clic en el botón Button1 cuando la aplicación se esté ejecutando. La etiqueta en Button1 cambiará a “WOW, hola programador!” si hizo el botón lo suficientemente amplio. Ahora tiene su primera aplicación funcional, pero no utiliza ningún código ensamblador. En el ejemplo 7-9(a) se utilizó la función SetDlgItemText(ID de control, cadena) para cambiar el texto mostrado en IDC_BUTTON1. En el ejemplo 7-9(b) aparece una variación que utiliza un objeto de cadena de caracteres (CString).

EJEMPLO 7-8

```
void CPrimeroDlg::OnBnClickedButton1()
{
    // TODO: Agregue aquí su código de controlador de notificación de control
}
```

EJEMPLO 7-9

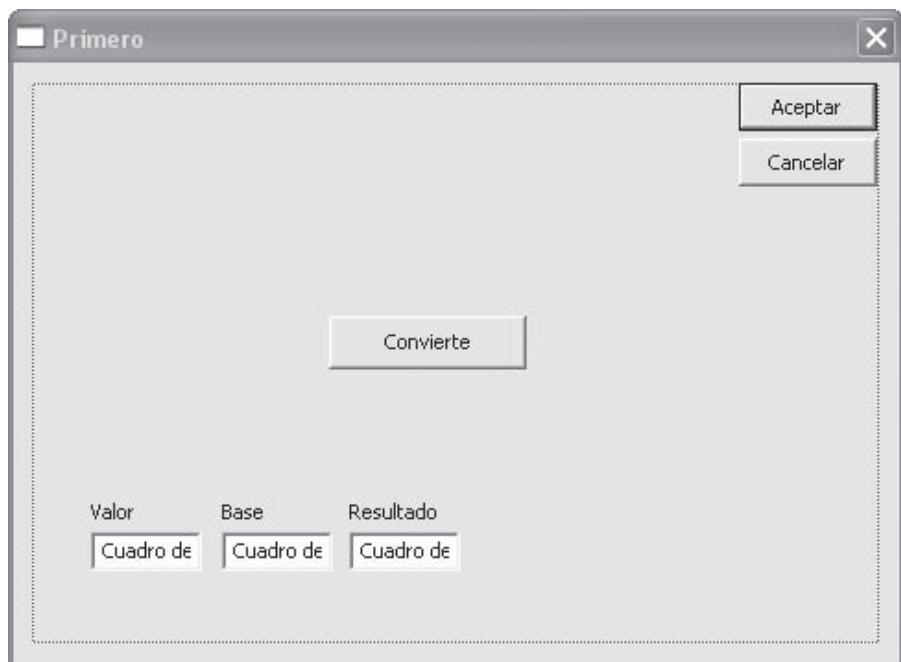
```
//Versión (a)

void CPrimeroDlg::OnBnClickedButton1()
{
    SetDlgItemText( IDC_BUTTON1, "WOW, hola programador!" );
}

//Versión (b)

void CPrimeroDlg::OnBnClickedButton1()
{
    CString cad1 = "WOW, hola programador!";
    SetDlgItemText( IDC_BUTTON1, cad1 );
}
```

FIGURA 7-6 Captura de pantalla de una aplicación completa que muestra cualquier número en cualquier base numérica (raíz).



Ahora que ha escrito una aplicación simple, podrá modificarla para ilustrar una aplicación más complicada, como se muestra en la figura 7-6. La leyenda del botón se ha cambiado a la palabra Convierte. Para crear esta pantalla, seleccione el cuadro de diálogo en el margen superior izquierdo (Vista de recursos) y haga clic en los signos positivos al lado de Primero, de Primero.rc y por último de Dialog. Ahora haga doble clic en IDD_PRIMERO_DIALOG para mostrar otra vez el cuadro de diálogo en medio de la pantalla. Para cambiar la leyenda en el objeto Button1 haga clic en el botón, después vaya a la ventana Propiedades (margen inferior izquierdo) y haga clic en el botón de propiedades.■ Ahora vaya a la entrada Caption en la lista de propiedades que aparece y cambie la leyenda a Convierte. Observe en la figura 7-6 que hay tres controles de texto estáticos y tres controles de edición debajo y a la izquierda del botón Convierte. Encontrará estos dos elementos en la parte superior del cuadro de herramientas. Dibújelos en la pantalla; trate de hacerlo en los mismos lugares que se muestran en la figura 7-6. Tendrá que ir a la ventana Propiedades de cada control de texto estático para cambiar las leyendas como se indica.

Nuestro objetivo en este ejemplo es mostrar cualquier número decimal que se introduzca en el cuadro del valor en cualquier raíz (base numérica) que se introduzca en el cuadro de la raíz. El resultado aparece en el cuadro de resultado cuando se hace clic en el botón Convierte. Para obtener el valor de un control de edición utilice la función GetDlgItemInt(Id de control). Esta función devuelve el valor entero del control de edición. La parte difícil de este ejemplo es la conversión de base 10 a cualquier base numérica. Si escribiéramos todo el programa completo en C++, aparecería como en el ejemplo 7-10. Observe que sólo se utiliza la función OnBnClickedButton1 en este ejemplo. La función de entero a ASCII llamada itoa(int número, char cadena, int base) convierte un entero a cualquier base, y también devuelve una cadena al procedimiento que la llamó. Este software supone que los tres cuadros de edición son (de izquierda a derecha) IDC EDIT1, IDC EDIT2 e IDC EDIT3.

EJEMPLO 7-10

Como este libro es sobre lenguaje ensamblador, no vamos a utilizar la función itoa; y por una buena razón, ya que la función es bastante grande. Para ver qué tan grande es, puede colocar un punto de interrupción en el software, a la izquierda de la función SetDlgItemText(), si hace clic con el botón izquierdo del ratón en la barra gris que está a la izquierda de la línea de código. A continuación aparecerá un círculo café (el punto de interrupción). Si ejecuta el programa, se detendrá en este punto y entrará al modo de depuración, para que pueda verlo en forma de lenguaje ensamblador. El ejemplo 7-11 muestra parte del código que Microsoft inserta en un programa para la función itoa. Como puede ver, ésta es una función muy grande y que consume mucho tiempo. Para mostrar el código desensamblado, ejecute el programa hasta que se detenga y luego vaya al menú Depurar | Ventanas. En el menú Ventanas encontrará la palabra “Desensamblador”.

EJEMPLO 7-11

```

static void __cdecl xtoa (
    unsigned long val,
    char *buf,
    unsigned radix,
    int is_neg
)
{
102194E0  push      ebp
102194E1  mov       ebp,esp
102194E3  sub       esp,10h
    char *p;           /* apuntador para recorrer cadena */
    char *firstdig;   /* apuntador al primer digito */
    char temp;         /* char temporal */
    unsigned digval;   /* valor del digito */
    p = buf;
102194E6  mov       eax,dword ptr [buf]
102194E9  mov       dword ptr [p],eax

    if (is_neg) {
102194EC  cmp       dword ptr [is_neg],0
102194F0  je        xtoa+29h (10219509h)
        /* como es negativo, se imprime '-' en pantalla y se niega */
        *p++ = '-';
102194F2  mov       ecx,dword ptr [p]
102194F5  mov       byte ptr [ecx],2Dh
102194F8  mov       edx,dword ptr [p]
102194FB  add       edx,1
102194FE  mov       dword ptr [p],edx
        val = (unsigned long)(-(long)val);
10219501  mov       eax,dword ptr [val]
10219504  neg       eax
10219506  mov       dword ptr [val],eax
    }

    firstdig = p;          /* almacena apuntador al primer digito */
10219509  mov       ecx,dword ptr [p]
1021950C  mov       dword ptr [firstdig],ecx
    do {
        digval = (unsigned) (val % radix);
1021950F  mov       eax,dword ptr [val]
10219512  xor       edx,edx
10219514  div       eax,dword ptr [radix]
10219517  mov       dword ptr [digval],edx
        val /= radix;        /* obtiene el siguiente digito */
1021951A  mov       eax,dword ptr [val]
1021951D  xor       edx,edx
1021951F  div       eax,dword ptr [radix]
10219522  mov       dword ptr [val],eax

        /* convierte a ascii y almacena */
        if (digval > 9)
10219525  cmp       dword ptr [digval],9
10219529  jbe       xtoa+61h (10219541h)
        *p++ = (char) (digval - 10 + 'a'); /* una letra */
    }
}

```

```

1021952B mov edx,dword ptr [digval]
1021952E add edx,57h
10219531 mov eax,dword ptr [p]
10219534 mov byte ptr [eax],dl
10219536 mov ecx,dword ptr [p]
10219539 add ecx,1
1021953C mov dword ptr [p],ecx
    else
1021953F jmp xtoa+75h (10219555h)
    *p++ = (char) (digval + '0'); /* un digito */
10219541 mov edx,dword ptr [digval]
10219544 add edx,30h
10219547 mov eax,dword ptr [p]
1021954A mov byte ptr [eax],dl
1021954C mov ecx,dword ptr [p]
1021954F add ecx,1
10219552 mov dword ptr [p],ecx
} while (val > 0);
10219555 cmp dword ptr [val],0
10219559 ja xtoa+2Fh (1021950Fh)
/* ahora tenemos el digito del numero en el bufer, pero en orden inverso.
Por ende lo invertimos ahora. */

*p-- = '\0'; /* termina cadena; p apunta al ultimo digito */
1021955B mov edx,dword ptr [p]
1021955E mov byte ptr [edx],0
10219561 mov eax,dword ptr [p]
10219564 sub eax,1
10219567 mov dword ptr [p],eax

do {
    temp = *p;
1021956A mov ecx,dword ptr [p]
1021956D mov dl,byte ptr [ecx]
1021956F mov byte ptr [temp],dl
    *p = *firstdig;
10219572 mov eax,dword ptr [p]
10219575 mov ecx,dword ptr [firstdig]
10219578 mov dl,byte ptr [ecx]
1021957A mov byte ptr [eax],dl
    *firstdig = temp; /* intercambia *p y *firstdig */
1021957C mov eax,dword ptr [firstdig]
1021957F mov cl,byte ptr [temp]
10219582 mov byte ptr [eax],cl
    --p;
10219584 mov edx,dword ptr [p]
10219587 sub edx,1
1021958A mov dword ptr [p],edx
    ++firstdig; /* avanza a los siguientes dos digitos */
1021958D mov eax,dword ptr [firstdig]
10219590 add eax,1
10219593 mov dword ptr [firstdig],eax
} while (firstdig < p); /* repite hasta llegar a la mitad */
10219596 mov ecx,dword ptr [firstdig]
10219599 cmp ecx,dword ptr [p]
1021959C jb xtoa+8Ah (1021956Ah)
}
1021959E mov esp,ebp
102195A0 pop ebp
102195A1 ret

```

Como puede ver, ésta es una cantidad considerable de código que puede reducirse de manera significativa si se reescribe mediante el uso del ensamblador en línea. El ejemplo 7-12 muestra la versión en lenguaje ensamblador de la función OnBnClickedButton1. Esta función es mucho más corta y se ejecuta muchas veces más rápido que la función del ejemplo 7-11. Tal vez sea conveniente que cambie la función OnBnClickedButton1 y vea si la aplicación de cuadro de diálogo funciona correctamente. Este ejemplo recalca la ineficiencia del código generado por un lenguaje de alto nivel, lo cual no siempre puede ser importante, pero para muchos casos se requiere de código reducido y eficiente, que

sólo puede escribirse en lenguaje ensamblador. Mi opinión es que, a medida que se llegue a una meta en la velocidad de los procesadores, se escribirán más cosas en lenguaje ensamblador. Además, las nuevas instrucciones como MMX y SEC no están disponibles en lenguajes de alto nivel, ya que requieren de un excelente conocimiento funcional del código ensamblador.

EJEMPLO 7-12

```
void CFirstDlg::OnBnClickedButton1()
{
    char temp[10];
    int valor = GetDlgItemInt(IDC_EDIT1);
    int base = GetDlgItemInt(IDC_EDIT2);
    _asm
    {
        lea    ebx, temp          ;direcciona la cadena
        mov    ecx, base          ;obtiene la base
        push   ecx
        mov    eax, valor          ;prueba si es negativo
        test   eax, 80000000h
        jz     L1
        neg    eax
        mov    byte ptr[ebx], '-'  ;guarda el signo
        inc    ebx
L1:
        mov    edx, 0              ;divide entre la base
        div    ecx
        push   edx              ;almacena el residuo
        cmp    eax, 0              ;mientras que eax != 0
        jne   L1
L2:
        pop    edx              ;convierte en ASCII
        cmp    edx, ecx          ;si se completó
        je    L4
        add    dl, 30h
        cmp    dl, 39h
        jbe   L3
        add    dl, 7
L3:
        mov    [ebx], dl          ;almacena en la cadena
        jmp   L2
L4:
        mov    byte ptr [ebx+1], 0 ;almacena valor nulo al final de la cadena
    }
    SetDlgItemText(IDC_EDIT3, temp);
}
```

7-3

OBJETOS DE ENSAMBLADOR SEPARADOS

Como se mencionó en las secciones anteriores, el ensamblador en línea está limitado porque no puede utilizar secuencias MACRO ni las directivas condicionales del flujo del programa que se presentó en el capítulo 6. En ciertos casos es mejor desarrollar módulos en lenguaje ensamblador que después se enlacen con C++ para obtener una mejor flexibilidad. Esto es muy cierto si hay un equipo de programadores que desarrollan la aplicación. En esta sección se mostrarán los detalles acerca del uso de distintos objetos que se enlazan para formar un programa mediante el uso de lenguaje ensamblador y C++. La información que se cubrirá en este capítulo se aplica a Microsoft Visual C++ para Windows.

Cómo enlazar el lenguaje ensamblador con Visual C++

El ejemplo 7-13 muestra un procedimiento de modelo plano que se enlazará a un programa en C++. Para indicar que el módulo de ensamblador es un módulo de C++ se utiliza la letra C después de la palabra "flat" en la declaración del modelo. El enlace que se especifica mediante la letra C es el mismo para los

lenguajes C o C++. El modelo plano permite que el software en lenguaje ensamblador sea de cualquier longitud, hasta 2 Gbytes. Observe que el modificador .586 aparece antes de la declaración del modelo, lo que hace que el ensamblador genere código que funciona en el modo protegido de 32 bits. El procedimiento Invierte del ejemplo 7-13 acepta una cadena de caracteres de un programa de C++, invierte su orden y lo devuelve al programa de C++. Observe cómo este programa utiliza instrucciones condicionales del flujo del programa, que no están disponibles en el ensamblador en línea que se describieron en secciones anteriores de este capítulo. El módulo de lenguaje ensamblador puede tener cualquier nombre y puede contener más de un procedimiento, siempre y cuando cada procedimiento contenga una instrucción PUBLIC que defina el nombre del procedimiento como público. Cualquier parámetro que se transfiera entre el programa de C++ y el de lenguaje ensamblador se indica mediante una barra diagonal invertida después del nombre del procedimiento. Con esto se nombra el parámetro para el programa en lenguaje ensamblador (puede tener un nombre distinto en C++) y se indica el tamaño del parámetro. Lo único que no es diferente en el programa de C++ que hace la llamada y el programa ensamblador es el orden de los parámetros. En este ejemplo, el parámetro es un apuntador a una cadena de caracteres y el resultado se devuelve mediante la sustitución de la cadena original.

EJEMPLO 7-13

```

;
;Función externa que invierte el orden de una cadena de caracteres
;
.586                                ;selecciona el Pentium y el modelo de 32 bits
.model flat, C                         ;selecciona el modelo plano con enlace a C/C++
.stack 1024                            ;asigna espacio a la pila
.code                                 ;inicio del segmento de código

public Invierte                        ;define Invierte como función pública

Invierte proc uses esi, \
arreglochar:ptr                         ;define el procedimiento
                                         ;define el apuntador externo

    mov     esi, arreglochar            ;direcciona la cadena
    mov     eax, 0                      ;indica el final de la cadena
    push    eax

    .repeat                           ;mete todos los caracteres en la pila
        mov     al, [esi]
        push   eax
        inc    esi
    .until byte ptr [esi] == 0

    mov     esi, arreglochar            ;direcciona el inicio de la cadena

    .while eax != 0                  ;saca los caracteres en orden inverso
        pop    eax
        mov    [esi], al
        inc    esi
    .endw
    Ret

Invierte     endp
End

```

El ejemplo 7-14 muestra un programa en lenguaje C++ para aplicaciones de consola del DOS que utiliza el procedimiento en lenguaje ensamblador llamado Invierte. La instrucción EXTERN se utiliza para indicar que se va a utilizar un procedimiento externo llamado Invierte en el programa de C++. El nombre del procedimiento es susceptible al uso de minúsculas/mayúsculas, por lo que debe asegurarse que se escriba igual tanto en el módulo de lenguaje ensamblador como en el módulo de lenguaje C++. La instrucción EXTERN del ejemplo 7-14 muestra que el procedimiento externo en lenguaje ensamblador transfiere una cadena de caracteres al procedimiento y no devuelve datos. Si se devuelven datos del procedimiento en lenguaje ensamblador, se hace mediante un valor en el registro EAX en caso de que sean bytes, palabras o dobles palabras. Si se devuelven números de punto flotante, deben devolverse en la pila del coprocesador de punto flotante. Si se devuelve un puntero, debe estar en EAX.

EJEMPLO 7-14

```

/* Programa que invierte el orden de una cadena de caracteres */

#include "stdafx.h"
#include <stdio.h>
#include <conio.h>

#using <mscorlib.dll>

using namespace System;

extern "C" void Invierte(char *);

char chararreglo[15] = "Y que es esto?";

int _tmain()
{
    printf ("%s \n", chararreglo);
    Invierte (chararreglo);
    printf ("%s \n", chararreglo);
    getch();                                //espera a que el usuario vea el resultado
    return 0;
}

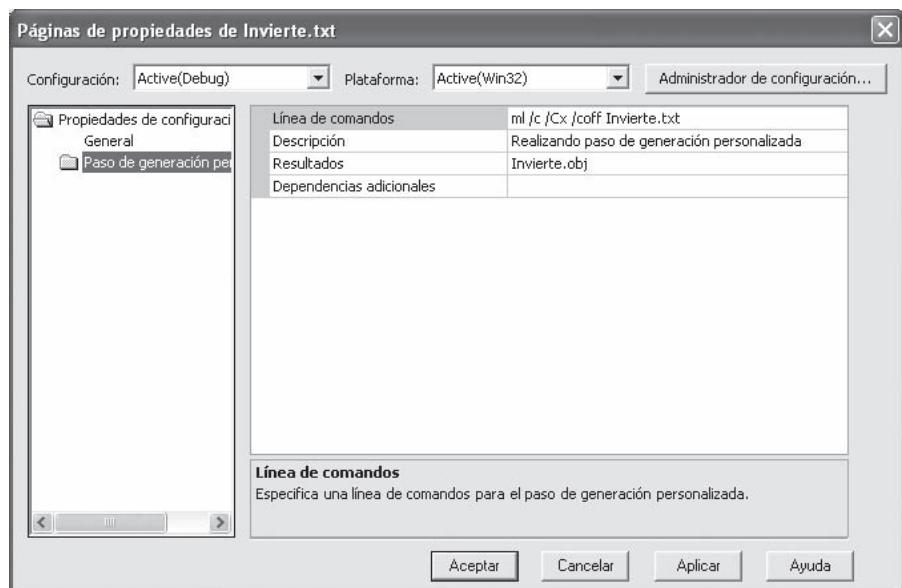
```

Una vez que se hayan escrito los programas de lenguaje C++ y de lenguaje ensamblador, el sistema de desarrollo de Visual C++ debe configurarse para enlazar estos dos programas. Para los procedimientos de enlazar y ensamblar se supondrá que el módulo de lenguaje ensamblador se llama Invierte.txt (no puede agregar una extensión de archivo .asm a la lista de archivos para incluir en un proyecto, por lo que se utilizará la extensión .txt y agregaremos un archivo .txt) y el módulo de lenguaje C++ se llama Main.cpp. Ambos módulos se almacenan en el directorio C:\PROYECTO\MIO o en algún otro directorio que escoja. Después de colocar los módulos en el mismo espacio de trabajo del proyecto, se utiliza el programa Programmer's WorkBench para editar los módulos de lenguaje ensamblador y de lenguaje C++.

Para configurar el programa Visual C++ para compilar, ensamblar y enlazar estos archivos, siga estos pasos:

1. Inicie Visual Studio y seleccione la opción Nuevo del menú Archivo.
 - (a) Seleccione la opción Nuevo Proyecto.
 - (b) Cuando aparezca el Asistente para aplicaciones, haga clic en Proyectos de Visual C++.
 - (c) Seleccione la opción Proyecto de consola Win32 y escriba el nombre Mio para el proyecto. Haga clic en Aceptar.
 - (d) Cuando aparezca el Asistente para aplicaciones Win32, haga clic en Finalizar.
2. Ahora verá el proyecto en la ventana Explorador de soluciones que está en el margen izquierdo al centro. Tendrá un solo archivo llamado Main.cpp, que es el archivo del programa en C++. Modifíquelo para que aparezca como en el ejemplo 7-14.
3. Para agregar el módulo de lenguaje ensamblador, haga clic con el botón derecho en la línea Source Files y seleccione la opción Agregar del menú. Seleccione la opción Agregar nuevo elemento de la lista. Navegue por la lista de tipos de archivos hasta que encuentre la opción Archivo de texto y selecciónela, después escriba el nombre de archivo como Invierte y haga clic en Abrir. A continuación se creará el módulo de ensamblador llamado Invierte.txt. Puede escribir el código ensamblador del ejemplo 7-13 en este archivo.
4. En el listado Source Files del Explorador de soluciones, haga clic con el botón derecho del ratón en Invierte.txt y seleccione la opción Propiedades. La figura 7-7 muestra lo que debe escribir en este asistente después de que haga clic en Paso de generación personalizada. Asegúrese de escribir el nombre del archivo objeto (Invierte.obj) en el cuadro Resultados y ml /c /Cx /coff Invierte.txt en el cuadro Línea de comandos. El archivo Invierte de lenguaje ensamblador se ensamblará y se incluirá en el proyecto.
5. Una vez que haya escrito los ejemplos 7-13 y 7-14, y que haya completado todos los pasos, el programa funcionará.

FIGURA 7-7 Uso del ensamblador para ensamblar un módulo en Visual C++.



Por último, puede ejecutar el programa. Haga clic en la flecha azul. Ahora deberá ver dos líneas de datos de texto ASCII en la pantalla. La primera línea debe estar en el orden correcto hacia delante y la segunda línea debe estar en orden invertido. Aunque esta aplicación es trivial, muestra cómo crear y enlazar el lenguaje C++ con el lenguaje ensamblador.

Ahora que tenemos una buena comprensión de cómo crear una interfaz entre el lenguaje ensamblador y el lenguaje C++, necesitamos un ejemplo más largo que utilice unos cuantos procedimientos en lenguaje ensamblador con un programa en lenguaje C++. El ejemplo 7-15 ilustra un paquete en lenguaje ensamblador que incluye un procedimiento (Explora) para comparar la entrada de un carácter con una tabla de búsqueda y devolver un número que indique la posición relativa en la tabla. Un segundo procedimiento (Busca) utiliza un número que recibe y lo regresa con una cadena de caracteres que representa código Morse. (El código no es importante, pero si le interesa, en la tabla 7-2 se lista el código Morse.)

EJEMPLO 7-15

```
.586
.model flat, C
.data

tabla db 2,1,4,8,4,10,3,4      ;ABCD
       db 1,0,4,2,3,6,4,0      ;EFGH
       db 2,0,4,7,3,5,4,4      ;IJKL
       db 2,3,2,2,3,7,4,6      ;MNOP
```

TABLA 7-2 El código Morse.

A	·—	J	———	S	...
B	—...—	K	—·—	T	—
C	—·—·	L	·—··	U	··—
D	—··	M	——	V	····—
E	.	N	—·	W	·——
F	··—·	O	—---	X	—··—
G	——·	P	·——·	Y	——·—
H	Q	——·—	Z	——··
I	..	R	·—..		

```

db  4,13,3,2,3,0,1,1      ;QRST
db  3,1,4,1,3,3,4,9      ;UVWX
db  4,11,4,12             ;YZ

.code

Public Explora
Public Busca

Explora proc uses ebx,\

char:dword

    mov  ebx, char
    .if bl >= 'a' && bl <= 'z'
        sub  bl, 20h
    .endif
    sub  bl, 41h
    add  bl, bl
    add  ebx, offset tabla
    mov  ax, word ptr[ebx]
    ret

Explora      endp

Busca proc uses ebx ecx,\

numer:dword, \
apunt:ptr

    mov  ebx, apunt
    mov  eax, numer
    mov  ecx, 0
    mov  cl, al
    .repeat
        shr   ah, 1
        .if carry?
            mov  byte ptr[ebx], '-'
        .else
            mov  byte ptr[ebx], '.'
        .endif
        inc   ebx
    .untilcxz
    mov  byte ptr[ebx], 0
    ret

Busca      endp
End

```

La tabla de búsqueda del ejemplo 7-15 contiene dos bytes para cada carácter entre A y Z. Por ejemplo, el código para una A es un 2 seguido de un 1. El 2 indica cuántos puntos o guiones se utilizan para formar el carácter en código Morse y el 1 (01) es el código para la letra A (.-), mientras que el 0 es un punto y el 1 es un guión. El procedimiento Explora accede a esta tabla de búsqueda para obtener el código Morse correcto, el cual se devuelve en AX como parámetro para la llamada en lenguaje C++. El resto del código ensamblador es mundano.

El ejemplo 7-16 lista el programa en C++, el cual llama a los dos procedimientos listados en el ejemplo 7-15. Como este software es simple de entender, no lo explicaremos.

EJEMPLO 7-16

```

//Morse.cpp : define el punto de entrada para la aplicación de consola.

#include "stdafx.h"
#include <iostream>
using namespace std;

extern "C" int Explora(int);
extern "C" void Busca(int, char *);

```

```

int _tmain(int argc, _TCHAR* argv[])
{
    int a = 0;
    char arreglochar[] = "Éste, es el truco!\n";
    char arreglochar1[10];
    while ( arreglochar[a] != '\n' )
    {
        if ( arreglochar[a] < 'A' || arreglochar[a] > 'z' )
            cout << arreglochar[a] << '\n';
        else
        {
            Busca ( Explora ( arreglochar[a] ), arreglochar1 );
            cout << arreglochar[a] << " = " << arreglochar1 << "\n";
        }
        a++;
    }
    cout << "Oprima Intro para salir!";
    cin.get();
    return 0;
}

```

Aunque los ejemplos que se presentaron aquí son para aplicaciones de consola, el mismo método de instruir a Visual Studio para que ensambla y enlace un módulo de lenguaje ensamblador se utiliza también para aplicaciones de Visual C/C++ para Windows. La principal diferencia es que las aplicaciones Windows no utilizan printf o cout. En el capítulo 8 se explicará cómo se pueden utilizar archivos de bibliotecas con Visual C++ y también se expondrán muchos ejemplos de programación.

Cómo agregar instrucciones nuevas de lenguaje ensamblador a los programas de C/C++

De vez en cuando, a medida que Intel introduce nuevos procesadores también se introducen nuevas instrucciones en lenguaje ensamblador. Estas nuevas instrucciones no pueden usarse en C++, a menos que desarrolle una macro para que C++ pueda incluirlas en el programa. Un ejemplo es la instrucción CPUID de lenguaje ensamblador. Esta instrucción no funcionará en un bloque _asm dentro de C++, ya que el ensamblador en línea no la reconoce. Otro grupo de instrucciones más recientes es el de las instrucciones MMX y SEC. Éstas tampoco se reconocen, pero para poder ilustrar cómo se agrega una nueva instrucción que no estaba en el ensamblador, se mostrará la técnica. Para utilizar cualquier instrucción que sea nueva, primero busque el código de lenguaje máquina en el apéndice B o en el sitio Web de Intel, www.intel.com. Por ejemplo, el código de lenguaje máquina para la instrucción CPUID es 0F A2. Esta instrucción de dos bytes puede definirse como una macro en C++, como se muestra en el ejemplo 7-17. Para utilizar la nueva macro en un programa de C++, todo lo que tenemos que hacer es escribir CPUID. La macro _emit almacena en el programa el byte que le sigue.

EJEMPLO 7-17

```
#define CPUID __asm __emit 0x0f __asm __emit 0xa2
```

7-4

RESUMEN

1. El ensamblador en línea se utiliza para insertar secuencias cortas y limitadas de lenguaje ensamblador en un programa de C++. La principal limitación del ensamblador en línea es que no puede utilizar macro secuencias o instrucciones condicionales del flujo del programa.
2. Hay dos versiones disponibles del lenguaje C++. Una está diseñada para aplicaciones DOS de consola de 16 bits y la otra para aplicaciones Windows de 32 bits. El tipo a elegir para una aplicación depende del entorno, pero en la mayoría de los casos los programadores de la actualidad utilizan Windows y la versión Visual C++ de 32 bits.

3. Las aplicaciones de 16 bits en lenguaje ensamblador utilizan los comandos de la función INT 21H del DOS para acceder a los dispositivos en el sistema. Las aplicaciones de 32 bits en lenguaje ensamblador no pueden acceder con eficiencia o facilidad a las llamadas de la función INT 21H del DOS, aún y cuando hay muchas disponibles.
4. El método más flexible y utilizado de crear una interfaz entre el lenguaje ensamblador y un programa en C++ es por medio de módulos separados de lenguaje ensamblador. La única diferencia es que estos módulos separados de lenguaje ensamblador deben definirse mediante el uso de la directiva C que va después de la instrucción .model, para definir el enlace de módulos como compatible con C/C++.
5. La instrucción PUBLIC se utiliza en un módulo de lenguaje ensamblador para indicar que el nombre del procedimiento es público y está disponible para usarse con otro módulo. En un módulo de lenguaje ensamblador, los parámetros externos se definen mediante el uso del nombre del procedimiento en la instrucción PROC. Los parámetros se devuelven desde el procedimiento de lenguaje ensamblador al procedimiento de C/C++ que hizo la llamada mediante el registro EAX.
6. Los módulos de lenguaje ensamblador se declaran como externos para el programa de C++ mediante el uso de la directiva extern. Si esta directiva va seguida de la letra C, se utiliza en un programa de lenguaje C/C++.
7. Al utilizar Visual Studio podemos instruirlo para que ensamble un módulo de lenguaje ensamblador si hacemos clic en Propiedades para el módulo y agregamos el programa de lenguaje ensamblador (ml /c /Cx /coff Nombrearchivo.txt) y el archivo de salida como archivo objeto (Nombrearchivo.obj) en el Paso de generación personalizada para el módulo.
8. Los módulos de lenguaje ensamblador pueden contener muchos procedimientos, pero no pueden contener programas que utilicen la directiva .startup.

7-5**PREGUNTAS Y PROBLEMAS**

1. ¿Soporta el ensamblador en línea macro, secuencias en lenguaje ensamblador?
2. ¿Puede un byte definirse en el ensamblador en línea mediante el uso de la directiva DB?
3. ¿Cómo se definen las etiquetas en el ensamblador en línea?
4. ¿Qué registros pueden usarse en lenguaje ensamblador (ya sea módulos en línea o enlazados) sin almacenarse?
5. ¿Qué registro se utiliza para devolver datos enteros de lenguaje ensamblador al programa en lenguaje C++ que hizo la llamada?
6. ¿Qué registro se utiliza para devolver datos tipo punto flotante de lenguaje ensamblador al programa en lenguaje C++ que hizo la llamada?
7. ¿Es posible utilizar la instrucción .if en el ensamblador en línea?
8. En el ejemplo 7-3, explique cómo la instrucción mov dl,cadena1[si] accede a los datos de cadena1.
9. En el ejemplo 7-3, explique por qué se sacó y se metió el registro SI.
10. Observe que en el ejemplo 7-5 no se utilizan bibliotecas de C++ (#include). ¿Piensa que el código compilado para este programa es más pequeño que para un programa que realice la misma tarea en lenguaje C++? ¿Por qué?
11. ¿Cuál es la principal diferencia entre las versiones de 16 y de 32 bits de C/C++ cuando se utiliza el ensamblador en línea?
12. ¿Puede utilizarse la instrucción INT 21H (que se utiliza para acceder a las funciones del DOS) en un programa mediante el uso de la versión de 32 bits del compilador de C/C++? Explique su respuesta.
13. ¿Para qué se utiliza la biblioteca #include<conio.h> de C/C++ en un programa?
14. Escriba un programa corto en C/C++ que utilice la función _getche() para leer una tecla y la función _putch() para mostrarla. El programa debe terminar si se escribe '@'.
15. ¿Podría una aplicación integrada que no sea escrita para la PC utilizar alguna vez la biblioteca conio.h?

16. En el ejemplo 7-7, ¿cuál es el propósito de la secuencia de instrucciones `_putch(10);` seguida de `_punch(13);?`
17. En el ejemplo 7-7, explique cómo se muestra un número en cualquier base numérica.
18. ¿Qué es más flexible en su aplicación, el ensamblador en línea o los módulos de lenguaje ensamblador que se enlazan a C++?
19. ¿Cuál es el propósito de una instrucción PUBLIC en un módulo de código ensamblador?
20. ¿Cómo se prepara un módulo de código ensamblador para usarse con el lenguaje C++?
21. En un programa en lenguaje C++, ¿qué indica la instrucción extern void Obtiene(int); acerca de la función Obtiene?
22. ¿Cómo se define una palabra de datos de 16 bits en C++?
23. ¿Qué es un control en un programa de Visual C++ y en dónde se obtiene?
24. ¿Qué es un evento en un programa de Visual C++ y qué es un manejador de eventos?
25. En el ejemplo 7-13, ¿qué tipo de parámetro es arreglochar?
26. ¿Puede utilizarse la pantalla de edición de Visual Studio C++ para escribir y editar un módulo de programación en lenguaje ensamblador?
27. ¿Cómo se indican los procedimientos escritos en lenguaje ensamblador para un programa en C++?
28. Muestre cómo podría agregarse la instrucción RDTSC (el código de operación es 0F 31) a un programa de C++ mediante la macro `_emit`.
29. En el ejemplo 7-15, explique qué tipo de datos utiliza la función Explora.
30. Escriba un módulo corto en lenguaje ensamblador que se utilice con C++ para desplazar en forma cíclica un número tres posiciones a la izquierda. Llame a su procedimiento DesplazaIzquierda3 y suponga que el número es tipo char de 8 bits (byte en ensamblador).
31. Repita la pregunta 30, pero escriba la misma función en C++ sin el ensamblador.
32. Escriba un módulo corto en lenguaje ensamblador que reciba un parámetro (tipo byte) y devuelva un resultado tipo byte al procedimiento que hizo la llamada. Su procedimiento deberá tomar este byte y convertirlo en letra mayúscula. Si aparece una letra minúscula o cualquier otra cosa, el byte no deberá modificarse.
33. ¿Cómo se ejecuta una aplicación MFC de Visual C++ desde Visual Studio?
34. ¿Qué son las propiedades en una aplicación de Visual C++?
35. ¿Qué es un control u objeto ActiveX?
36. Muestre cómo puede insertarse una sola instrucción en lenguaje ensamblador (tal como inc ptr) en un programa de Visual C++.

CAPÍTULO 8

Programación del microprocesador

INTRODUCCIÓN

En este capítulo se desarrollarán programas y técnicas de programación que utilizan el programa macro ensamblador MASM y el programa ensamblador en línea de Visual C++. En capítulos anteriores se explicó y demostró el funcionamiento del ensamblador MASM y el ensamblador en línea de Visual C++, pero aún quedan más características por aprender.

Algunas de las técnicas de programación que se explicarán en este capítulo incluyen macrosecuencias para MASM y módulos independientes de lenguaje ensamblador, la manipulación del teclado y la pantalla, módulos de programa, archivos de bibliotecas, el uso del ratón y el uso de temporizadores. Como introducción a la programación, este capítulo proporciona una buena cantidad de información. Usted podrá desarrollar programas con facilidad para la computadora personal, mediante el uso de MASM y el ensamblador en línea como un trampolín para las aplicaciones de Visual creadas para Windows.

OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

1. Utilizar el ensamblador MASM y el programa enlazador para crear programas que contengan más de un módulo.
2. Explicar el uso de las instrucciones EXTRN y PUBLIC en cuanto a su aplicación en la programación modular.
3. Establecer un archivo de biblioteca con subrutinas de uso común y aprenderá a utilizar el programa DUMPBIN.
4. Escribir y utilizar MACRO y ENDM para desarrollar macrosecuencias utilizadas con la programación lineal en módulos que se enlazan con código de C++.
5. Mostrar cómo se desarrollan los archivos de acceso secuencial y aleatorio para su uso en un sistema.
6. Desarrollar programas que utilicen manejadores de eventos para realizar tareas relacionadas con el teclado y la pantalla.
7. Utilizar instrucciones condicionales de lenguaje ensamblador en un programa.
8. Utilizar el ratón en ejemplos de programas.

8-1

PROGRAMACIÓN MODULAR

Muchos programas son demasiado grandes como para que una sola persona pueda desarrollarlos. Esto significa que, de manera rutinaria, los programas se desarrollan mediante equipos de programadores. El programa enlazador se incluye en Visual Studio para que los módulos de programación puedan enlazarse entre sí para formar un programa completo. El proceso de enlace también está disponible desde el símbolo de sistema de Windows. En esta sección se describirán el enlazador, la tarea de enlace, los archivos de bibliotecas y las instrucciones EXTRN y PUBLIC en cuanto a su aplicación en los módulos de programa y la programación modular.

El ensamblador y el enlazador

El **programa ensamblador** convierte un **módulo fuente** simbólico (archivo) en un **archivo objeto** hexadecimal. Incluso forma parte de Visual Studio y se encuentra en la carpeta C:\Archivos de programa\Microsoft Visual Studio .NET 2003\Vc7\bin. En capítulos anteriores se han visto muchos ejemplos de archivos fuente simbólicos escritos en lenguaje ensamblador. El ejemplo 8-1 muestra cómo se ensambla el diálogo ensamblador que aparece como un módulo fuente llamado NUEVO.ASM. Observe que este diálogo se utiliza con la versión 6.15 de la línea de comandos del DOS. La versión que viene con Visual C no funciona con programas DOS de 16 bits. Si se necesita un ensamblador y un enlazador de 16 bits, puede obtenerse en el Kit de desarrollo de controladores para Windows (DDK). Cada vez que cree un archivo fuente debe utilizar la extensión ASM, pero como se vio en el capítulo 7, eso no siempre es posible. Los archivos fuente se crean mediante el uso del Bloc de notas o de casi cualquier otro procesador de palabras o editor capaz de generar un archivo ASCII.

EJEMPLO 8-1

```
C:\masm611\BIN>ml nuevo.asm

Microsoft (R) Macro Assembler Version 6.11
Copyright (C) Microsoft Corp 1981-1993. All rights reserved.

Assembling: nuevo.asm

Microsoft (R) Segmented Executable Linker Version 5.60.220 Step 9 1994
Copyright (C) Microsoft Corp 1984-1993. All rights reserved.

Object Modules [.obj]: nuevo.obj
Run File [nuevo.exe]: "nuevo.exe"
List File [nul.map]: NUL
Libraries [.lib]:
Definitions File [nul.def]:
```

El programa ensamblador (ML) requiere el nombre del archivo fuente después del comando ML. En el ejemplo 8-1 se utiliza el modificador /F1 para crear un archivo de listado llamado NUEVO.LST. Aunque esto es opcional, se recomienda para que pueda verse la salida producida por el ensamblador, en caso de que haya que solucionar problemas. El archivo de listado fuente (.LST) contiene la versión ensamblada del archivo fuente y su equivalente en lenguaje máquina hexadecimal. El archivo de referencia cruzada (.CRF), que no se genera en este ejemplo, lista todas las etiquetas y la información pertinente que se requiera para la aplicación de referencias cruzadas. El comando ML también genera un archivo objeto que sirve como entrada para el programa enlazador. En muchos casos sólo necesitamos generar un archivo objeto, lo cual se logra mediante el modificador /c.

El **programa enlazador**, que se ejecuta como la segunda parte de ML, lee los archivos objeto que se crean mediante el programa ensamblador y los enlaza en un solo archivo ejecutable. Un **archivo ejecutable** se crea con la extensión EXE en el nombre del archivo. Para seleccionar un archivo ejecutable se escribe su nombre en el símbolo del DOS (C:). El archivo RANA.EXE es un ejemplo de archivo ejecutable, el cual se ejecuta si se escribe RANA en el intérprete de comandos.

Si un archivo es lo suficientemente corto (menos de 64 Kbytes de longitud), puede convertirse de un archivo ejecutable a un **archivo de comandos** (.COM). El archivo de comandos es un poco diferente

al archivo ejecutable en cuanto a que el programa debe originarse en la posición 0100H para poder ejecutarse. Esto significa que el programa no debe ser mayor de 64K-100H en longitud. El programa ML genera un archivo de comandos si se utiliza el modelo diminuto (TINY) con una dirección inicial de 100H. Los archivos de comandos se utilizan sólo en el DOS o si se necesita una versión binaria real (para un quemador de EPROM/FLASH). La principal ventaja de un archivo de comandos es que se carga del disco a la computadora de forma mucho más rápida que un archivo ejecutable. También requiere menos espacio de almacenamiento en disco que el archivo ejecutable equivalente.

El ejemplo 8-2 muestra el protocolo del programa enlazador cuando se utiliza para enlazar los archivos NUEVO, QUE y DONA. El enlazador también enlaza archivos de biblioteca (LIBS) de manera que puedan usarse los procedimientos incluidos en esos archivos con el archivo ejecutable enlazado. Para llamar al enlazador escriba LINK en el símbolo del sistema, como se muestra en el ejemplo 8-2. Antes de poder enlazar los archivos, primero deben ensamblarse y no deben contener errores. ML no sólo enlaza los archivos, sino que también los ensambla antes de enlazarlos.

EJEMPLO 8-2

```
C:\masm611\BIN>ml nuevo.asm que.asm dona.asm
Microsoft (R) Macro Assembler Version 6.11
Copyright (C) Microsoft Corp 1981-1993. All rights reserved.

Assembling: nuevo.asm
Assembling: que.asm
Assembling: dona.asm

Microsoft (R) Segmented Executable Linker Version 5.60.220 Sep 9 1994
Copyright (C) Microsoft Corp 1984-1993. All rights reserved.

Object Modules [.obj]: nuevo.obj+
Object Modules [.obj]: "que.obj"+
Object Modules [.obj]: "dona.obj"/t
Run File [nuevo.com]: "nuevo.com"
List File [nul.map]: NUL
Libraries [.lib]:
Definitions File [nul.def]:
```

En este ejemplo, después de escribir ML el programa enlazador pide los “Módulos objeto”, que se crean mediante el ensamblador. En este ejemplo tenemos tres módulos objeto: NUEVO, QUE y DONA. Si existe más de un archivo objeto, se escribe primero el nombre del programa principal (NUEVO en este ejemplo) seguido de cualquier otro módulo de soporte.

Los archivos de biblioteca se escriben después del nombre del archivo y del modificador /LINK. En este ejemplo no se usaron archivos de biblioteca. Si desea utilizar una biblioteca llamada NUME.LIB mientras ensambla un programa llamado NUEVO.ASM, debe escribir ML NUEVO.ASM /LINK NUME.LIB.

En el entorno Windows no se puede enlazar un programa; sólo se puede ensamblar. Para enlazar los programas durante la creación de un ejecutable, debe utilizar Visual Studio. Puede ensamblar uno o varios archivos y generar objetos para usarlos con Visual C++. El ejemplo 8-3 muestra cómo se compila un módulo, pero no se enlaza con ML. El modificador /c (en minúscula) indica al ensamblador que debe compilar y generar archivos objeto, /Cx preserva las mayúsculas o minúsculas de todas las funciones y variables y /coff genera una salida en **formato de archivo objeto común** para los archivos objeto que se utilizan en un entorno de 32 bits.

EJEMPLO 8-3

```
C:\Program Files\Microsoft Visual Studio .NET 2003\ Vc7\bin>ml \c \Cx \ coff new.
asm
Microsoft (R) Macro Assembler Version 7.10.3077
Copyright (C) Microsoft Corporation. All rights reserved.

Assembling: nuevo.asm
```

PUBLIC y EXTRN

Estas directivas son muy importantes en la programación modular, ya que permiten las comunicaciones entre módulos. Utilizamos PUBLIC (pública) para declarar las etiquetas de código, datos o segmentos completos que están disponibles para otros módulos de programa. EXTRN (externo) declara que las etiquetas son externas para un módulo. Sin estas instrucciones, los módulos no podrían enlazarse entre sí para crear un programa. Tal vez sí podrían enlazarse, pero un módulo no podría comunicarse con otro.

La directiva PUBLIC se coloca en el campo del código objeto de una instrucción en lenguaje ensamblador para definir una etiqueta como pública, de manera que ésta pueda usarse (ser vista) por otros módulos. La etiqueta que se declare como pública puede ser una dirección de salto, una dirección de datos o un segmento completo. El ejemplo 8-4 muestra la instrucción PUBLIC que se utiliza para definir algunas etiquetas y hacerlas públicas para otros módulos en un fragmento de programa. Cuando los segmentos se hacen públicos, se combinan con otros segmentos públicos que contienen datos con el mismo nombre de segmento.

EJEMPLO 8-4

```
.model flat, c
.data
    public Datos1 ;declara Datos1 y Datos2 como públicas
    public Datos2

    Datos1 db 100 dup (?)
0000 0064[00]
]
0064 0064[00]
]
.code
.startup

    public Lee ;declara Lee como pública

Lee proc far
    mov ah, 6
0006 B4 06
```

La instrucción EXTRN aparece en los segmentos de datos y de código para definir las etiquetas como externas para cada segmento. Si los datos se definen como externos, su tamaño debe definirse como BYTE, WORD o DWORD. Si una dirección de salto o de llamada es externa, debe definirse como NEAR o FAR. El ejemplo 8-5 muestra cómo se utiliza la instrucción EXTRN para indicar que varias etiquetas son externas para el programa listado. Observe en este ejemplo que cualquier dirección o dato externo se define con la letra E en el listado de código ensamblador hexadecimal. Se asume que los ejemplos 8-4 y 8-5 están enlazados entre sí.

EJEMPLO 8-5

```
.model flat, c
.data
    extrn Datos1:byte
    extrn Datos2:byte
    extrn Datos3:word
    extrn Datos4:dword

    .code
    extrn Lee:far
.startup
0005 Bf 0000 E      mov dx, offset Datos1
0008 B9 000A         mov cx, 10
000B                 Inicio:
000B 9A 0000 ---- E call Lee
0010 AA              stosb
0011 E2 F8          loop Inicio
.exit
End
```

Bibliotecas

Los archivos de biblioteca son colecciones de procedimientos comunes que se recolectan en un solo lugar para que puedan utilizarse en muchas aplicaciones distintas. Estos procedimientos se ensamblan y se compilan en un archivo de biblioteca mediante el programa LIB que viene con el programa ensamblador MASM. Tal vez haya notado al configurar Visual C++ para crear los módulos de lenguaje ensamblador del capítulo 7 que había muchos archivos de biblioteca en la lista de enlace que utiliza Visual C++. El archivo de biblioteca (NOMBREARCH.LIB) se invoca cuando se enlaza un programa con el programa enlazador.

¿Por qué tomarse la molestia de trabajar con archivos de biblioteca? Un archivo de este tipo es un excelente lugar para almacenar una colección de procedimientos relacionados. Cuando el archivo de biblioteca se enlaza con un programa, sólo se sacan de este archivo los procedimientos que requiere el programa y se agregan al mismo. Si se requiere ejecutar con eficiencia cierta cantidad de código en lenguaje ensamblador, es esencial tener un buen conjunto de archivos de biblioteca ya que nos ahorran muchas horas de tener que volver a codificar las funciones comunes.

Creación de un archivo de biblioteca. Un archivo de este tipo se crea mediante el comando LIB, el cual ejecuta el programa LIB.EXE que se incluye en Visual Studio. Un archivo de biblioteca es una colección de archivos .OBJ ensamblados que contienen procedimientos o tareas escritas en lenguaje ensamblador o en cualquier otro lenguaje. El ejemplo 8-6 muestra dos funciones separadas (Mayusculas y Minusculas) que se incluyen en un módulo escrito para Windows, el cual se utilizará para estructurar un archivo de biblioteca. Tenga en cuenta que el nombre del procedimiento debe declararse como PUBLIC en un archivo de biblioteca y no es necesario que concuerde con el nombre del archivo, aunque en este ejemplo así es. Como se transfiere una variable a cada archivo, la instrucción EXTRN también aparece en cada procedimiento para obtener acceso a una variable externa. El ejemplo 8-7 muestra los protocolos de C++ que se requieren para utilizar las funciones de este archivo de biblioteca en un programa de C++, siempre y cuando la biblioteca esté enlazada al programa.

EJEMPLO 8-6

```
.586
.model flat, c
.code
    public Mayusculas
    public Minusculas
Mayusculas proc ,\
    Datos1:byte
    mov al, Datos1
    .if al >= 'a' && al <= 'z'
        sub al, 20h
    .endif
    ret
Mayusculas endp

Minusculas proc ,\
    Datos2:byte
    mov al, Datos2
    .if al >= 'A' && al <= 'z'
        add al, 20h
    .endif
    ret
Minusculas endp
End
```

EJEMPLO 8-7

```
extern "C" char Mayusculas(char);
extern "C" char Minusculas(char);
```

El programa LIB empieza con el mensaje de derechos de autor de Microsoft, seguido del indicador *Nombre de biblioteca*. El nombre de biblioteca que se eligió fue letras para el archivo LETRAS.

LIB. Como éste es un nuevo archivo, el programa LIB pedirá el nombre del archivo objeto. Usted debe ensamblar LETRAS.ASM con ML. El comando LIB actual se muestra en el ejemplo 8-8. Observe que el programa LIB se invoca seguido del nombre del objeto en la línea de comandos.

EJEMPLO 8-8

```
C:\Archivos de programa\Microsoft Visual Studio .NET 2003\Vc7\bin>lib LETRAS.obj
Microsoft (R) Library Manager Version 7.10.3077
Copyright (C) Microsoft Corporation. All rights reserved.
```

Hay un programa de utilería llamado DUMPBIN.EXE que se proporciona para mostrar el contenido de la biblioteca o de cualquier otro archivo. El ejemplo 8-9 muestra el resultado de un vaciado binario mediante el uso del modificador /all para mostrar el módulo de biblioteca LETRAS.LIB y todos sus componentes. Cerca de la parte superior del listado están los nombres públicos para _Mayusculas y _Minusculas. La sección RAW DATA #1 (datos puros) contiene las instrucciones actuales en código hexadecimal para los dos procedimientos.

EJEMPLO 8-9

```
C:\Archivos de programa\Microsoft Visual Studio .NET 2003\Vc7\bin>dumpbin /all
letras.lib

Microsoft (R) COFF/PE Dumper Version 7.10.3077
Copyright (C) Microsoft Corporation. All rights reserved.

Dump of file letras.lib

File Type: LIBRARY

Archive member name at 8: /
438FC2D0 time/date Thu Dec 01 21:43:12 2005
    uid
    gid
    0 mode
    24 size
correct header end

2 public symbols

    CC _Mayusculas
    CC _Minusculas

Archive member name at 68: /
438FC2D0 time/date Thu Dec 01 21:43:12 2005
    uid
    gid
    0 mode
    28 size
correct header end

1 offsets

    1      CC

2 public symbols

    1 _Mayusculas
    1 _Minusculas

Archive member name at CC: LETRAS.obj/
438FC111 time/date Thu Dec 01 21:35:45 2005
    uid
    gid
    100666 mode
    272 size
correct header end
```

```

FILE HEADER VALUES
    14C machine (x86)
        3 number of sections
    438FC111 time date stamp Thu Dec 01 21:35:45 2005
        148 file pointer to symbol table
            F number of symbols
            0 size of optional header
            0 characteristics

SECTION HEADER #1
    .text name
        0 physical address
        0 virtual address
        24 size of raw data
        8C file pointer to raw data (0000008C to 000000AF)
        0 file pointer to relocation table
        0 file pointer to line numbers
        0 number of relocations
        0 number of line numbers
    60500020 flags
        Code
        16 byte align
        Execute Read

RAW DATA #1
    00000000: 55 8B EC 8A 45 08 3C 61 72 06 3C 7A 77 02 2C 20 U.i.E.<ar.<zw.,
    00000010: C9 C3 55 8B EC 8A 45 08 3C 41 72 06 3C 5A 77 02 ÉÄU.i.E.<Ar.<Zw.
    00000020: 04 20 C9 C3 .ÉÄ

SECTION HEADER #2
    .data name
        24 physical address
        0 virtual address
        0 size of raw data
        0 file pointer to raw data
        0 file pointer to relocation table
        0 file pointer to line numbers
        0 number of relocations
        0 number of line numbers
    C0500040 flags
        Initialized Data
        16 byte align
        Read Write

SECTION HEADER #3
    .debug$S name
        24 physical address
        0 virtual address
        97 size of raw data
        B0 file pointer to raw data (000000B0 to 00000146)
        0 file pointer to relocation table
        0 file pointer to line numbers
        0 number of relocations
        0 number of line numbers
    42100040 flags
        Initialized Data
        Discardable
        1 byte align
        Read Only

RAW DATA #3
    00000000: 04 00 00 00 F1 00 00 00 00 00 00 00 00 53 00 01 11 ....ñ.....S...
    00000010: 00 00 00 00 43 3A 5C 41 72 63 68 69 76 6F 73 20 ....C:\Archivos
    00000020: 64 65 20 70 72 6F 67 72 61 6D 61 5C 4D 69 63 72 de programa\Micro
    00000030: 6F 73 6F 66 74 20 56 69 73 75 61 6C 20 53 74 75 soft Visual Stu
    00000040: 64 69 6F 20 2E 4E 45 54 20 32 30 30 33 5C 56 63 dio .NET 2003\Vc
    00000050: 37 5C 62 69 6E 5C 4C 45 54 52 41 53 2E 6F 62 6A 7\bin\LETRAS.obj
    00000060: 00 34 00 16 11 03 02 00 00 05 00 00 00 00 00 00 .4.....
    00000070: 00 07 00 0A 00 05 0C 4D 69 63 72 6F 73 6F 66 74 .....Microsoft
    00000080: 20 28 52 29 20 4D 61 63 72 6F 20 41 73 73 65 6D (R) Macro Assem

```

```

00000090: 62 6C 65 72 00 00 00 bler...
COFF SYMBOL TABLE
000 00000000 DEBUG notype      Filename    | .file
                           C:\Archivos de programa\Microsoft Visual Studio .NET 2003\Vc7\bin\LETRAS.ASM

006 000F0C05 ABS     notype      Static      | @comp.id
007 00000000 SECT1  notype      Static      | .text
                           Section length 24, #relocs 0, #linenums 0, checksum 0
009 00000000 SECT2  notype      Static      | .data
                           Section length 0, #relocs 0, #linenums 0, checksum 0
00B 00000000 SECT3  notype      Static      | .debug$S
                           Section length 97, #relocs 0, #linenums 0, checksum 0
00D 00000000 SECT1  notype ()   External    | _Mayusculas
00E 00000012 SECT1  notype ()   External    | _Minusculas

String Table Size = 0x1C bytes

Summary

0 .data
97 .debug$S
24 .text

```

Una vez que el archivo de biblioteca se enlace con su archivo de programa, sólo se colocarán en el archivo ejecutable los procedimientos que se utilicen en su programa. No olvide utilizar la instrucción extern “C” en el programa de C++ para utilizar una función de un archivo de biblioteca.

Macros

Una **macro** es un grupo de instrucciones que realizan una tarea, de la misma forma que un procedimiento realiza una tarea. La diferencia es que un procedimiento se utiliza mediante una instrucción CALL, mientras que una macro (y todas las instrucciones que se definen dentro de ella) se inserta en el programa, en el punto en donde va a utilizarse. La creación de una macro es un proceso muy similar a la creación de un nuevo código de operación, el cual es en realidad una secuencia de instrucciones (en este caso) que pueden utilizarse en el programa. Usted escribe el nombre de la macro y cualquier parámetro asociado con ella, para que después el ensamblador la inserte en el programa. Las macrosecuencias se ejecutan más rápido que los procedimientos, ya que no hay una instrucción CALL o RET que ejecutar. El ensamblador coloca las instrucciones de la macro en su programa, en el punto en el que se invocan. Tenga en cuenta que las macros no funcionarán si utiliza el ensamblador en línea; sólo funcionan en módulos externos de lenguaje ensamblador.

Las directivas MACRO y ENDM se utilizan para delinear una macrosecuencia. La primera instrucción de una macro es la instrucción MACRO, la cual contiene el nombre de la macro y cualquier parámetro asociado con ella. Un ejemplo es MUEVE MACRO A,B que define el nombre de la macro como MUEVE. Este nuevo seudocódigo utiliza dos parámetros: A y B. La última instrucción de una macro es ENDM, la cual se coloca en una línea por sí sola. Nunca debe colocar una etiqueta en frente de la instrucción ENDM, ya que la macro no se ensamblará.

El ejemplo 8-10 muestra cómo se crea una macro y cómo se utiliza en un programa. Las primeras seis líneas de código definen la macro. Esta macro mueve el contenido tipo palabra de la posición de memoria B hacia la posición de memoria A tipo palabra. Después de definir la macro en el ejemplo, se utiliza dos veces. La macro se expande mediante el ensamblador en este ejemplo, para que pueda ver cómo se ensambla para generar los movimientos. Cualquier instrucción en lenguaje máquina hexadecimal que vaya seguida de un número (1 en este ejemplo) es una instrucción de expansión de macro. Las instrucciones de expansión no se escriben en el programa fuente; el ensamblador las genera (si se incluye .LISTALL en el programa) para mostrar que las ha insertado en el programa. Observe que los comentarios en la macro van precedidos por ;; en vez de ; como es costumbre. Las macrosecuencias deben definirse siempre antes de utilizarlas en un programa, por lo que generalmente aparecen en la parte superior del segmento de código.

EJEMPLO 8-10

```

MUEVE    MACRO A,B
          PUSH AX
          MOV  AX,B
          MOV  A,AX
          POP  AX
          ENDM

          MUEVE VAR1, VAR2      ;;mueve VAR2 hacia VAR1

0000 50      1      PUSH  AX
0001 A1 0002 R 1      MOV   AX, VAR2
0004 A3 0000 R 1      MOV   VAR1,AX
0007 58      1      POP   AX

          MOVE  VAR3,VAR4      ;;mueve VAR4 hacia VAR3
0008 50      1      PUSH  AX
0009 A1 0006 R 1      MOV   AX,VAR4
000C A3 0004 R 1      MOV   VAR3,AX
000F 58      1      POP   AX

```

Variables locales en una macro. Algunas veces las macros contienen variables locales. Una **variable local** aparece en la macro, pero no está disponible fuera de ella. Para definir una variable local se utiliza la directiva LOCAL. El ejemplo 8-11 muestra cómo aparece una variable local, que se utiliza como dirección de salto, en una definición de macro. Si esta dirección de salto no se define como local, el ensamblador generará mensajes de error en el segundo intento por utilizar la macro, y en intentos posteriores.

EJEMPLO 8-11

```

LLENA     MACRO DONDE, CUANTOS      ;;llena la memoria
          LOCAL LLENA1
          PUSH SI
          PUSH CX
          MOV  SI, OFFSET DONDE
          MOV  CX,CUANTOS
          MOV  AL,0
          LLENA1: MOV  [SI],AL
          INC  SI
          LOOP LLENA1
          POP  CX
          POP  SI
          ENDM

          LLENA  MENS1,5

          LOCAL LLENA1
          PUSH SI
          PUSH CX
          MOV  SI,OFFSET MENS1
          MOV  CX,5
          MOV  AL,0
          0029 88 04 1 ??0000: MOV  [SI],AL
          002B 46      1 INC  SI
          002C E2 FB    1 LOOP ??0000
          002E 59      1 POP  CX
          002F 5E      1 POP  SI

          LLENA2 MENS2,10

          LOCAL LLENA2
          PUSH SI
          PUSH CX
          MOV  SI,OFFSET MENS2
          MOV  CX,10
          MOV  AL,0
          003A 88 04 1 ??0001: MOV  [SI],AL

```

```

003C 46           1           INC    SI
003D E2 FB        1           LOOP   ??0001
003F 59           1           POP    CX
0040 5E           1           POP    SI
                                .EXIT

```

El ejemplo 8-11 muestra una macro llamada LLENA, la cual almacena cualquier número (parámetro CUANTOS) desde 00H en la posición de memoria direccionada por el parámetro DONDE. Observe cómo se trata la dirección LLENA1 cuando se expanden las macros. El ensamblador utiliza etiquetas que empiezan con ?? para designarlas como etiquetas generadas por el ensamblador.

La directiva LOCAL debe siempre utilizarse en la línea que sigue inmediatamente después de la instrucción MACRO, o de lo contrario se producirá un error. La instrucción LOCAL puede tener hasta 35 etiquetas, todas separadas por comas.

Colocación de las definiciones MACRO en su propio módulo. Las definiciones de macros pueden colocarse en el archivo de programa como se vio anteriormente, o pueden colocarse en su propio módulo de macros. Puede crearse un archivo que contenga sólo macros para incluirlos con otros archivos de programa. La directiva INCLUDE se utiliza para indicar que un archivo de programa incluirá un módulo que contiene definiciones de macros externas. Aunque éste no es un archivo de biblioteca, para todos los fines prácticos funciona como una biblioteca de macrosecuencias.

Cuando se colocan macrosecuencias en un archivo (a menudo con la extensión INC o MAC), no contienen instrucciones PUBLIC como una biblioteca. Si un archivo llamado MACRO.MAC contiene macrosecuencias, se coloca la instrucción INCLUDE en el archivo de programa de la siguiente forma: INCLUDE C:\ASSMMACRO.MAC. En este ejemplo, el archivo de macros se encuentra en la unidad C, subdirectorío ASSM. La instrucción INCLUDE incluye estas macros, justo igual como si usted las hubiera escrito en el archivo. No se necesita una instrucción EXTRN para acceder a las macro instrucciones que se hayan incluido. Los programas pueden contener archivos de inclusión de macros y archivos de bibliotecas.

8-2

USO DEL TECLADO Y LA PANTALLA

Actualmente existen muy pocos programas que no utilizan el teclado y la pantalla. En esta sección se explicará cómo utilizar el teclado y la pantalla que están conectados a la IBM PC o una computadora compatible que utilice Windows.

Lectura del teclado

El teclado de la computadora personal se lee mediante muchos objetos distintos, disponibles en Visual C++. Los datos que se leen del teclado se encuentran en formato de código ASCII o ASCII extendido. Después se almacenan en formato ASCII de 8 bits o en formato Unicode de 16 bits. Como se explicó en un capítulo anterior, Unicode contiene código ASCII en los códigos 0000H-00FFH. El resto de los códigos se utilizan para los conjuntos de caracteres de lenguajes extranjeros. En Visual C++ no se utiliza cin o getch para leer teclas, como en una aplicación de consola del DOS en C++; en vez de ello se utilizan objetos que realizan la misma tarea.

Los datos en código ASCII aparecen como se describe en la tabla 1-8 de la sección 1-4 del capítulo 1. El conjunto extendido de caracteres de la tabla 1-9 se aplica a los datos impresos o mostrados en pantalla solamente, y no a los datos del teclado. Los códigos ASCII de la tabla 1-8 corresponden a la mayoría de las teclas en el teclado. Por medio del teclado también se pueden obtener datos en código ASCII extendido. La tabla 8-1 muestra la mayoría de los códigos ASCII extendidos que se obtienen con varias teclas y combinaciones de ellas. Hay que tener en cuenta que la mayoría de las teclas en el teclado tienen códigos de tecla alternativos. Cada tecla de función tiene cuatro conjuntos de códigos que se seleccionan si se oprime sólo la tecla de función, la combinación Mayús+Tecla de función, la combinación Alt+Tecla de función y la combinación Ctrl+Tecla de función.

TABLA 8-1 La exploración del teclado y los códigos ASCII extendidos que se regresan del teclado.

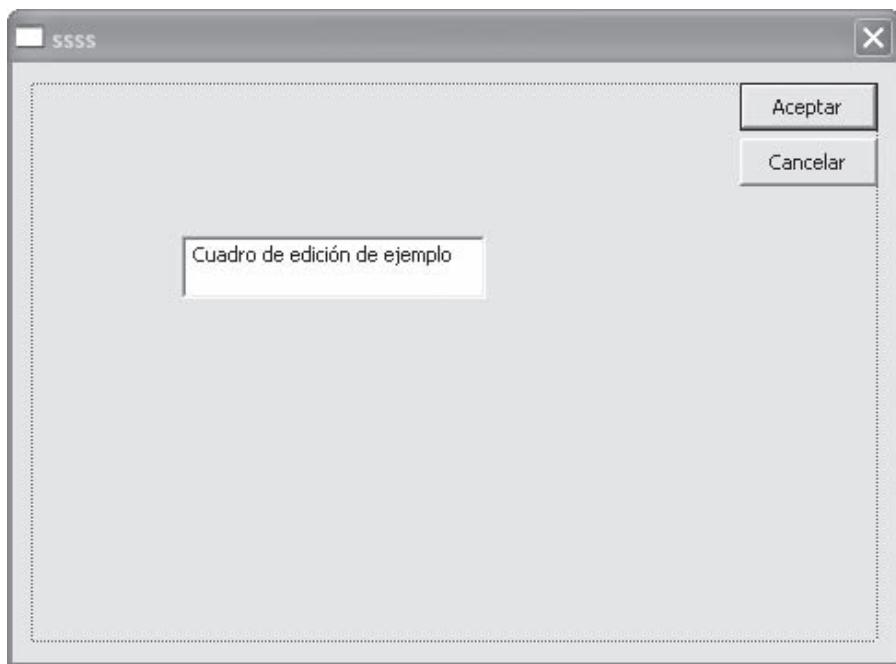
Tecla	Código de exploración	Código ASCII extendido con...			
		Nada	Mayús	Ctrl	Alt
Esc	01				01
1	02				78
2	03			03	79
3	04				7A
4	05				7B
5	06				7C
6	07				7D
7	08				7E
8	09				7F
9	0A				80
0	0B				81
-	0C				82
+	0D				83
Retroceso	0E				0E
Tab	0F	0F	94		A5
Q	10				10
W	11				11
E	12				12
R	13				13
T	14				14
Y	15				15
U	16				16
I	17				17
O	18				18
P	19				19
[1A				1A
]	1B				1B
Intro	1C				1C
Intro	1C				A6
Ctrl Izq	1D				
Ctrl Der	1D				
A	1E				1E
S	1F				1F
D	20				20
F	21				21
G	22				22
H	23				23
J	24				24
K	25				25
L	26				26
;	27				27
'	28				28
'	29				29
Mayús Izq	2A				
\	2B				

(continúa en la siguiente página)

TABLA 8–1 (continuación)

Tecla	Código de exploración	Código ASCII extendido con...			
		Nada	Mayús	Ctrl	Alt
Z	2C				2C
X	2D				2D
C	2E				2E
V	2F				2F
B	30				30
N	31				31
M	32				32
,	33				33
.	34				34
/	35				35
/ gris	35			95	A4
Mayús Der	36				
Impr pant	E0 2A E0 37				
Alt Izq	38				
Alt Der	38				
Espacio	39				
Bloq Mayús	3A				
F1	3B	3B	54	5E	68
F2	3C	3C	55	5F	69
F3	3D	3D	56	60	6A
F4	3E	3E	57	61	6B
F5	3F	3F	58	62	6C
F6	40	40	59	63	6D
F7	41	41	5A	64	6E
F8	42	42	5B	65	6F
F9	43	43	5C	66	70
F10	44	44	5D	67	71
F11	57	85	87	89	8B
F12	58	86	88	8A	8C
Núm	45				
Bloq Despl	46				
Inicio	E0 47	47	47	77	97
Arriba	48	48	48	8D	98
Re Pág	E0 49	49	49	84	99
- gris	4A				
Izquierda	4B	4B	4B	73	9B
Centro	4C				
Derecha	4D	4D	4D	74	9D
+ gris	4E				
Fin	E0 4F	4F	4F	75	9F
Abajo	E0 50	50	50	91	A0
Av Pág	E0 51	51	51	76	A1
Ins	E0 52	52	52	92	A2
Supr	E0 53	53	53	93	A3
Pausa	E0 10 45				

Figura 8-1 Uso del teclado:
ejemplo de una aplicación
basada en cuadro de diálogo.



Para comprender mejor cómo se lee una tecla en Windows, vamos a crear una aplicación en Visual C++ que contiene un cuadro de edición simple (objeto de edición). La figura 8-1 muestra esa aplicación escrita como aplicación basada en cuadro de diálogo. Vamos a recordar cómo se crea una aplicación MFC (Microsoft Foundation Classes) basada en cuadro de diálogo:

1. Inicie Visual Studio.
2. Haga clic en Archivo | Nuevo | Proyecto para abrir un nuevo proyecto.
3. En Proyectos de Visual C++ seleccione Aplicación MFC y escriba un nombre; después haga clic en Aceptar.
4. En Tipo de aplicación, seleccione la opción Basada en cuadros de diálogo y después haga clic en Finalizar.

Una vez creada la nueva aplicación basada en cuadro de diálogo, seleccione el control de edición del cuadro de herramientas y dibújelo en la pantalla del cuadro de diálogo, como se muestra en la figura 8-1. A continuación, elimine el mensaje TODO:. Una vez que el control de edición esté en la pantalla, haga clic en él con el botón derecho del ratón y seleccione la opción Agregar variable. Escriba el nombre Edicion1 para la variable del control.

Establecimiento del enfoque. Lo primero que debemos hacer para la aplicación es establecer el enfoque en el control de edición. Cuando se establece el enfoque, el cursor se desplaza al objeto (en este caso, el control de edición). Para ello se utiliza la instrucción Edicion1.SetFocus(), ya que en nuestro caso el control de edición se llama Edicion1. Esta instrucción se coloca en la función OnInitDialog de la clase del cuadro de diálogo. Vaya a la función OnInitDialog y coloque la instrucción anterior en donde aparece el comentario TODO:. Cambie también el retorno en la función OnInitDialog para que devuelva falso en vez de verdadero. Esto se requiere para establecer el enfoque. Ahora la aplicación establecerá el enfoque en el control de edición. Esto significa que el cursor intermitente aparecerá dentro del control de edición.

Al ejecutar la aplicación y escribir en el control de edición, el programa lee el teclado y muestra cada carácter a medida que se va escribiendo. En algunos casos esto puede ser indeseable y se puede requerir cierto filtrado. Uno de esos casos es cuando el programa requiere que el usuario escriba sólo datos hexadecimales. Para poder interceptar las pulsaciones de tecla a medida que se va escribiendo, se utiliza la función **PreTranslateMessage**. Esta función intercepta todos los mensajes de Windows *antes* de que actúe en base a ellos.

Para insertar esta función en la aplicación de cuadro de diálogo haga lo siguiente: Abra la ventana Vista de clases (menú Ver | Vista de clases) y haga clic en el signo positivo al lado del nombre del proyecto para expandir el árbol de clases. Después haga clic con el botón derecho del ratón en la clase del cuadro de diálogo que acaba de crear (en nuestro caso se llama *CsssDlg*) y seleccione la opción Agregar | Agregar función en el menú contextual. A continuación aparecerá la ventana del Asistente para agregar funciones miembro. En Tipo de valor devuelto, escriba **BOOL**. En Nombre de la función, escriba **PreTranslateMessage**. En Tipo de parámetro, escriba **MSG*** y en Nombre de parámetro escriba **pMsg**. Haga clic en el botón Agregar para agregar el parámetro a la firma de la función. Por último, haga clic en la opción Virtual. Ahora haga clic en Finalizar para agregar la declaración y la definición de la función en la aplicación. Lo único que queda ahora es escribir el código del ejemplo 8-12 dentro de la definición de la función.

Para ilustrar el filtrado, esta aplicación utiliza la función **PreTranslateMessage** para analizar el carácter del teclado que se escribe antes de que el programa vea las pulsaciones de tecla, para que puedan modificarse. Aquí el programa sólo permite escribir los números del 0 al 9 y las letras de la “A” a la “F”. Si se escribe una letra minúscula, se convierte en mayúscula. Para lograr esto, utilice el mensaje de Windows **WM_CHAR** como se muestra en el ejemplo 8-12. En este ejemplo, se utiliza C++ para filtrar la entrada desde el teclado en el control de edición.

EJEMPLO 8-12

```
BOOL CsssDlg::PreTranslateMessage (MSG* pMsg)
{
    if ( pMsg->message == WM_CHAR )
    {
        unsigned int tecla = pMsg->wParam;           //obtiene la tecla
        if (tecla >= 'a' && tecla <= 'f')           //convierte en mayúscula
            tecla -= 32;
        if ( (tecla < '0' || tecla > 'F') || (tecla > '9' && tecla < 'A') )
            return true;                            //ignora éstos
        pMsg->wParam = tecla;                      //sustituye la tecla
    }
    return CDialog::PreTranslateMessage (pMsg);
}
```

La estructura de mensajes (pMsg) asociada con la función **PreTranslateMessage** contiene información sobre el mensaje de Windows. Todos los mensajes de Windows son nombres asignados que empiezan con **WM_**. En este caso, el mensaje interpretado por la función es **WM_CHAR**. Este mensaje contiene el carácter ASCII escrito en el teclado en la variable wParam de la estructura pMsg. La instrucción if más externa analiza el mensaje **WM_CHAR**, el cual sólo se produce cuando se escribe una tecla. La siguiente instrucción if evalúa wParam para ver si contiene las letras minúsculas a, b, c, d, e y f. Si se escribe una de estas teclas, se resta 32 de wParam para convertirla en mayúscula. La diferencia entre las letras mayúsculas y minúsculas es un 32 decimal, o 20H. Esto es, la A es 41H y la a es 61H, una diferencia de 20H.

Si no se escribe una letra entre la a y la f, el siguiente paso es comprobar los números del 0 al 9; si no se escribió un número, se ejecuta la instrucción **return TRUE**. Si la función **PreTranslateMessage** devuelve verdadero, la pulsación de tecla se borra o se ignora. Un valor de retorno falso hace que Windows emita un sonido de advertencia y se desecha la pulsación de tecla. En este caso, si se escribe un número o una letra entre a y f, o entre A y F, la pulsación de tecla se pasa al marco de trabajo de Windows mediante el retorno normal al final de la función **PreTranslateMessage**. Esto filtra las pulsaciones de tecla de manera que sólo aparezcan las letras A-F o los números 0-9 en el cuadro de edición.

En el ejemplo 8-13 se repite el ejemplo 8-12, esta vez mediante el uso del ensamblador en línea para realizar la misma tarea. En este ejemplo, parece que se requiere escribir menos código en C++ que en lenguaje ensamblador, pero es importante poder visualizar ambas formas.

EJEMPLO 8-13

```
BOOL CsssDlg::PreTranslateMessage (MSG* pMsg)
{
    if ( pMsg->message == WM_CHAR )
```

```

{
    unsigned int tecla = pMsg->wParam;
    _asm
    {
        mov     eax, tecla
        cmp     eax, 'a'
        jb      PreTransl
        cmp     eax, 'f'
        ja      PreTransl
        sub     eax, 32           ;si a - f
        mov     tecla, eax

PreTransl:
        cmp     eax, '0'
        jb      PreTrans2
        cmp     eax, '9'
        jbe     PreTrans3
        cmp     eax, 'A'
        jb      PreTrans2
        cmp     eax, 'F'
        jbe     PreTrans3

PreTrans2:
}
return true;
PreTrans3:
    pMsg->wParam = tecla;
}
return CDlg::PreTranslateMessage(pMsg);
}

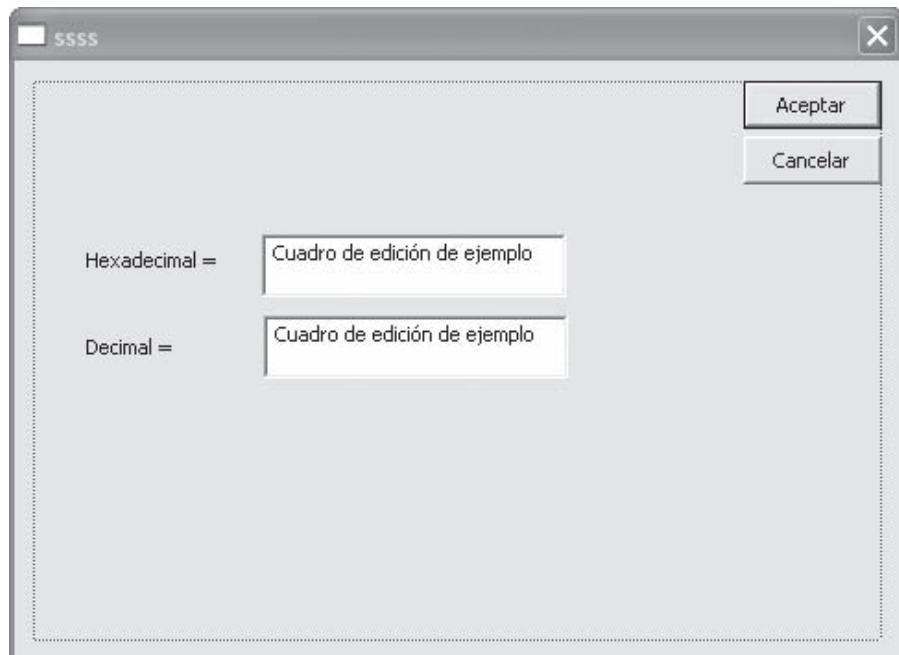
```

Si se agrega este código a la aplicación y se ejecuta, las únicas teclas que aparecerán en el control de edición serán 0-9 y A-F. Puede llevarse a cabo cualquier nivel de filtrado en forma similar en el entorno de Visual C++. Las propiedades del control de edición incluyen la propiedad Uppercase, que podría haberse activado para reducir la tarea de filtrado, pero aquí se logró mediante software.

Uso de la pantalla

Al igual que el teclado, en Visual C++ los objetos se utilizan para mostrar información. El control de edición puede utilizarse ya sea para leer datos o para mostrarlos, como puede hacerse en la mayoría

FIGURA 8-2 Conversión de hexadecimal a decimal.



de los objetos. Modifique la aplicación que se presenta en la figura 8-1 de manera que contenga un control de edición adicional, como se muestra en la figura 8-2. Observe que se han agregado unas cuantas etiquetas estáticas para identificar el contenido de los controles de edición. En esta nueva aplicación los datos del teclado se leen todavía en el control Edicion1, pero cuando se oprime Intro aparece en Edicion2 (el segundo control de edición) una versión decimal de los datos introducidos en Edicion1. Asegúrese de que el segundo control se llame Edicion2 para mantener la compatibilidad con el software que presentamos aquí.

Para hacer que el programa reaccione a la tecla Intro, representada por el código ASCII 13 (0DH o 0x0d), modifique la función PreTranslateMessage del ejemplo 8-12 como se muestra en el ejemplo 8-14 (vea la sección en cursiva). Observe cómo se detecta la tecla Intro. Una vez que se detecta, la variable Edicion1 puede convertirse a decimal para mostrarla en Edicion2.

EJEMPLO 8-14

```
BOOL CssssDlg::PreTranslateMessage (MSG* pMsg)
{
    if ( pMsg->message == WM_KEYDOWN && pMsg->wParam == 13 ) // es Intro?
    {
        // el software para convertir y mostrar datos decimales va aquí
    }
    else if ( pMsg->message == WM_CHAR )
    {
        unsigned int tecla = pMsg->wParam;
        if (tecla >= 'a' && tecla <= 'f')
            tecla -= 32;
        if ( tecla < '0' || tecla > 'F' || tecla > '9' && tecla < 'A' )
            return true;
        pMsg->wParam = tecla;
    }
    return CDialog::PreTranslateMessage (pMsg);
}
```

El problema principal es que los datos que se introducen en un control de edición pueden utilizarse como una cadena o como un número decimal, pero no como un número hexadecimal. Para obtener la cadena de un control de edición se utiliza la función miembro GetDlgItemText. En el programa del ejemplo 8-15, esta función aparece como GetDlgItemText(IDC_EDIT1, temp), en donde temp es una variable tipo CString que recibe la cadena de caracteres del control de edición. Una vez que está disponible, la cadena puede convertirse a binario para que a continuación se convierta a decimal y pueda mostrarse en Edicion2.

EJEMPLO 8-15

```
BOOL CssssDlg::PreTranslateMessage (MSG* pMsg)
{
    if ( pMsg->message == WM_KEYDOWN && pMsg->wParam == 13 ) // es Intro?
    {
        CString temp;
        int temp1 = 0;
        char temp2;
        GetDlgItemText(IDC_EDIT1, temp);
        for ( int a = 0; a < temp.GetLength(); a++)
        {
            temp2 = temp.GetAt(a);
            __asm
            {
                mov al,temp2 ;convierte el dígito ASCII en binario
                sub al,30h
                cmp al,9
                jbe L1
                sub al,7
            L1:
                shl temp1,4
                or byte ptr temp1,al
            }
    }
```

```

        }
        SetDlgItemInt(IDC_EDIT2,temp1);
        return true;
    }
    else if ( pMsg->message == WM_CHAR )
    {
        unsigned int tecla = pMsg->wParam;
        if (tecla >= 'a' && tecla <= 'f')
            tecla -= 32;
        if ( tecla < '0' || tecla > 'F' || tecla > '9' && tecla < 'A' )
            return true;
        pMsg->wParam = tecla;
    }
    return CDialog::PreTranslateMessage(pMsg);
}

```

El ejemplo 8-15 muestra la aplicación completa. Cuando se oprime Intro, el programa utiliza GetDlgItemText(IDC_EDIT1, temp) para obtener la cadena de caracteres de Edicion1 y la almacena en la variable CString llamada temp. La siguiente porción del programa (el ciclo for) explora la cadena de caracteres un carácter a la vez, convirtiendo cada carácter a binario a medida que ensambla la versión binaria completa de la cadena en la variable tipo entero llamada temp1. Cada dígito se obtiene de la cadena de caracteres mediante el uso de la función miembro CString llamada GetAt(int posición). Así, el carácter ASCII que se obtiene se almacena en la variable tipo char llamada temp2 y se utiliza lenguaje ensamblador para realizar la conversión.

Al convertir de ASCII a binario se restan 30H. Esto convierte los números ASCII (0-9) del 30H al 39H en los números binarios del 0 al 9. No convierte los números ASCII del 41H al 46H (A-F) en binario, ya que el resultado que se obtiene es del 11H al 16H y no del 0AH al 0FH. Para ajustar los valores que se obtienen para las letras se utiliza una instrucción para detectar los números del 11H al 16H y después se resta un 7 adicional para convertir los números del 11H al 16H en números del 0AH al 0FH. Una vez que el dígito ASCII se encuentra en forma binaria, el entero almacenado en temp1 se desplaza cuatro posiciones binarias a la izquierda y se aplica un OR entre este entero y la versión binaria del dígito ASCII para acumular los dígitos convertidos en temp1.

Después de que el número hexadecimal de Edicion1 se convierte en binario y se almacena en la variable temp1, se muestra en Edicion2 mediante el uso de la función SetDlgItemInt. Como se mencionó antes, un valor de retorno verdadero informa a la interfaz de Windows que se ha oprimido la tecla Intro. En el ejemplo 8-16 podrá ver la versión de este programa en la que se utiliza sólo lenguaje C++, en su formato de lenguaje ensamblador mediante la opción Depurar de Visual Studio, para compararlo con el código ensamblador que se utiliza en el ejemplo 8-15. Sólo se muestra el ciclo for para compararlo con la versión en lenguaje ensamblador del ejemplo 8-15.

EJEMPLO 8-16

```

// el ciclo for antes de desensamblarlo

for ( int a = 0; a < temp.GetLength(); a++)
{
    temp1 <= 4;
    if ( temp.GetAt(a) > '9' )
        temp1 |= temp.GetAt(a) - 0x37;
    else
        temp1 |= temp.GetAt(a) - 0x30;
}

// el ciclo for después de desensamblarlo

for ( int a = 0; a < temp.GetLength(); a++)
00419756 mov     dword ptr [a],0
0041975D jmp     CssssDlg::PreTranslateMessage+98h (419768h)
0041975F mov     eax,dword ptr [a]
00419762 add     eax,1
00419765 mov     dword ptr [a],eax
00419768 mov     esi,esp

```

```

0041976A lea      ecx,[temp]
0041976D call    dword ptr [__imp_ATL::CSimpleStringT<char,1>::GetLength
(42ABD8h)]
00419773 cmp      esi,esp
00419775 call    @ILT+2405(__RTC_CheckEsp) (41196Ah)
0041977A cmp      dword ptr [a],eax
0041977D jge     CssssDlg::PreTranslateMessage+121h (4197F1h)
{
    temp1 <= 4;
    eax,dword ptr [temp1]
00419782 shl      eax,4
00419785 mov      dword ptr [temp1],eax
    if ( temp.GetAt(a) > '9' )
00419788 mov      esi,esp
0041978A mov      eax,dword ptr [a]
0041978D push    eax
0041978E lea      ecx,[temp]
00419791 call    dword ptr [__imp_ATL::CSimpleStringT<char,1>::GetAt
(42ABDCh)]
00419797 cmp      esi,esp
00419799 call    @ILT+2405(__RTC_CheckEsp) (41196Ah)
0041979E movsx   ecx,al
004197A1 cmp      ecx,39h
004197A4 jle     CssssDlg::PreTranslateMessage+0FAh (4197CAh)
                  temp1 |= temp.GetAt(a) - 0x37;
004197A6 mov      esi,esp
004197A8 mov      eax,dword ptr [a]
004197AB push    eax
004197AC lea      ecx,[temp]
004197AF call    dword ptr [__imp_ATL::CSimpleStringT<char,1>::GetAt
(42ABDCh)]
004197B5 cmp      esi,esp
004197B7 call    @ILT+2405(__RTC_CheckEsp) (41196Ah)
004197BC movsx   ecx,al
004197BF sub      ecx,37h
004197C2 or       ecx,dword ptr [temp1]
004197C5 mov      dword ptr [temp1],ecx
    else
004197C8 jmp     CssssDlg::PreTranslateMessage+11Ch (4197EC)
                  temp1 |= temp.GetAt(a) - 0x30;
004197CA mov      esi,esp
004197CC mov      eax,dword ptr [a]
004197CF push    eax
004197D0 lea      ecx,[temp]
004197D3 call    dword ptr [__imp_ATL::CSimpleStringT<char,1>::GetAt
(42ABDCh)]
004197D9 cmp      esi,esp
004197DB call    @ILT+2405(__RTC_CheckEsp) (41196Ah)
004197E0 movsx   ecx,al
004197E3 sub      ecx,30h
004197E6 or       ecx,dword ptr [temp1]
004197E9 mov      dword ptr [temp1],ecx
}
004197EC jmp     CssssDlg::PreTranslateMessage+8Fh (41975Fh)

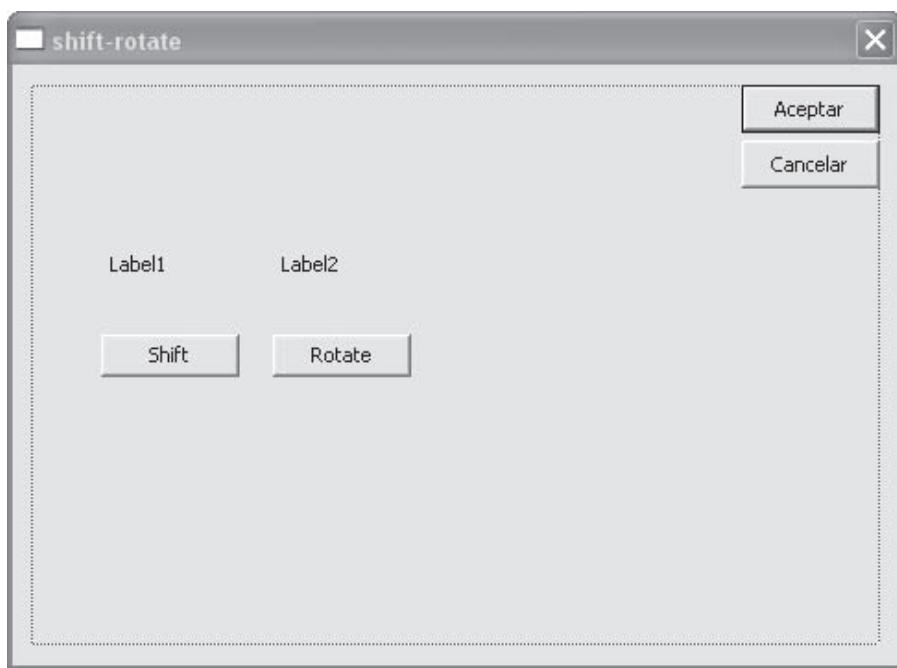
```

Uso de controles Active X en un programa

Los controles Active X son objetos que se utilizan en cualquier lenguaje de programación visual. Pueden escribirse en C++, BASIC o en cualquier lenguaje. Para seleccionar y utilizar controles Active X en un programa, haga clic con el botón derecho en el cuadro de diálogo y seleccione la opción Insertar control Active X. Tenga en cuenta que los controles Active X son propiedad de diversas compañías y que para usarlos en un programa tal vez tenga que pagar regalías al propietario del control. Mientras no venda un producto, podrá utilizar estos controles sin necesidad de pagar una cuota. Los controles Active X tienen por lo general muchas más características que los controles prefabricados que se incluyen en Visual Studio.

En la figura 8-3 se muestra una aplicación que utiliza un control Active X. Aunque ésta es una aplicación bastante sencilla, demuestra el uso de un control Active X para mostrar una etiqueta dinámica.

FIGURA 8–3 Una aplicación que muestra el uso de controles Active X y controles prefabricados.



Sólo hay una etiqueta estática (que no puede modificarse) disponible como control prefabricado en Visual C++ y algunas veces se necesita una etiqueta dinámica (que puede modificarse). Las funciones miembro que se utilizan más a menudo en un control Active X (suponiendo que utiliza Visual Studio .NET) son `put_` para mutar una característica del control, o `get_` para acceder a una característica del control. La figura 8-3 utiliza dos controles de comando prefabricados (con las etiquetas Shift y Rotate) y dos controles de Active X llamados Microsoft Forms 2.0 Label. Estos controles en realidad son parte de Microsoft Word, por lo que también debe tener Microsoft Word instalado para poder utilizar estos controles Active X.

Una vez que su cuadro de diálogo aparezca como en la figura 8-3, agregue nombres a los dos controles Active X Forms 2.0 Label: Etiqueta1 y Etiqueta2. Ahora agregue manejadores para las funciones de clic de los dos botones de comando (BN_CLICKED). (Recuerde que para hacer esto debe ir a Propiedades y hacer clic en el Botón, después debe hacer clic en el símbolo de rayo para acceder a los manejadores miembro para los controles.) Ahora haga clic en el ícono que está justo a la derecha del símbolo de rayo y vaya hasta el mensaje WM_TIMER. Agregue el manejador para el temporizador, que utilizaremos en este programa de ejemplo. Vamos a agregar dos variables mediante el uso de la vista de clases: haga clic en el menú Ver | Vista de clases para seleccionar la vista de clases. Ahora haga clic con el botón derecho del ratón en la clase que termina con Dlg (ésta es la clase del cuadro de diálogo en la que aparecen los manejadores que acabamos de instalar.) Seleccione la opción Agregar | Agregar variable y agregue una variable tipo Booleana llamada Shift, y una variable tipo char llamada datos1. Éstas deben ser variables públicas de la clase, y deben ser visibles desde cualquier parte de la clase Dlg.

El ejemplo 8-17 ilustra el software que se agrega a la aplicación en los tres manejadores requeridos para implementar la aplicación para la figura 8-3. El software para los manejadores del evento de clic de los dos botones de comando es corto y fácil de entender. La primera instrucción en cada uno de los manejadores coloca una leyenda en Etiqueta1. Si se oprime el botón de desplazamiento (shift) se muestra la leyenda “Despl. Izq. =”, y si se oprime el botón de desplazamiento cíclico (rotate) se muestra la leyenda “Despl. Cíclico Izq. =” en Etiqueta1. La segunda instrucción tan sólo asigna el valor de verdadero o falso a la variable Booleana Shift, dependiendo del botón oprimido. En ambos manejadores de los botones de comando, datos1 empieza en 1 y se muestra el valor 00000001 en Etiqueta2. Por último, se inicia un temporizador para el botón de control mediante el uso de la función SetTimer. La primera variable es un número elegido para el temporizador, el segundo número es el tiempo de ciclo del temporizador en milisegundos y el tercero es un cero para casi todas las aplicaciones de Visual C++. El temporizador se llama cada 500 milisegundos a la función OnTimer en el ejemplo. Por lo tanto, la

función OnTimer se ejecuta una vez cada medio segundo hasta que se cierra el programa. Esto permite que ocurra un evento cada medio segundo. En este ejemplo, cada medio segundo se aplica un shift (desplazamiento) o un rotate (desplazamiento cíclico) al número 00000001 y se muestra en pantalla.

EJEMPLO 8-17

```
void CshiftrotateDlg::OnBnClickedButton1()
{
    Etiqueta1.put_Caption("Despl. Izq = ");
    Shift = true;           //selecciona el desplazamiento (shift)
    datos1 = 1;
    Etiqueta2.put_Caption("00000001");
    SetTimer(1,500,0);
}

void CshiftrotateDlg::OnBnClickedButton2()
{
    Etiqueta1.put_Caption("Despl. Cílico Izq. = ");
    Shift = false;          //selecciona el desplazamiento cíclico (rotate)
    datos1 = 1;
    Etiqueta2.put_Caption("00000001");
    SetTimer(1,500,0);
}

void CshiftrotateDlg::OnTimer(UINT nIDEvent)
{
    if (nIDEvent == 1)
    {
        CString temp = "";
        char temp1 = datos1;
        if (Shift)
            _asm shl temp1,1;      //desplazamiento (shift)
        else
            _asm rol temp1,1;      //desplazamiento cíclico (rotate)
        datos1 = temp1;
        for (int a = 128; a > 0; a>>=1)
        {
            if ( ( temp1 & a ) == a )
                temp += "1";
            else
                temp += "0";
        }
        Etiqueta2.put_Caption(temp);
    }
    CDialog::OnTimer(nIDEvent);
}
```

La función OnTimer primero revisa para ver si el temporizador 1 produjo el evento OnTimer. Esto se requiere en sistemas que utilizan más de un temporizador. Aquí sería opcional, ya que sólo existe un temporizador. En este ejemplo se utilizan dos variables locales: una cadena de caracteres (temp) y una variable tipo char llamada temp1. Si Shift es verdadera, temp1 se desplaza (shift) y si Shift es falsa, temp1 se desplaza en forma cíclica (rotate). Recuerde que los botones de comando ajustan la variable Shift ya sea en verdadero o en falso para seleccionar el tipo de operación. Aquí se utiliza una variable local para los datos que se van a desplazar. La razón es que no se puede acceder con facilidad a la variable de una clase desde lenguaje ensamblador, por lo que antes de usar datos1, ésta variable se hace local por medio de temp1 y, de igual forma, se vuelve a hacer local una vez que temp1 se almacena de nuevo en datos1.

La porción más extraña de la función OnTimer es el desarrollo de la cadena de caracteres numéricos para mostrarla en pantalla. La instrucción for inicializa la variable tipo entera a con el valor 128 (10000000) y continúa mientras que a sea mayor que cero. La expresión de un ciclo for es por lo general a++ o a--, pero en este caso es a>>1. Esta instrucción desplaza el 10000000 una posición a la derecha cada vez que se ejecuta el ciclo, de manera que cambia a 01000000, después a 00100000 y así, sucesivamente. El cuerpo de la instrucción for evalúa temp1 para ver si hay un 1 en cada posición de bit a medida que la variable a cambia de valor; de esta forma, va reconstruyendo el número binario en la

cadena de caracteres temp. Por último, la cadena temp se muestra en pantalla como una leyenda para el control Etiqueta2, cuando Terminal la función OnTimer.

¿Se puede realizar la instrucción de desplazamiento cíclico (rotate) en lenguaje C++? Sí, pero no con una sola instrucción. El ejemplo 8-18 muestra la instrucción if (desplazamiento) en C++. Aquí podemos observar que es más complicada la instrucción de desplazamiento cíclico en C++, en base al código ensamblador que genera esta porción de lenguaje C++. Tal vez se pregunte por qué el código ensamblador generado por C++ parece ser muy inefficiente. El código está escrito para funcionar de manera correcta en todas las situaciones, por lo que la mayoría de las veces es mucho más elaborado de lo requerido para muchas aplicaciones. Debido a esto, el programa se puede hacer más eficiente si lo optimizamos para adaptarlo a nuestra situación específica.

EJEMPLO 8-18

```

        if ( shift )
00419E2C  mov      eax,dword ptr [this]
00419E2F  movzx   ecx,byte ptr [eax+270h]
00419E36  test    ecx,ecx
00419E38  je      CshiftrotateDlg::OnTimer+84h (419E44h)
            temp1 <= 1;
00419E3A  mov     al,byte ptr [temp1]
00419E3D  shl     al,1
00419E3F  mov     byte ptr [temp1],al
            else
00419E42  jmp     CshiftrotateDlg::OnTimer+0A4h (419E64h)
            if ( ( temp1 & 0x80 ) == 0x80 )
00419E44  movsx   eax,byte ptr [temp1]
00419E48  and    eax,80h
00419E4D  je      CshiftrotateDlg::OnTimer+9Ch (419E5Ch)
            temp1 = ( temp1 << 1 ) + 1;
00419E4F  movsx   eax,byte ptr [temp1]
00419E53  lea     ecx,[eax+eax+1]
00419E57  mov     byte ptr [temp1],cl
            else
00419E5A  jmp     CshiftrotateDlg::OnTimer+0A4h (419E64h)
            temp1 <= 1;
00419E5C  mov     al,byte ptr [temp1]
00419E5F  shl     al,1
00419E61  mov     byte ptr [temp1],al

```

El ratón

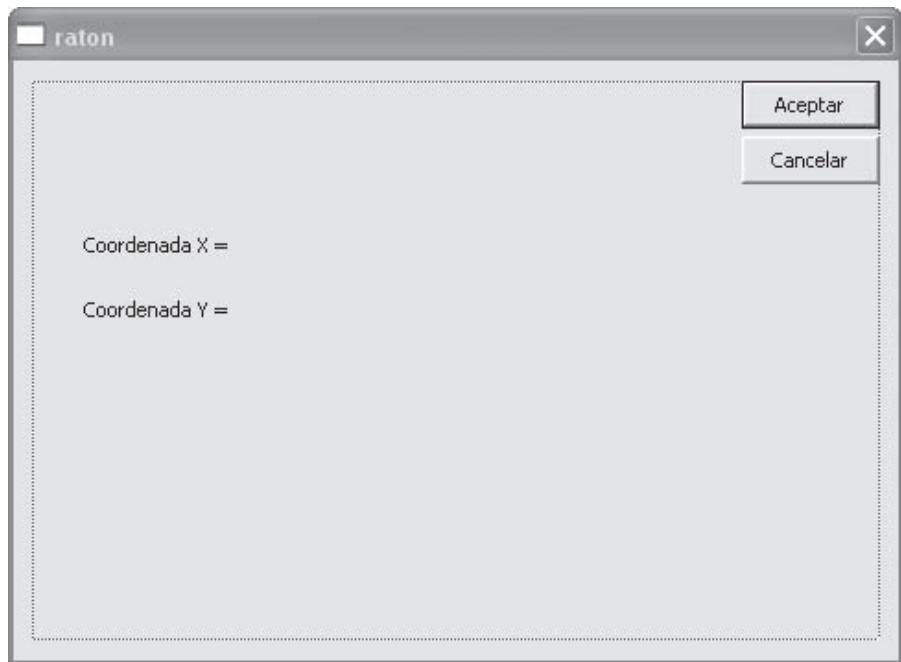
Dentro del marco de trabajo de Visual C++ podemos acceder al dispositivo señalador conocido como ratón y también a un dispositivo conocido como “trackball”, similar al ratón. Al igual que muchos otros dispositivos controlados por Windows, en una aplicación se pueden instalar manejadores de mensajes

TABLA 8-2 Mensajes del ratón.

Mensaje	Manejador	Parámetro 1	Parámetro 2	Parámetro 3
WM_MOUSEMOVE	OnMouseMove	UINT nFlags	CPoint point	
WM_MOUSEWHEEL	OnMouseWheel	UINT nFlags	short zDelta	CPoint punto
WM_LBUTTONDOWN	OnButtonDown	UINT nFlags	CPoint point	
WM_RBUTTONDOWN	OnRButtonDown	UINT nFlags	CPoint point	
WM_LBUTTONUP	OnButtonUp	UINT nFlags	CPoint point	
WM_RBUTTONUP	OnRButtonUp	UINT nFlags	CPoint point	
WM_LBUTTONDOWNDBLCLK	OnButtonDownDbClick	UINT nFlags	CPoint point	
WM_RBUTTONDOWNDBLCLK	OnRButtonDownDbClick	UINT nFlags	CPoint point	

Notas: zDelta = La distancia que se gira la rueda. nFlags son: MK_CONTROL = tecla Ctrl oprimida, MK_LBUTTON = botón izquierdo oprimido, MK_MBUTTON = botón medio oprimido, MK_RBUTTON = botón derecho oprimido y MK_SHIFT = tecla Mayús oprimida. Para la combinación de tecla Mayús con botón medio, utilice MK_SHIFT | MK_MBUTTON.

FIGURA 8-4 Aplicación que muestra las coordenadas del ratón en pantalla.



para el ratón y pueden utilizarse también otras funciones. Como vimos en ejemplos anteriores, los manejadores de mensajes (manejadores de eventos) se instalan en una aplicación mediante el uso de la sección Propiedades, haciendo clic en el ícono a la derecha del símbolo de rayo. En la tabla 8-2 se muestran los manejadores de mensajes para el ratón.

En la figura 8-4 se muestra una aplicación de ejemplo sobre el uso del ratón. Este ejemplo muestra las coordenadas del punto en el que se encuentra el ratón, a medida que se mueve el puntero dentro de la aplicación de cuadro de diálogo. Aunque el listado de software (vea el ejemplo 8-19) no utiliza lenguaje ensamblador, sí muestra cómo obtener y mostrar la posición del puntero del ratón.

EJEMPLO 8-19

```
void CratonDlg::OnMouseMove(UINT nFlags, CPoint point)
{
    char temp[10];
    CString coordenada;
    itoa(point.x, temp, 10);
    coordenada = "Coordenada X = ";
    Etiqueta1.put_Caption(coordenada + temp); //muestra el punto x
    itoa(point.y, temp, 10);
    coordenada = "Coordenada Y = ";
    Etiqueta2.put_Caption(coordenada + temp); //muestra el punto y
    CDialog::OnMouseMove(nFlags, point);
}
```

El ejemplo 8-19 muestra la única parte de la aplicación que se modifica para mostrar las coordenadas del ratón. La función OnMouseMove se instala cuando se instala el manejador WM_MOUSEMOVE en el programa. Esta aplicación utiliza dos controles Active X tipo Microsoft Forms 2.0 Label para mostrar las coordenadas del ratón. Estos dos objetos se llaman Etiqueta1 y Etiqueta2. La función OnMouseMove devuelve la posición del puntero del ratón en la estructura de datos CPoint, como los miembros x e y. Este ejemplo utiliza la función itoa (de entero a ASCII) para convertir el entero que se devuelve como punto x o y del ratón en una cadena de caracteres ASCII. Con esta función se asocian tres parámetros. El primer parámetro es el entero que se va a convertir; el segundo parámetro es la cadena de caracteres que va a recibir los datos en código ASCII; y el tercer parámetro es la base numérica de la conversión (en este caso, decimal). En este ejemplo no se utiliza el parámetro nFlags.

Instale un manejador para el botón izquierdo del ratón (WM_LBUTTONDOWN) y uno para el botón derecho (WM_RBUTTONDOWN). Modifique su aplicación como se muestra en el ejemplo 8-20. En este ejemplo, cuando se oprime el botón izquierdo cambia el color de las etiquetas a rojo y cuando se oprime el botón derecho cambia a azul. Los códigos de color que se utilizan con la mayoría de las funciones en Visual C++ son números de 24 bits que representan 16 millones de colores y están en el formato BBGGR, en donde BB es un número hexadecimal de dos dígitos que representa la saturación de la señal de video azul, GG es el verde y RR es el rojo. Esto significa que el número de color 0xff0000 es 255 (el máximo) para el video azul, 0 (ninguno) para el verde y 0 (ninguno) para el rojo, lo cual hace que aparezca el color azul más brillante en la pantalla. Todos los colores se forman en base a los tres colores primarios de la luz: rojo, verde y azul. En la entrada “Color Referente” del índice de los archivos de ayuda de Visual Studio podrá encontrar una paleta de colores con sus códigos respectivos. Observe que los números de colores que se presentan con los mosaicos de colores están en formato RGB, que es el formato inverso requerido para C++.

EJEMPLO 8-20

```
void CratonDlg::OnRButtonDown(UINT nFlags, CPoint point)
{
    Etiqueta1.put_ForeColor(0x0000ff);
    Etiqueta2.put_ForeColor(0x0000ff);
    CDialog::OnRButtonDown(nFlags, point);
}

void CratonDlg::OnLButtonDown(UINT nFlags, CPoint point)
{
    Etiqueta1.put_ForeColor(0xffff00);
    Etiqueta2.put_ForeColor(0xffff00);
    CDialog::OnLButtonDown(nFlags, point);
}
```

8-3

CONVERSIONES DE DATOS

En los sistemas computacionales, los datos raras veces se encuentran en el formato correcto. Una de las tareas principales del sistema es convertir datos de un formato a otro. En esta sección se describirán las conversiones entre datos binarios y ASCII. Los datos binarios se sacan de un registro o memoria y se convierten en ASCII para mostrarlos en pantalla. En muchos casos, los datos ASCII se convierten en binario a medida que se escriben en el teclado. También se explicará la conversión entre datos ASCII y hexadecimales.

Conversión de binario a ASCII

Existen tres formas para convertir datos de binario a ASCII: (1) mediante la instrucción AAM si el número es menor que 100, (2) mediante una serie de divisiones decimales (división entre 10), o (3) mediante el uso de la función itoa de C++. En esta sección se presentarán las técnicas 1 y 2.

La instrucción AAM convierte el valor en AX en un número BCD desempacado de dos dígitos, y lo pone en AX. Si el número en AX es 0062H (98 decimal) antes de que se ejecute AAM, AX tendrá el valor 0908H después de que se ejecute AAM. Éste no es código ASCII, pero puede convertirse si se suma 3030H a AX. El ejemplo 8-21 muestra un programa que utiliza el procedimiento que procesa el valor binario en AL (0-99) y lo muestra en la pantalla como un número decimal. El procedimiento borra el cero a la izquierda con el código de espacio ASCII cuando el número está entre 0 y 9. Este programa de ejemplo muestra el número 74 (datosprueba) en la pantalla. Para implementarlo, cree una aplicación basada en cuadro de diálogo en Visual C++ y coloque una sola etiqueta llamada Etiqueta1 (no olvide agregar esta variable) en la pantalla del cuadro de diálogo. El número 74 aparecerá si el código en el ejemplo 8-21 se coloca en la sección TODO: de la función OnInitDialog en la clase Dialog (la clase que termina con las letras Dlg).

EJEMPLO 8-21

```
// TODO: agregar aquí inicialización adicional

char datosprueba = 74;
char temp[3];
__asm
{
    mov al,datosprueba           ;obtiene los datos de prueba
    mov ah,0                      ;borra AH
    aam                          ;convierte en BCD
    add ah,20h                    ;evalúa si hay un cero a la
    cmp al,20h                   ;izquierda
    je D1                         ;si hay un cero a la izquierda
    add ah,10h                   ;convierte en ASCII
D1:
    mov temp[0],ah               ;almacena el carácter
    add al,30h                   ;convierte en ASCII
    mov temp[1],al
    mov temp[2],0                 ;termina cadena con valor nulo
}
Etiqueta1.put_Caption(temp);      //muestra el número de prueba
```

La razón por la que AAM convierte cualquier número entre 0 y 99 en un número BCD desempa-
cado de dos dígitos es porque divide AX entre 10. El resultado queda en AX, por lo que AH contiene el
cociente y AL el residuo. El mismo esquema de dividir entre 10 puede expandirse para convertir cual-
quier número entero de cualquier sistema numérico (si se cambia el número entre el que se va a dividir)
de binario a una cadena de caracteres en código ASCII que puede mostrarse en la pantalla. Por ejemplo,
si AX se divide entre 8 en vez de dividirse entre 10, el número que se muestra es octal.

El algoritmo (llamado **algoritmo de Horner**) para convertir de binario a código ASCII decimal
es:

1. Se divide entre 10 y después se almacena el residuo en la pila como un dígito BCD significativo.
2. Se repite el paso 1 hasta que el cociente sea 0.
3. Se recupera cada residuo y se suma 30H para convertirlo a ASCII antes de mostrarlo o imprimirllo.

El ejemplo 8-22 muestra cómo se convierte el contenido de EAX de 32 bits sin signo en ASCII
y se muestra en pantalla. Aquí se divide EAX entre 10 y se almacena el residuo en la pila después de
cada división, para su posterior conversión en ASCII. Una vez que se han convertido todos los dígitos,
el resultado se muestra en la pantalla de video mediante el proceso de quitar los residuos de la pila y
convertirlos en código ASCII. Este programa también borra los ceros a la izquierda. Como se mencionó
antes, puede utilizarse cualquier base numérica si se cambia la variable baseNum en este ejemplo. De
nuevo, para implementar este ejemplo cree una aplicación de cuadro de diálogo con una sola etiqueta
Active X llamada Etiqueta1. Coloque el software del ejemplo en la función OnInitDialog.

Si la base numérica es mayor que 10, se utilizan letras para representar caracteres más allá del 9.
Para modificar el software de manera que funcione con bases numéricas mayores a 10, coloque la ins-
trucción cmp dl,39h después de la instrucción add dl,30h y luego la instrucción jbe L2A seguida de add
dl,7. Después agregue el prefijo L2A: a la instrucción mov temp[ebx], dl.

EJEMPLO 8-22

```
int datosprueba = 2349;
int baseNum = 10;
char temp[20];
__asm
{
    mov ebx,0                  ;inicializa el apuntador
    push baseNum              ;mete la base a la pila
    mov eax,datosprueba       ;obtiene los datos de prueba
```

```

L1:
    mov  edx,0          ;borra el contenido de edx
    div  baseNum        ;divide entre la base
    push edx            ;almacena el residuo
    cmp  eax,0
    jnz  L1             ;se repite hasta que llegue a 0

L2:
    pop  edx            ;obtiene el residuo
    cmp  edx,baseNum   ;si ya terminó
    je   L3             ;convierte en ASCII
    add  dl,30h         ;almacena el dígito
    mov  temp[ebx],dl   ;apunta al siguiente
    inc  ebx             ;se repite hasta que termine
    jmp  L2             ;almacena el valor nulo en la cadena

L3:
    mov  temp[ebx],0
}
Etiqueta1.put_Caption(temp);

```

Conversión de ASCII a binario

Por lo general, las conversiones de ASCII a binario empiezan con la entrada desde el teclado. Si se escribe una sola tecla, la conversión ocurre cuando se resta el valor 30H del número. Si se escribe más de una tecla, aún hay que restar 30H pero surge un paso adicional. Después de restar 30H, el número se suma al resultado una vez que el resultado anterior se multiplica primero por 10.

El algoritmo para convertir de ASCII a binario es:

1. Se empieza con un resultado binario de 0.
2. Se restan 30H del carácter para convertirlo en BCD.
3. Se multiplica el resultado por 10 y luego se suma el nuevo dígito BCD.
4. Se repiten los pasos 2 y 3 para cada carácter del número.

El ejemplo 8-23 muestra un programa que implementa este algoritmo. Aquí, el número binario se muestra de la variable temp en un control de edición, mediante el uso de la función SetDlgItemInt. Cada vez que se ejecuta este programa, lee un número de la variable tipo char llamada numero y lo convierte en binario para mostrarlo en el control de edición llamado Edicion1.

EJEMPLO 8-23

```

char numero[] = "2356";
int temp = 0;
__asm
{
    mov  ebx,0          ;inicializa el apuntador
    mov  ecx,ebx
B1:
    mov  cl,numero[ebx]  ;obtiene el dígito
    inc  ebx            ;direcciona el siguiente dígito
    cmp  cl,0           ;si se encuentra el valor nulo
    je   B2             ;convierte de ASCII a BCD
    sub  cl,30h
    mov  eax,10
    mul  temp
    add  eax,ecx        ;suma el dígito
    mov  temp,eax        ;almacena el resultado
    jmp  B1
B2:
}
SetDlgItemInt(IDC_EDIT1, temp);

```

Cómo mostrar y leer datos hexadecimales

Es más fácil leer datos hexadecimales que decimales desde el teclado y mostrarlos en pantalla. Estos tipos de datos no se utilizan a nivel de aplicación, sino a nivel de sistema. Con frecuencia los datos a nivel de sistema son hexadecimales y deben mostrarse ya sea en formato hexadecimal, o leerse desde el teclado como datos hexadecimales.

Cómo leer datos hexadecimales. Los datos hexadecimales se representan con números del 0 al 9 y letras de la “A” a la “F”. Los códigos ASCII que se obtienen del teclado para los datos hexadecimales son del 30H al 39H para los números del 0 al 9, y del 41H al 46H (A-F) o del 61H al 66H (a-f) para las letras. Para que un programa que lea datos hexadecimales sea útil, debe poder aceptar tanto letras mayúsculas como minúsculas, así como números.

El ejemplo 8-24 muestra dos funciones: una (Conv) convierte el contenido de una variable tipo char sin signo de código ASCII a un solo dígito hexadecimal, y la otra (Leeh) convierte una variable tipo CString con hasta ocho dígitos hexadecimales en un valor numérico que se devuelve como entero sin signo de 32 bits. Este ejemplo muestra una mezcla balanceada de C++ y lenguaje ensamblador para realizar la conversión.

EJEMPLO 8-24

```
//la función Leeah convierte un valor CString de hexadecimal a un
//entero sin signo.

unsigned int CasciiDlg::Leeh(Cstring temp)
{
    unsigned int numero = 0;
    for ( int a = 0; a < temp.GetLength(); a++ )
    {
        numero <= 4;
        numero += Conv(temp.GetAt(a));
    }
    return numero;
}

//la función Conv convierte un dígito en binario

unsigned char CasciiDlg::Conv(unsigned char temp)
{
    __asm
    {
        cmp temp,'9'
        jbe Conv2          ;si es 0 - 9
        cmp temp,'a'
        jb Conv1          ;si es A - F
        sub temp,20h       ;a mayúsculas
Conv1:
        sub temp,7
Conv2:
        sub temp,30h
    }
    return temp;
}
```

Cómo mostrar datos hexadecimales. Para mostrar datos hexadecimales hay que dividir un número en secciones de dos, cuatro u ocho bits, las cuales se deben convertir en dígitos hexadecimales. La conversión se realiza mediante el proceso de sumar 30H a los números del 0 al 9, o 37H a las letras de la “A” a la “F” para cada sección.

En el ejemplo 8-25 se muestra el listado de una función (Muestrah) que convierte un entero sin signo en una cadena de caracteres de dos, cuatro u ocho dígitos, según lo determine el tamaño del parámetro. Esta función devuelve una variable tipo CString, la cual representa el contenido del parámetro

entero sin signo que recibe. Muestrah(número, 2) convierte un número entero sin signo en una cadena CString hexadecimal de dos dígitos; Muestrah(número, 4) convierte el número en una cadena hexadecimal de cuatro dígitos y Muestrah(número, 8) convierte el número en una cadena de caracteres hexadecimal de ocho dígitos.

EJEMPLO 8-25

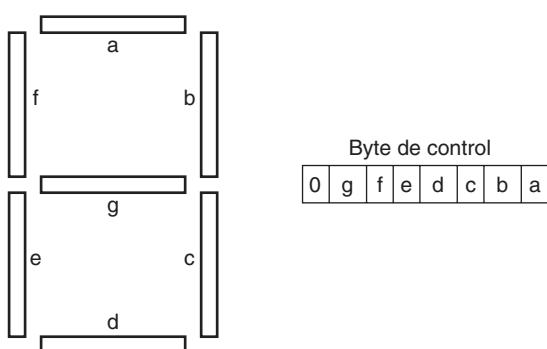
```
CString CasciiDlg::Muestrah(unsigned int numero, unsigned int tamanio)
{
    CString temp;
    numero <= ( 8 - tamanio ) * 4;           //ajusta la posición
    for ( int a = 0; a < tamanio; a++ )
    {
        char templ;
        _asm
        {
            rol numero,4;
            mov al,byte ptr numero
            and al,0fh          ;crea 0-f
            add al,30h           ;convierte en ASCII
            cmp al,39h
            jbe Muestrahl1
            add al,7
        Muestrahl1:
            mov templ,al
        }
        temp += templ;           //agrega el dígito a la cadena
    }
    return temp;
}
```

Uso de tablas de búsqueda para conversiones de datos

Las tablas de búsqueda se utilizan con frecuencia para convertir datos de un formato a otro. Una tabla de búsqueda se forma en la memoria como una lista de datos a los que hace referencia un procedimiento para realizar conversiones. En muchas tablas de búsqueda se utiliza a menudo la **instrucción XLAT** para buscar datos en la tabla, siempre y cuando ésta contenga datos de ocho bits y que su longitud sea menor o igual a 256 bytes.

Conversión de BCD a código de siete segmentos. Una aplicación simple que utiliza una tabla de búsqueda es la conversión de BCD a código de siete segmentos. El ejemplo 8-26 muestra una tabla de búsqueda que contiene los códigos de siete segmentos para los números del 0 al 9. Estos códigos se utilizan con la pantalla de siete segmentos que se muestra en la figura 8-5. Esta pantalla de siete segmentos utiliza entradas activas en alta (1 lógico) para alumbrar un segmento. El código de la tabla de búsqueda (el arreglo temp1) se ordena de manera que el segmento a esté en la posición de bit 0 y que el segmento g esté en la posición de bit 6. La posición de bit 7 es 0 en este ejemplo, pero puede utilizarse para mostrar un punto decimal, en caso de ser requerido.

FIGURA 8-5 La pantalla de siete segmentos.



EJEMPLO 8-26

```
CString CasciiDlg::Busca(unsigned char temp)
{
    char temp1[] = {0x3f, 6, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 7, 0x7f, 0x6f};
    __asm
    {
        lea ebx,temp1
        mov al,temp
        xlat
        mov temp,al
    }
    return temp;
}
```

La función Busca, que realiza la conversión, contiene sólo unas cuantas instrucciones y asume que el parámetro temp contiene el dígito BCD (0-9) a convertir en código de siete segmentos, el cual se devuelve como un valor tipo char sin signo. La primera instrucción direcciona la tabla de búsqueda al cargar su dirección en EBX; las otras instrucciones realizan la conversión y devuelven el código de siete segmentos como un valor tipo char sin signo. Aquí, el arreglo temp1 es indizado por el BCD que recibe la función en temp.

Uso de una tabla de búsqueda para acceder a datos ASCII. Algunas técnicas de programación requieren la conversión de códigos numéricicos a cadenas de caracteres ASCII. Por ejemplo, suponga que necesita mostrar los días de la semana para un programa de calendario. Como el número de caracteres ASCII en cada día es distinto, debe usarse algún tipo de tabla de búsqueda para hacer referencia a los días de la semana en código ASCII.

El programa del ejemplo 8-27 muestra una tabla en forma de arreglo, la cual hace referencia a unas cadenas de caracteres en código ASCII. Cada cadena de caracteres contiene un día de la semana en código ASCII. La tabla contiene referencias para cada día de la semana. La función que accede al día de la semana utiliza el parámetro dia, con los números del 0 al 6 para referirse a los días del Domingo hasta el Sábado. Si dia contiene un 2 cuando se invoca esta función, se muestra la palabra “Martes” en la pantalla. Tenga en cuenta que esta función no utiliza código ensamblador, ya que sólo estamos accediendo a un elemento en un arreglo mediante el uso del día de la semana como índice. Se muestra de forma que puedan presentarse usos adicionales para los arreglos, ya que pueden tener aplicación en programas que se utilicen con microprocesadores integrados.

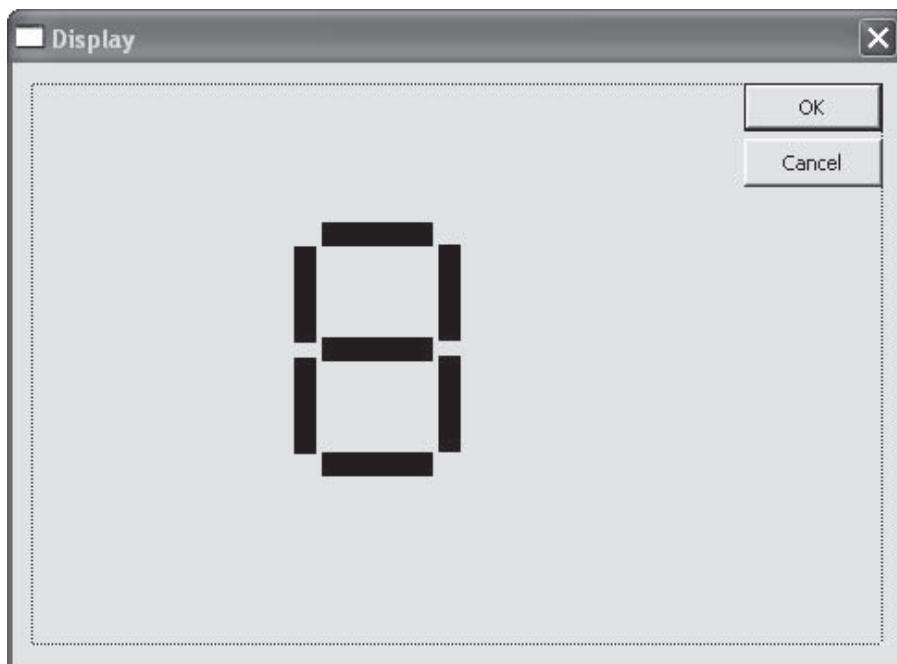
EJEMPLO 8-27

```
CString CasciiDlg::ObtieneDia(unsigned char dia)
{
    CString temp[] =
    {
        "Domingo",
        "Lunes",
        "Martes",
        "Miércoles",
        "Jueves",
        "Viernes",
        "Sábado",
    };
    return temp[dia];
}
```

Un programa de ejemplo que utiliza una tabla de búsqueda

La figura 8-6 muestra la pantalla de una aplicación de cuadro de diálogo llamada Pantalla, la cual muestra un carácter estilo siete segmentos en la pantalla para cada tecla numérica que se escribe desde el teclado. Como se vio en ejemplos anteriores, el teclado puede interceptarse en un programa de Visual C++ mediante la función PreTranslateMessage; esto es lo que hace este programa para obtener la tecla

FIGURA 8–6 Ejemplo que muestra un dígito estilo siete segmentos.



desde el teclado. Después, el código escrito se filtra de manera que sólo se acepten números del 0 al 9 y luego se utiliza una tabla de búsqueda para acceder al código de siete segmentos que se va a mostrar en pantalla.

El dígito a mostrar se dibuja mediante los objetos Active X Microsoft Forms 2.0 Label. Las barras horizontales se muestran usando medidas de 40×8 y las barras verticales se dibujan usando medidas de 8×32 . Las medidas de un objeto aparecen en la esquina del extremo inferior derecho de la pantalla de recursos en Visual Studio. Asegúrese de agregar las etiquetas en el mismo orden que el de la pantalla; esto es, primero agregue la etiqueta a, después la etiqueta b y así sucesivamente, de igual forma que la pantalla de siete segmentos de la figura 8-5. Utilice los nombres de variables Etiqueta1 hasta Etiqueta7 y no olvide agregarlas como controles en la aplicación.

Agregue a su programa la función Borra, que se muestra en el ejemplo 8-28. Esta función se utiliza para borrar el dígito de la pantalla la primera vez que se ejecuta el programa y también antes de mostrar un nuevo dígito. Observe que el color de fondo de cada etiqueta se cambia a 0x8000000f. Éste es el color del fondo de un cuadro de diálogo en Windows, aún y cuando se identifica como el sombreado en la cara de los botones de comando. La tabla 8-3 muestra todas las constantes de color del sistema para Windows. Asegúrese de colocar una instrucción Borra() en el área TODO: del procedimiento OnInitDialog en la clase del cuadro de diálogo. Esto hace que la pantalla se borre cuando el programa se ejecuta en este punto.

EJEMPLO 8-28

```
void CPantallaDlg::Borra(void)
{
    Etiqueta1.put_BackColor(0x8000000f);
    Etiqueta2.put_BackColor(0x8000000f);
    Etiqueta3.put_BackColor(0x8000000f);
    Etiqueta4.put_BackColor(0x8000000f);
    Etiqueta5.put_BackColor(0x8000000f);
    Etiqueta6.put_BackColor(0x8000000f);
    Etiqueta7.put_BackColor(0x8000000f);
}
```

Al escribir una tecla, la función PreTranslateMessage (vea el ejemplo 8-29) filtra la pulsación de tecla y la convierte en código de siete segmentos mediante el uso de la tabla de búsqueda. Después de la

TABLA 8-3

Constantes de color del sistema Windows.

<i>Valor</i>	<i>Descripción</i>
0x80000000	Color de la barra de desplazamiento.
0x80000001	Color del escritorio.
0x80000002	Color de la barra de título para la ventana activa.
0x80000003	Color de la barra de título para la ventana inactiva.
0x80000004	Color de fondo del menú.
0x80000005	Color de fondo de la ventana.
0x80000006	Color del marco de la ventana.
0x80000007	Color del texto en los menús.
0x80000008	Color del texto en las ventanas.
0x80000009	Color del texto en leyenda, cuadro de tamaño y flecha de desplazamiento.
0x8000000A	Color del borde de la ventana activa.
0x8000000B	Color del borde de la ventana inactiva.
0x8000000C	Color de fondo de las aplicaciones con interfaz de múltiples documentos (MDI).
0x8000000D	Color de fondo de los elementos seleccionados en un control.
0x8000000E	Color del texto de los elementos seleccionados en un color.
0x8000000F	Color de sombreado en la cara de los botones de comando.
0x80000010	Color de sombreado en el borde de los botones de comando.
0x80000011	Texto en gris (deshabilitado).
0x80000012	Color del texto en los botones de opción.
0x80000013	Color del texto en una leyenda inactiva.
0x80000014	Color de resalte para elementos de pantalla en 3D.
0x80000015	Color de sombra más oscuro para elementos de pantalla en 3D.
0x80000016	Segundo color en 3D más claro.
0x80000017	Color del texto en la Información sobre las herramientas (Tooltips).
0x80000018	Color de fondo en la Información sobre las herramientas (Tooltips).

conversión a código de siete segmentos se hace una llamada a la función MuestraDigito para mostrar el dígito en la pantalla. Esta función evalúa cada bit del código de siete segmentos y cambia el color de fondo del segmento apropiado a negro.

EJEMPLO 8-29

```
BOOL CPantallaDlg::PreTranslateMessage(MSG* pMsg)
{
    char busca[] = {0x3f, 6, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 7, 0x7f, 0x6f};
    char temp;
    if ( pMsg->message == WM_KEYDOWN )
    {
        if ( pMsg->wParam >= '0' && pMsg->wParam <= '9' )
        {
            temp = pMsg->wParam - 0x30;
            _asm                                //busca el código de 7
                                                //segmentos
            {
                lea ebx,busca
                mov al,temp
                xlat
                mov temp,al
            }
            MuestraDigito(temp);           //muestra el dígito
        }
        return true;                      //termina con la pulsación de
                                            //tecla
    }
    return CDialog::PreTranslateMessage(pMsg);
}

void CPantallaDlg::MuestraDigito(char codigo)
```

```

{
    Borra();
    if ( ( codigo & 1 ) == 1 )
        Etiqueta11.put_BackColor(0);           //evalúa el segmento a
    if ( ( codigo & 2 ) == 2 )
        Etiqueta12.put_BackColor(0);           //evalúa el segmento b
    if ( ( codigo & 4 ) == 4 )
        Etiqueta13.put_BackColor(0);           //evalúa el segmento c
    if ( ( codigo & 8 ) == 8 )
        Etiqueta14.put_BackColor(0);           //evalúa el segmento d
    if ( ( codigo & 16 ) == 16 )
        Etiqueta15.put_BackColor(0);           //evalúa el segmento e
    if ( ( codigo & 32 ) == 32 )
        Etiqueta16.put_BackColor(0);           //evalúa el segmento f
    if ( ( codigo & 64 ) == 64 )
        Etiqueta17.put_BackColor(0);           //evalúa el segmento g
}

```

8-4**ARCHIVOS EN DISCO**

En el disco los datos se almacenan en forma de archivos. El disco en sí está organizado en cuatro partes principales: el sector de arranque, la **tabla de asignación de archivos** (FAT), el directorio raíz y las áreas de almacenamiento de datos. El sistema Windows NTFS (**Sistema de archivos de nueva tecnología**) contiene un sector de arranque y una **tabla de archivos maestra** (MFT). El primer sector en el disco es el sector de arranque, el cual se utiliza para cargar el sistema operativo en disco (DOS) del disco a la memoria cuando se aplica energía a la computadora.

La FAT (o MFT) es donde el sistema operativo almacena los nombres de los archivos/subdirectorios y sus posiciones en el disco. Todas las referencias a cualquier archivo en disco se manejan a través de la FAT (o MFT). Se hace referencia a todos los demás subdirectorios y archivos a través del **directorio raíz** en el sistema FAT. El sistema NTFS no tiene un directorio raíz, aún y cuando el sistema de archivos parezca tenerlo. Los archivos en disco se consideran todavía como archivos de acceso secuencial, lo que significa que se accede a ellos un bit a la vez, desde el principio del archivo hacia el final. Tanto el sistema de archivos NTFS como el FAT están en uso. En la mayoría de los sistemas Windows modernos se utiliza NTFS; en el disco flexible, el CD-ROM y el DVD se utiliza el sistema FAT.

Organización del disco

La figura 8-7 muestra la organización de los sectores y pistas en la superficie del disco. Esta organización se aplica tanto a los sistemas de memoria de disco flexible como a los de disco duro. La pista exterior es siempre la pista 0, y la pista interior es 39 (doble densidad) o 79 (alta densidad) en los discos flexibles. La pista interna en un disco duro se determina en base al tamaño del disco, y podría ser 10,000 o mayor para los discos duros muy grandes.

La figura 8-8 muestra la organización de los datos en un disco. La longitud de la FAT se determina en base al tamaño del disco. En el sistema NTFS, la longitud de la MFT se determina en base al número de archivos que se almacenan en el disco. De igual forma, la longitud del directorio raíz en un volumen FAT se determina en base al número de archivos y subdirectorios que se encuentran dentro de él. El sector de arranque siempre es un sector individual de 512 bytes, que se encuentra en la pista exterior en el sector 0, el primer sector.

El sector de arranque contiene un programa con la secuencia inicial de instrucciones de arranque (**bootstrap loader**), el cual se lee en la RAM cuando el sistema se enciende. Este programa se ejecuta y carga el sistema operativo en la RAM. A continuación, el programa con la secuencia inicial de instrucciones de arranque pasa el control al programa del sistema operativo, lo que permite a la computadora estar bajo el control de Windows y ejecutar este sistema, en la mayoría de los casos. También ocurre la misma secuencia de eventos si se encuentra el sistema operativo Linux en el disco.

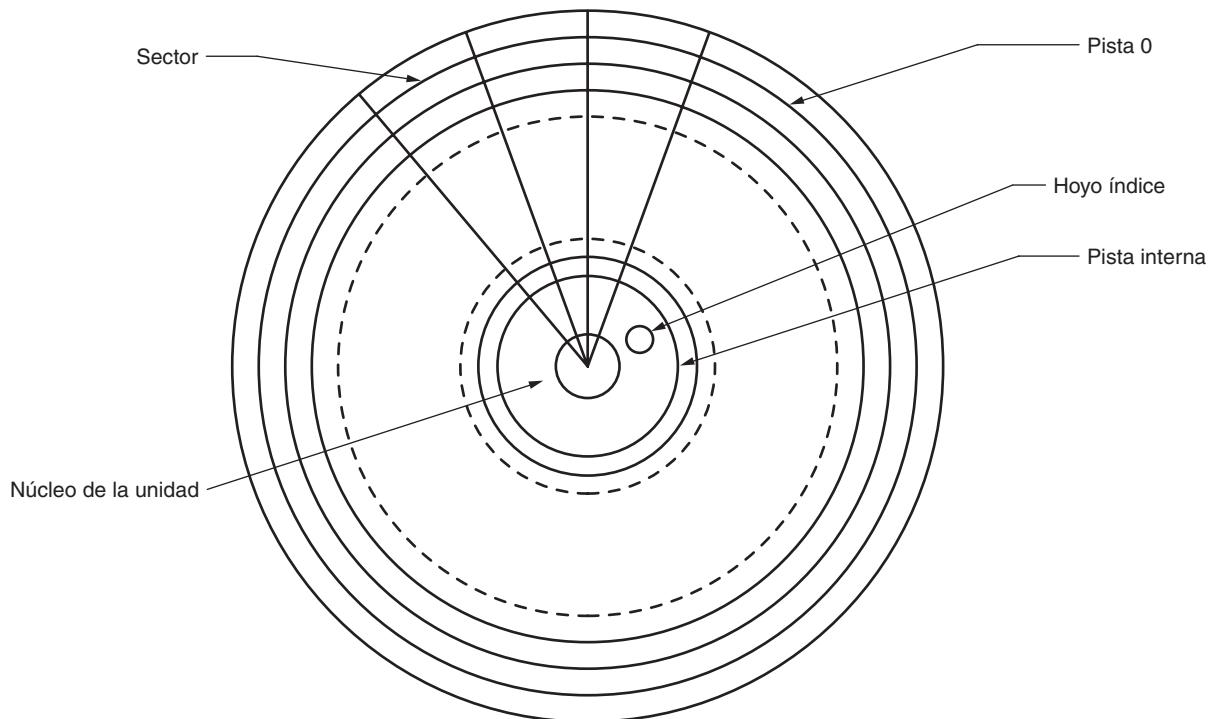
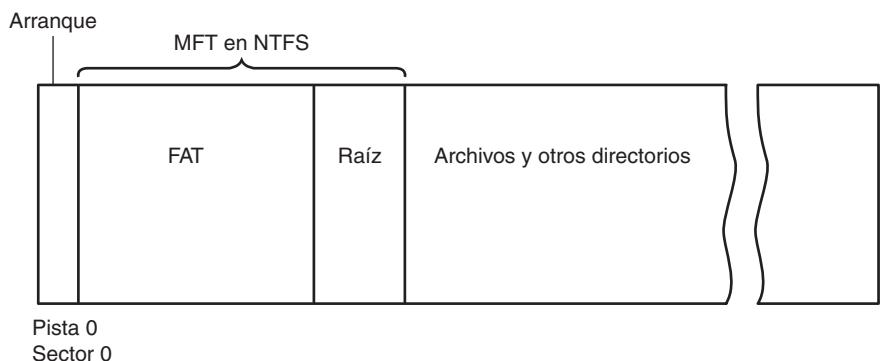


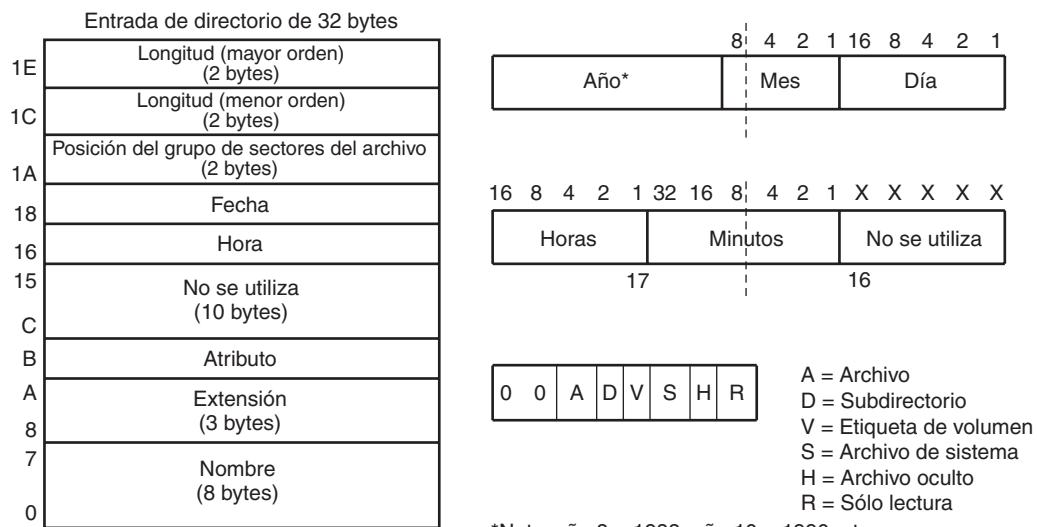
FIGURA 8-7 Estructura del disco.

La FAT indica qué sectores están libres, cuáles están corruptos (inutilizables) y cuáles contienen datos. Se hace referencia a la tabla FAT cada vez que el sistema operativo escribe datos en el disco, de manera que pueda encontrar un sector libre. Cada clúster libre se indica mediante 0000H en la FAT y cada sector ocupado se indica mediante el número de grupo. Un **clúster** puede tener desde un solo sector hasta cualquier número de sectores de longitud. Muchos sistemas de memoria de disco duro utilizan cuatro sectores por clúster, lo que significa que el archivo más pequeño es de 512 bytes × 4, o de 2048 bytes de longitud. En un sistema que utilice NTFS, el tamaño del clúster es por lo general de 4 bytes, que equivale a cuatro sectores de longitud.

La figura 8-9 muestra el formato de cada entrada de directorio en el directorio raíz, o en cualquier otro directorio o subdirectorios. Cada entrada contiene el nombre, la extensión, el atributo, la hora, la fecha, la posición y la longitud. La longitud del archivo se almacena como un número de 32 bits. Esto

FIGURA 8-8 Las principales áreas de almacenamiento de datos en un disco.





*Nota: año 8 = 1988, año 10 = 1990, etc.

FIGURA 8-9 Formato de cualquier entrada de directorio o subdirectorío en el sistema FAT.

significa que un archivo puede tener una longitud máxima de 4 Gbytes. La posición es el número del clúster inicial.

El sistema Windows NTFS utiliza una entrada de directorio o registro mucho mayor (1024 bytes) que la del sistema FAT (32 bytes). El registro de la MFT contiene el nombre del archivo, la fecha, el atributo y los datos, que pueden ser todo el contenido completo del archivo o un apuntador hacia la posición en la que están almacenados los datos en el disco, a lo cual se le conoce como un recorrido de archivo. Por lo general, los archivos menores de 1500 bytes caben en el registro de la MFT. Los archivos más grandes cabe en un recorrido o recorridos de archivos. Un **recorrido de archivo** es una serie de clústeres contiguos que almacenan los datos del archivo. La figura 8-10 muestra un registro de la MFT en el sistema de archivos Windows NTFS. El atributo de información contiene la fecha de creación, la fecha de última modificación, la hora de creación, la hora de última modificación y los atributos del archivo: de sólo lectura, archivo o directorio, etc. El atributo de seguridad almacena toda la información de seguridad para el archivo, para limitar su acceso en el sistema Windows. El encabezado almacena información sobre el tipo de registro, su tamaño, su nombre (opcional) y si es residente o no.

Nombres de archivos

Los archivos y los programas se almacenan en disco y se hace referencia a ellos mediante un nombre de archivo y una extensión. En el sistema operativo DOS, el *nombre de archivo* sólo puede tener de uno a ocho caracteres de longitud. El nombre de archivo puede contener casi cualquier carácter ASCII, excepto los espacios o los caracteres “\./[]*:;<>I;?=”. Además de su nombre, el archivo puede tener una *extensión* opcional de uno a tres dígitos. El nombre de un archivo y su extensión siempre van separados por un punto. Si se utiliza uno de los sistemas desde Windows 95 hasta Windows XP, el nombre de archivo puede tener cualquier longitud (hasta 255 caracteres) y puede incluso contener espacios. Esto representa una mejoría en comparación con la limitación de ocho caracteres para los nombres de archivo del DOS. Además, un archivo de Windows puede tener más de una extensión.

Encabezado	Atributo de información	Atributo de nombre de archivo	Datos	Atributo de seguridad
------------	-------------------------	-------------------------------	-------	-----------------------

FIGURA 8-10 Un registro en la Tabla de archivos maestra del sistema NTFS.

Nombres de directorios y subdirectorios. El sistema de administración de archivos del DOS ordena los datos y programas en un disco en forma de directorios y subdirectorios. En Windows, los directorios y subdirectorios se llaman carpetas de archivos. Las reglas que se aplican a los nombres de los archivos también se aplican a los nombres de las carpetas. El disco está estructurado de forma que contenga un directorio raíz cuando se le da formato por primera vez. El directorio o carpeta raíz para un disco duro que se utiliza como unidad C es C:\. Cualquier otra carpeta se coloca en el directorio raíz. Por ejemplo, C:\DATOS es la carpeta DATOS en el directorio raíz. Cada carpeta que se coloca en el directorio raíz también puede tener subdirectorios o subcarpetas. Como ejemplos están las subcarpetas C:\DATOS\AREA1 y C:\DATOS\AREA2, en donde la carpeta DATOS contiene dos subcarpetas: AREA1 y AREA2. Las subcarpetas también pueden tener subcarpetas adicionales. Por ejemplo, C:\DATOS\AREA2\LISTA representa a la carpeta DATOS, subcarpeta AREA, la cual contiene una subcarpeta llamada LISTA.

Archivos de acceso secuencial

Todos los archivos de DOS y Windows son archivos secuenciales. Un archivo secuencial se almacena y se utiliza desde el principio del archivo hasta el final, en donde se accede al primer byte y a todos los bytes entre éste y el último para leer el último byte. Afortunadamente, se puede leer y escribir en los archivos en C++ mediante el uso del objeto CFile, lo cual facilita su acceso y manipulación. En esta sección se describirá cómo crear, leer, escribir, borrar y cambiar el nombre de un archivo de acceso secuencial.

Creación de archivos. Antes de poder utilizar un archivo, debe existir en el disco. Para crear un archivo se utiliza el objeto CFile con el atributo ::modeCreate, el cual indica a CFile que debe crear un archivo. El archivo se crea con la función miembro Open de CFile, como se muestra en el ejemplo 8-30. Aquí el nombre del archivo que crea el programa se almacena en una variable tipo CString llamada NombreArchivo. A continuación, el objeto CFile se inicializa como Archivo. Por último, la instrucción if abre y crea el archivo. Si el valor de retorno de Archivo.Open es falso significa que Windows no pudo crear el archivo, y si es verdadero, entonces el archivo se abrió sin errores. En este ejemplo, si el archivo no se abre debido a que el disco esté lleno o que no se encuentre la carpeta, aparece un cuadro de mensaje de Windows en el que se muestra la leyenda “No se pudo abrir el archivo:” seguida del nombre del archivo, y luego el programa termina cuando se hace clic en el botón Aceptar del cuadro de mensaje. Para probar este ejemplo, cree una aplicación de cuadro de diálogo y coloque el código en la función OnInitDialog, después de la instrucción TODO:. Elija un nombre de carpeta que no exista (probablemente funcione el nombre prueba) y ejecute la aplicación. Ahora deberá ver el mensaje de error. Si cambia el NombreArchivo de manera que no incluya la carpeta, no obtendrá el mensaje de error.

EJEMPLO 8-30

```
CString NombreArchivo = "C:\\prueba\\Prueba.txt";
CFile Archivo;

if ( !Archivo.Open( NombreArchivo, CFile::modeWrite | CFile::modeCreate ) )
{
    MessageBox("No se pudo abrir el archivo:" + NombreArchivo,
    "Información", MB_OK);
    exit(0);           //si no pudo abrirse el archivo, salir del programa
}

//ahora Prueba.txt existe en la carpeta prueba, con una longitud de 0 bytes
```

Cómo escribir en un archivo. Una vez creado un archivo, se puede escribir en él. De hecho, sería muy inusual crear un archivo sin escribir algo en él. Los datos se escriben en un archivo un byte a la vez. No existe una provisión para utilizar CFile para escribir cualquier otro tipo de datos en un archivo. Los datos siempre se escriben empezando en el primer byte en un archivo. El ejemplo 8-31 lista un programa que crea un archivo en el directorio raíz llamado Prueba1.txt, y almacena la letra A en cada uno de sus 256 bytes. Si ejecuta este código y analiza el archivo Prueba1.txt con el Bloc de notas, verá un archivo lleno con 256 letras A. Hay que tener en cuenta que se debe cerrar el archivo cuando se termina de escribir, mediante la función Close().

EJEMPLO 8-31

```
CString NombreArchivo = "C:\\Prueba1.txt";
CFile Archivo;
unsigned char Bufer[256];

if ( !Archivo.Open( NombreArchivo, CFile::modeWrite | CFile::modeCreate ) )
{
    AfxMessageBox("No se pudo abrir el archivo:" + NombreArchivo);
    exit(0);
}

for ( int a = 0; a < 256; a++ )           //llena el Búfer
    Bufer[a] = 'A';

Archivo.Write( Bufer, sizeof( Bufer ) );   //escribe el Búfer
Archivo.Close();                         //cierra el archivo
```

Suponga que debe escribirse un entero de 32 bits en un archivo. Como sólo pueden escribirse bytes, debe utilizarse un método para convertir los cuatro bytes del entero en un formato que pueda escribirse en un archivo. En C++ se utiliza LOBYTE para acceder a los 8 bits de menor orden de un entero corto y HIBYTE para acceder a los 8 bits de mayor orden de un entero corto. De igual forma, LOWORD accede a los 16 bits de menor orden de un entero de 32 bits y HIWORD accede a los 16 bits de mayor orden del entero de 32 bits. Estas cuatro funciones se utilizan en C++ para acceder a los bytes separados de un entero de 32 bits. Por ejemplo, HIWORD(LOBYTE(número)) accede a los bits 23 a 16 del número. También se puede realizar la misma tarea en lenguaje ensamblador en menos bytes, como se muestra en el ejemplo 8-32. Si analiza el código ensamblado para cada método, podrá ver que el método en lenguaje ensamblador es mucho más corto y veloz. Si la velocidad y el tamaño son importantes, entonces el código ensamblador es la mejor opción de todas.

EJEMPLO 8-32

```
int numero = 0x20000;
unsigned char Buf[4];

//conversión en C++
Buf[0] = LOWORD(LOBYTE(numero));
Buf[1] = LOWORD(HIBYTE(numero));
Buf[2] = HIWORD(LOBYTE(numero));
Buf[3] = HIWORD(HIBYTE(numero));

//conversión en lenguaje ensamblador

_asm
{
    mov eax,numero
    mov Buf[0],al
    mov Buf[1],ah
    bswap eax          ;tomado de little endian a big endian
    mov Buf[2],ah
    mov Buf[3],al
}
```

Cómo leer los datos de un archivo. Los datos de un archivo se leen desde el principio del archivo hacia el final, mediante el uso de la función Read de CFile. El ejemplo 8-33 muestra un programa que lee el archivo que se escribió en el ejemplo 8-31 y coloca los datos en un búfer. La función Read devuelve el número de bytes que se leen del archivo, pero no se utiliza en este ejemplo. Observe que el ejemplo utiliza la clase CFileNotFoundException para reportar cualquier error que pueda ocurrir al abrir el archivo. El mensaje de error de Windows aparece en la variable Unicode llamada Causa, mediante el uso del miembro GetErrorMessage de la clase CFileNotFoundException. También se muestra el cursor de espera (reloj de arena) en este ejemplo para mostrar que Windows está ocupado en el proceso de abrir y leer el archivo. Esto ocurre tan rápido en este ejemplo que probablemente no pueda verse, pero si se abre y se lee un archivo muy grande sí podría verse.

EJEMPLO 8-33

```

CString NombreArchivo = "C:\\Prueba1.txt";
CFile Archivo;
CFileException ex;

unsigned char Bufer[256];
unsigned int ConteoBytesArchivo;

CWaitCursor espera;                                //muestra el reloj de arena

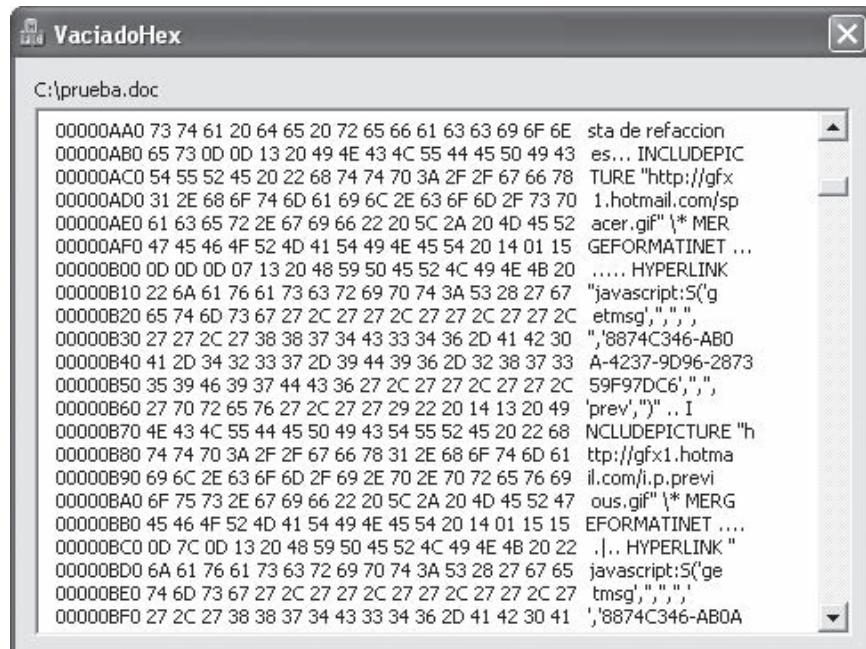
if ( !Archivo.Open( NombreArchivo, CFile::modeRead, &ex ) )
{
    TCHAR Causa[255];                            //si el archivo tiene error
    CString Cad;
    ex.GetErrorMessage(Causa, 255);
    Cad = _T("No se pudo abrir el archivo. Error = ");
    Cad += Causa;
    AfxMessageBox(Cad);                         //muestra el mensaje de error
}
else
{
    ConteoBytesArchivo = Archivo.Read(Bufer, sizeof( Bufer ) );
    Archivo.Close();
}

espera.Restore();                                 //regresa al cursor normal

```

Un programa de ejemplo de vaciado binario. Una herramienta que no está disponible en Windows es un programa que muestre el contenido de un archivo en código hexadecimal. Aunque tal vez muchos programadores no lo utilicen, es de mucha utilidad cuando se desarrolla software para poder ver el contenido actual de un archivo en formato hexadecimal. Empiece una aplicación de cuadro de diálogo en Windows MFC; utilice el nombre VaciadoHex. Coloque un control Active X del tipo Microsoft Forms 2.0 Label y un control Active X del tipo Microsoft Forms 2.0 TextBox en el formulario, como se muestra en la figura 8-11. Use los nombres Etiqueta1 para la etiqueta y Rico1 para el control de edición de texto. En Propiedades del control de texto, asegúrese de cambiar las propiedades Locked y MultiLine a True, y la propiedad ScrollBars a Vertical. Si desea mostrar un archivo muy grande, es conveniente poder desplazarse a través del código.

FIGURA 8-11 Captura de pantalla del programa VaciadoHex.



Este programa utiliza la función Muestrah que aparece en el ejemplo 8-25 para mostrar la dirección como una dirección hexadecimal de ocho dígitos y también para mostrar el contenido de la dirección en formato hexadecimal, como un número de dos dígitos. Agregue la función Muestrah a la clase Vaciado-HexDlg, de manera que su valor de retorno sea CString y que contenga dos parámetros enteros: uno para el número y otro para el número de dígitos, como se muestra en el ejemplo 8-25.

El programa en sí se coloca en la función OnInitDialog, justo debajo de la instrucción TODO:. El ejemplo 8-34 muestra el programa para realizar un vaciado hexadecimal. Observe que para cambiar el archivo que se va a vaciar en formato hexadecimal, hay que cambiar el nombre del archivo en el programa. Para ello puede utilizarse un cuadro de edición en el que se escriba el nombre del archivo, pero no lo hicimos en el ejemplo por cuestión de brevedad. En este programa se leen 16 bytes a la vez y se les da formato para poder mostrarlos en pantalla. Este proceso continúa hasta que no quedan bytes en el archivo y la variable ConteoBytesArchivo se regresa con el valor de cero. Los datos ASCII que se muestran al final del listado hexadecimal están filtrados, de manera que cualquier carácter ASCII menor de 32 (un espacio) se muestra como un punto. Esto es importante porque los caracteres de control tales como el salto de línea, el retroceso y demás destruirán el formato en pantalla del texto ASCII, y eso es algo indeseable.

EJEMPLO 8-34

```
CString NombreArchivo = "C:\\prueba.doc";           //cambie para mostrar un archivo
distinto
CFile Archivo;
CFileException ex;
unsigned char Bufer[16];
unsigned int ConteoBytesArchivo;
unsigned int Direccion = 0;

Etiqueta1.put_Caption(NombreArchivo);
CWaitCursor espera;
if ( !Archivo.Open( NombreArchivo, CFile::modeRead, &ex ) )
{
    TCHAR Causa[255];
    CString Cad;
    ex.GetErrorMessage(Causa, 255);
    Cad = _T("No se pudo abrir el archivo. Error = ");
    Cad += Causa;
    AfxMessageBox(Cad);
}
else
{
    while ( ( ConteoBytesArchivo = Archivo.Read( Bufer,
                                                sizeof( Bufer ) ) )!= 0)
    {
        CString ascii, linea;
        linea = Muestrah(Direccion, 8);
        for (int a = 0; a < ConteoBytesArchivo; a++)
        {
            linea += " " + Muestrah(Bufer[a], 2);
            if ( Bufer[a] >= 32 )
                ascii += Bufer[a];
            else
                ascii += ".";
        }
        Direccion += 16;
        Ricol.put_Text(Ricol.get_Text() + linea + " " + ascii + "\n");
    }
    Archivo.Close();
}
espera.Restore();
```

El apuntador de archivo y la función Seek. Cuando se abre o se lee un archivo, o cuando se escribe en él, el apuntador del archivo direcciona la posición actual en el archivo secuencial. Cuando se abre un archivo, el apuntador del archivo siempre direcciona el primer byte del archivo. Si un archivo es

de 1024 bytes y una función para leer el archivo lee 1023 bytes, el apuntador del archivo direcciona el último byte del archivo, pero no el final del mismo.

El **apuntador de archivo** es un número de 32 bits que direcciona cualquier byte en un archivo. La función miembro Seek de CFile se utiliza para modificar el apuntador de archivo. Un apuntador de archivo puede desplazarse desde el inicio del archivo (CFile::begin), desde la posición actual (CFile::current) o desde el final (CFile::end). En la práctica se utilizan las tres direcciones de movimiento para acceder a distintas partes del archivo. La distancia desplazada por el apuntador de archivo (en bytes) se especifica como el primer parámetro en una función Seek.

Suponga que existe un archivo en el disco y que debemos anexarle 256 bytes de información nueva. Al abrir el archivo, el apuntador de archivo direcciona el primer byte del mismo. Si trata de escribir sin desplazar el apuntador de archivo hasta el final del archivo, los nuevos datos *sobrescribirán* los primeros 256 bytes del mismo. El ejemplo 8-35 muestra una secuencia de instrucciones con las que se abre un archivo, se desplaza el apuntador de archivo hasta el final del mismo, se escriben 256 bytes de datos y luego se cierra el archivo. Con este programa se anexan 256 bytes de datos nuevos al archivo, provenientes del área Bufer.

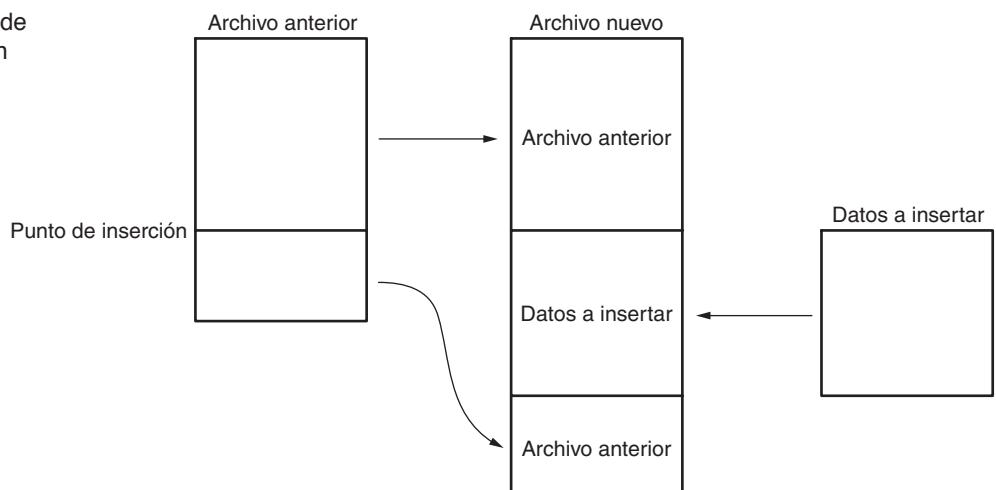
EJEMPLO 8-35

```
CString NombreArchivo = "C:\\pruebal.txt";
CFile Archivo;
CFileException ex;
unsigned char Bufer[256];

CWaitCursor espera;
if ( !Archivo.Open( NombreArchivo, CFile::modeWrite, &ex ) )
{
    TCHAR Causa[255];
    CString Cad;
    ex.GetErrorMessage(Causa, 255);
    Cad = _T("No se pudo abrir el archivo. Error = ");
    Cad += Causa;
    AfxMessageBox(Cad);
}
else
{
    Archivo.Seek(0, CFile::end); //va al final
    Archivo.Write(Bufer, 256); //anexa datos al archivo
    Archivo.Close();
}

espera.Restore();
```

FIGURA 8-12 Inserción de datos nuevos dentro de un archivo ya existente.



Una de las maniobras más difíciles con los archivos es la inserción de datos nuevos en medio de un archivo. La figura 8-12 muestra cómo se logra esto mediante la creación de un segundo archivo. Observe que la parte correspondiente al archivo antes del punto de inserción se copia en el archivo nuevo. Después de estos datos se anexa la nueva información antes de que se inserten los datos restantes del archivo anterior en el archivo nuevo. Una vez que está completo el archivo nuevo, el archivo anterior se elimina y se cambia el nombre del archivo nuevo por el del archivo anterior.

El ejemplo 8-36 muestra un programa que inserta datos nuevos en un archivo ya existente. Este programa copia el archivo NUEVO.DAT en el archivo ANTERIOR.DAT en la siguiente posición después de los primeros 256 bytes del archivo ANTERIOR.DAT. Los datos nuevos de Bufer2 se agregan al archivo y después se agrega el resto del archivo anterior. En este ejemplo utilizamos nuevas funciones miembro de CFile para eliminar el archivo anterior y cambiar el nombre del archivo nuevo por el del archivo anterior.

EJEMPLO 8-36

```
CString NombreArchivo1 = "C:\\\\Anterior.dat";
CString NombreArchivo2 = "C:\\\\Nuevo.dat";
CFile Archivo1;
CFile Archivo2;
CFileException ex;
unsigned char Bufer1[256];
unsigned char Bufer2[6];
unsigned int longitud;

CWaitCursor espera;
if ( !Archivo1.Open( NombreArchivo1, CFile::modeRead, &ex ) )
{
    TCHAR Causa[255];
    CString Cad;
    ex.GetErrorMessage(Causa, 255);
    Cad = _T("No se pudo abrir el archivo. Error = ");
    Cad += Causa;
    AfxMessageBox(Cad);
}
else
{
    if ( !Archivo2.Open( NombreArchivo2, CFile::modeCreate | CFile::modeWrite, &ex ) )
    {
        TCHAR Causa[255];
        CString Cad;
        ex.GetErrorMessage(Causa, 255);
        Cad = _T("No se pudo abrir el archivo. Error = ");
        Cad += Causa;
        AfxMessageBox(Cad);
    }
    else
    {
        Archivo1.Read(Bufer1, 256);
        Archivo2.Write(Bufer1, 256);
        Archivo2.Write(Bufer2, sizeof Bufer2);
        while ( ( longitud = Archivo1.Read(Bufer1, 256) ) != 0 )
            Archivo2.Write(Bufer1, longitud);
        Archivo1.Close();
        Archivo2.Close();
        CFile::Remove( NombreArchivo1 ); //elimina el archivo anterior
        CFile::Rename( NombreArchivo2, NombreArchivo1 ); //cambia el
                                                       nombre al
                                                       archivo nuevo
    }
}
espera.Restore();
```

Archivos de acceso aleatorio

Los archivos de acceso aleatorio se desarrollan a través de software, mediante el uso de archivos de acceso secuencial. Un archivo de acceso aleatorio se direcciona mediante un número de registro, en

vez de tener que desplazarse por el archivo en busca de los datos. La función Seek tiene un papel muy importante cuando se crean archivos de acceso aleatorio. Estos archivos son más fáciles de usar para volúmenes extensos de datos, que a menudo se les conoce como bases de datos.

Creación de un archivo de acceso aleatorio. La planeación es imprescindible cuando se crea un sistema de archivos de acceso aleatorio. Suponga que se requiere un archivo de acceso aleatorio para almacenar los nombres de los clientes. Cada uno de los registros de los clientes requiere 32 bytes para almacenar el apellido, 32 bytes para el primer nombre y un byte para la inicial del segundo nombre. El registro de cada cliente contiene dos líneas para el domicilio de 64 bytes cada una, una línea para la ciudad de 32 bytes, dos bytes para el código del estado y nueve bytes para el código postal. La información básica de los clientes requiere por sí sola de 236 bytes; la información adicional expande el registro a 512 bytes. Como el negocio está creciendo, se tiene previsto tener 500 clientes. Esto significa que la longitud total del archivo de acceso aleatorio será de 2,560,000 bytes.

El ejemplo 8-37 muestra un programa corto que crea un archivo llamado CLIENT.DAT e inserta 5000 registros en blanco de 512 bytes cada uno. Un registro en blanco contiene el valor 00H en cada byte. Éste parece ser un archivo extenso, pero cabe hasta en el más pequeño de los discos duros.

EJEMPLO 8-37

```
CString NombreArchivo = "C:\\\\Cliente.dat";
CFile Archivo;

CFileException ex;
unsigned char Bufer[512];

for ( int a = 0; a < 512; a++ )           //llena el búfer
    Bufer[a] = 0;

CWaitCursor espera;
if ( !Archivo.Open( NombreArchivo, CFile::modeCreate | CFile::modeWrite, &ex ) )
{
    TCHAR Causa[255];
    CString Cad;
    ex.GetErrorMessage(Causa, 255);
    Cad = _T("No se pudo abrir el archivo. Error = ");
    Cad += Causa;
    AfxMessageBox(Cad);
}
else
{
    for ( int a = 0; a< 5000; a++ )           //crea 5000 registros
        Archivo.Write(Bufer, 512);
    Archivo.Close();
}
espera.Restore();
```

Lectura y escritura de un registro. Cada vez que debe leerse un registro, el número del registro se encuentra mediante el uso de la función Seek. El ejemplo 8-38 muestra una función que se utiliza para buscar un registro con Seek. Esta función asume que se ha abierto un archivo como ArchivoCliente y que CLIENTE.DAT permanece abierto en todo momento.

Observe cómo se multiplica el número de registro por 512 para obtener un conteo con el que se desplaza el apuntador de archivo mediante Seek. En cada caso, el apuntador de archivo se desplaza desde el inicio del archivo hasta el registro deseado.

EJEMPLO 8-38

```
void CBaseDatosClientDlg::BuscaRegistro(unsigned int NumeroRegistro)
{
    Archivo.Seek(NumeroRegistro * 512, CFile::begin );
```

Se necesitan otras funciones (las cuales se muestran en el ejemplo 8-39) para administrar la base de datos de los clientes. Éstas son EscribeRegistro, LeeRegistro, BuscaApellidoRegistro, BuscaRegistroEnBlanco, etc.

Algunas de estas funciones se muestran en el ejemplo, así como la estructura de datos que contiene la información para cada registro.

EJEMPLO 8-39

```

struct BASE {                                     //registro de base de datos
    char PrimerNombre[32];
    char IniSegNombre[1];
    char Apellido[32];
    char Calle1[64];
    char Calle2[64];
    char Ciudad[32];
    char Estado[2];
    char CodPostal[9];
    char Otros[276];
}client[2];                                //prepara 2 registros

void CBaseDatosClientDlg::EscribeRegistro(unsigned int NumeroRegistro)
{
    BuscaRegistro(NumeroRegistro);
    Archivo.Write(client[0].PrimerNombre, 32);
    Archivo.Write(client[0].IniSegNombre, 1);
    Archivo.Write(client[0].Apellido, 32);
    Archivo.Write(client[0].Calle1, 64);
    Archivo.Write(client[0].Calle2, 64);
    Archivo.Write(client[0].Ciudad, 32);
    Archivo.Write(client[0].Estado, 2);
    Archivo.Write(client[0].CodPostal, 9);
}

void CBaseDatosClientDlg::LeeRegistro(unsigned int NumeroRegistro)
{
    BuscaRegistro(NumeroRegistro);
    Archivo.Read(client[0].PrimerNombre, 32);
    Archivo.Read(client[0].IniSegNombre, 1);
    Archivo.Read(client[0].Apellido, 32);
    Archivo.Read(client[0].Calle1, 64);
    Archivo.Read(client[0].Calle2, 64);
    Archivo.Read(client[0].Ciudad, 32);
    Archivo.Read(client[0].Estado, 2);
    Archivo.Read(client[0].CodPostal, 9);
}

unsigned int CBaseDatosClientDlg::BuscaPrimerNombre(void) //usa client[1].
{
    PrimerNombre
    {
        for ( int a = 0; a < 5000; a++ )
        {
            LeeRegistro(a);
            if ( !strcmp(client[0].PrimerNombre, client[1].PrimerNombre) )
                return a;           //si se encuentra devuelve el número de
                                      //registro
        }
        return 5000;             //si no se encuentra devuelve 5000
    }
}

unsigned int CBaseDatosClientDlg::BuscaRegistroEnBlanco(void)
{
    for ( int a = 0; a < 5000; a++ )
    {
        LeeRegistro(a);
        if ( !strlen(client[0].Apellido) == 0 )
            return a;
    }
    return 0;
}

```

8-5**PROGRAMAS DE EJEMPLO**

Ahora que hemos hablado sobre muchos de los fundamentos básicos de programación, presentaremos algunos programas de aplicaciones de ejemplo. Aunque estos programas de ejemplo puedan parecer triviales, muestran algunas técnicas de programación adicionales, además de que ilustran los estilos de programación para el microprocesador.

Programa para mostrar la hora/fecha

Aunque este programa no utiliza lenguaje ensamblador, demuestra cómo obtener la fecha y la hora de la API de Windows y cómo darle formato para mostrarla en pantalla. También demuestra cómo se utiliza un temporizador en Visual C++. El ejemplo 8-40 muestra un programa que utiliza un temporizador, ajustado para interrumpir el programa una vez por segundo, para mostrar la hora y la fecha. Estos datos se obtienen mediante el uso de la función miembro COleTimeDate::GetCurrentTime() para leer la hora y la fecha de la computadora y colocar la información en una variable llamada HoraFecha. El miembro Format de COleTimeDate se utiliza para dar formato a la variable HoraFecha. Cree una aplicación de cuadro de diálogo que se llame HoraFecha y coloque dos controles Active X tipo Microsoft Forms 2.0 Label, como se muestra en la figura 8-13. Use los nombres EtiquetaHora para la etiqueta superior y EtiquetaFecha para la otra etiqueta.

La tabla 8-4 muestra las opciones de formato (Format) para la variable COleTimeDate, de manera que puedan utilizarse otros estilos para mostrar una hora o una fecha. Este objeto contiene la fecha y hora correctas desde enero 1, 100 hasta diciembre 31, 0000. Asegúrese de utilizar este objeto para todas las horas y fechas, para no tener otro problema como el del año 2000, por lo menos no hasta el año 10,000.

EJEMPLO 8-40

```
//Coloque estas dos líneas después de TODO: en la función OnInitDialog  
HoraFecha();  
SetTimer(1, 1000, 0);
```

FIGURA 8-13 La hora y la fecha se muestran en pantalla.



TABLA 8-4 El comando Format para fecha y hora.

Código	Función
%a	Nombre del día de la semana abreviado.
%A	Nombre del día de la semana completo.
%b	Nombre del mes abreviado.
%B	Nombre del mes completo.
%c	Fecha y hora para la configuración regional apropiada.
%d	Día del mes (01-31).
%H	Formato de 24 horas (00-23).
%I	Formato de 12 horas (01-12).
%j	Día del año (001-366).
%m	Mes (01-12).
%M	Minuto (00-59).
%p	AM/PM actual para la configuración regional.
%S	Segundo (00-59).
%U, %W	Semana (00-53) (%U = Domingo es el primer día) (%W = Lunes es el primer día).
%w	Número del día de la semana (0-6: Domingo es 0).
%x	Fecha para la configuración regional.
%X	Hora para la configuración regional.
%y	Año (00-99).
%Y	Año (100-9999).
%z, %Z	Zona horaria.
%%	Signo de porcentaje.
#	Elimina el cero a la izquierda.

```
//Funciones para la aplicación HoraFecha

void CHoraFechaDlg::HoraFecha(void)
{
    COleDateTime FechaHora = COleDateTime::GetCurrentTime();
    EtiquetaHora.put_Caption( FechaHora.Format( "%#I:%M:%S %p" ) );
    EtiquetaFecha.put_Caption( FechaHora.Format( "%A %B %#d, %Y" ) );
}

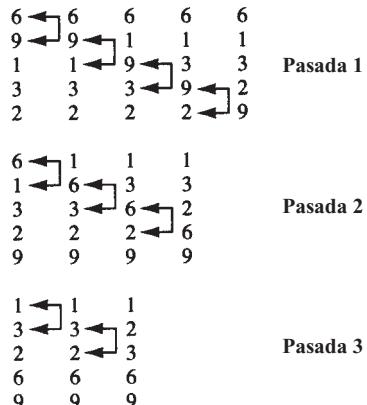
void CHoraFechaDlg::OnTimer(UINT nIDEvent)
{
    HoraFecha();
    CDialog::OnTimer(nIDEvent);
}
```

La función que vuelve a llamar al temporizador (OnTimer) se inserta en la sección Propiedades al hacer clic en WM_TIMER. En esta aplicación, la función OnTimer se llama una vez por segundo (1000 milisegundos). Por lo general se comprueba el evento nIDEvent, pero como sólo se utiliza un temporizador en esta aplicación, no hay necesidad de comprobar el número de temporizador (en este caso 1, el primer parámetro en SetTimer). La función FechaHora.Format convierte la fecha o la hora en una cadena de caracteres que se muestra en pantalla, según la cadena de formato seleccionada, en los controles EtiquetaHora o EtiquetaFecha. Este código puede insertarse con facilidad en cualquier aplicación para mostrar la hora/fecha o ambas, en cualquier formato que se desee.

Programa para ordenar números

Algunas veces es necesario ordenar números. Esto se realiza a menudo con un ordenamiento tipo burbuja. La figura 8-14 muestra cinco números que se ordenan con este método. Observe que el conjunto de cinco números se evalúa cuatro veces con cuatro pasadas. En cada pasada se comparan dos números

FIGURA 8-14 El método de ordenamiento tipo burbuja que muestra los datos a medida que se ordenan. Nota: Para ordenar cinco números pueden requerirse cuatro pasadas.

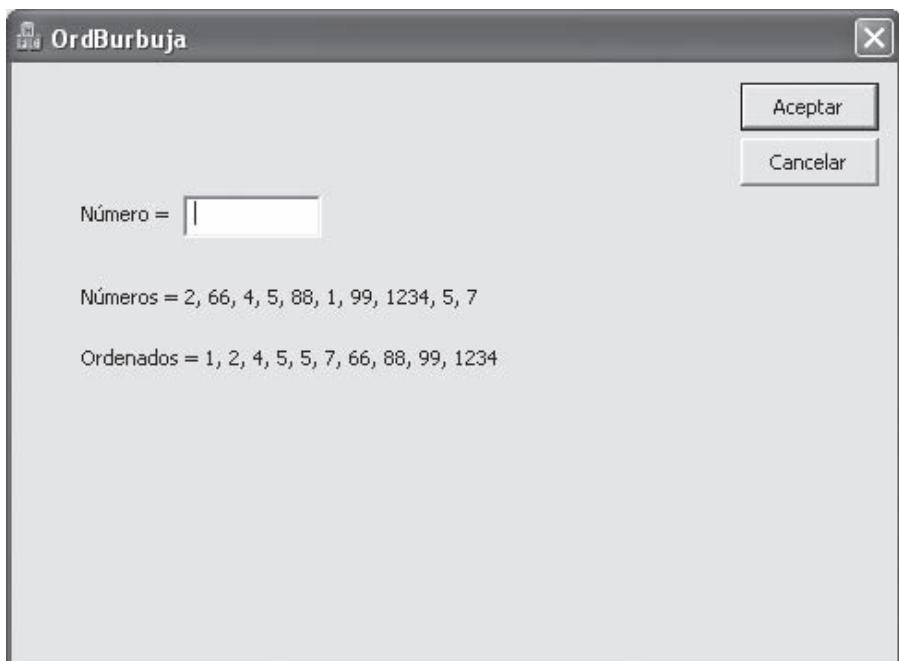


consecutivos y algunas veces se intercambian. Además observe que durante la primera pasada hay cuatro comparaciones, durante la segunda hay tres, y así sucesivamente.

El ejemplo 8-41 muestra un programa que acepta 10 números desde el teclado (enteros de 32 bits). Después de aceptar estos números de 32 bits y de almacenarlos en una sección de memoria llamada Numeros, se ordenan mediante la técnica de ordenamiento tipo burbuja. Esta técnica de burbuja utiliza una bandera de intercambio para determinar si se intercambiaron números en una pasada. Si no se intercambiaron números, significa que están en orden y el ordenamiento termina. Esta terminación temprana por lo general incrementa la eficiencia del ordenamiento, ya los números muy raras veces están desordenados por completo.

Una vez que se ordenan los números, se muestran en orden ascendente. Para especificar un arreglo, vaya al encabezado de la clase Dlg e inserte en forma manual el arreglo tipo unsigned int llamado Numeros[0]. En el ejemplo 8-41 también se muestra el contenido de la sección pública del encabezado con el programa. La figura 8-15 muestra cómo aparece la aplicación después de ejecutarla.

FIGURA 8-15 La aplicación para ordenar números mediante el método de burbuja.



EJEMPLO 8-41

```

//Parte del encabezado
// Implementación
protected:
    HICON m_hIcon;

    // Funciones de asignación de mensajes generadas
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    DECLARE_MESSAGE_MAP()
public:
    CEdit Edicion1;
    CLabel1 Etiqueta1;
    CLabel1 Etiqueta2;
    int Conteo;
    UINT Numeros[10];           //se agregó al encabezado en forma manual
    virtual BOOL PreTranslateMessage(MSG* pMsg);
};

//la función OnInitDialog después de TODO:

Edicion1.SetFocus();
Etiqueta1.put_Caption("Números = ");
Etiqueta2.put_Caption("Ordenados = ");
Conteo = 0;

return FALSE; // Devuelve TRUE a menos que establezca el foco en un
               control
}

//La función PreTranslateMessage (debe insertarse en las propiedades)

BOOL COrdBurbujaDlg::PreTranslateMessage(MSG* pMsg)
{
    if ( Conteo != 10 && pMsg->message == WM_KEYDOWN && pMsg->wParam == 13 )
    {
        CString temp;
        char bandera;
        char temp1[10];
        UINT *pNumeros = &Numeros[0]; //obtiene el apuntador del arreglo
        Numeros[Conteo] = GetDlgItemInt(IDC_EDIT1);
        GetDlgItemText(IDC_EDIT1, temp);
        SetDlgItemText(IDC_EDIT1, "");
        Etiqueta1.put_Caption(Etiqueta1.get_Caption() + temp);
        if ( Conteo != 9 )
            Etiqueta1.put_Caption(Etiqueta1.get_Caption() + ", ");
        Conteo++;
        if ( Conteo == 10 )
        {
            __asm{
                mov ecx, 9          ;9 para 10 números
L1:
                mov bandera, 0      ;borra la bandera
                mov edx, 0
L2:
                mov ebx, pNumeros
                mov eax, [ebx+edx*4]
                cmp eax, [ebx+edx*4+4]
                jbe L3
                push eax           ;intercambio
                mov eax, [ebx+edx*4+4]
                mov [ebx+edx*4], eax
                pop dword ptr [ebx+edx*4+4]
                mov bandera, 1      ;establece la bandera
L3:
                inc edx
                cmp edx, ecx
                jne L2
            }
        }
    }
}

```

```

        cmp bandera,0
        jz L4                         ;si no hay intercambios
        loop L1
L4:
}
for ( int a = 0; a < 9; a++ )
    Etiqueta2.put_Caption(Etiqueta2.get_Caption() +
        itoa(Numeros[a], temp1, 10) + ", ");
    Etiqueta2.put_Caption(Etiqueta2.get_Caption() +
        itoa(Numeros[9],
            temp1, 10));
}
return true;
}
return CDialog::PreTranslateMessage(pMsg);
}

```

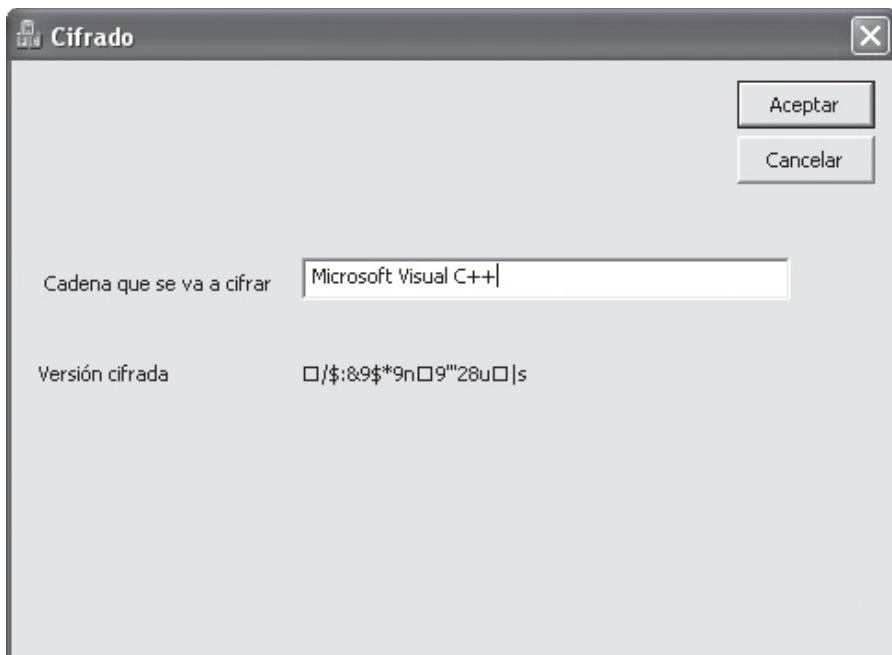
Cifrado de datos

Parece ser que el cifrado de datos está de moda, debido al aspecto relacionado con la seguridad de muchos sistemas. Para ilustrar el cifrado simple de datos para una cadena de caracteres, suponga que se aplica un OR exclusivo entre cada uno de los caracteres de una cadena y un número, al cual se le llama clave de cifrado. Esta operación realmente cambia el código del carácter, pero para hacerlo más aleatorio, suponga que la clave de cifrado se cambia después de que se cifra cada carácter. De esta manera, es mucho más difícil detectar los patrones en el mensaje cifrado, con lo que se dificulta más el proceso de descifra el mensaje.

Para ilustrar este esquema simple, la figura 8-16 muestra una captura de pantalla del programa para evaluar el esquema mediante el uso de un control de edición para aceptar una cadena de caracteres y una etiqueta Active X para mostrar el mensaje cifrado. Este ejemplo se generó mediante el uso de una clave de cifrado inicial de 0x45. Si se cambia el valor inicial, cambiará el mensaje cifrado.

El ejemplo 8-42 lista el programa que se utiliza para generar el mensaje en formato cifrado en la Etiqueta1, la etiqueta Active X. La primera sección es el diálogo de inicialización que se utiliza para iniciar el programa y establecer el enfoque en Edicion1, el control de edición para introducir la cadena de caracteres que se va a cifrar. Observe cómo el programa utiliza lenguaje ensamblador para aplicar un OR exclusivo entre cada carácter de la cadena y la ClaveDeCifrado, y cómo se modifica después esta

FIGURA 8-16 Captura de pantalla del programa de cifrado.



ClaveDeCifrado para el siguiente carácter. Esta clave puede modificarse en cualquier momento para dificultar aún más el proceso de descifrar el mensaje. Por ejemplo, suponga que la clave se incrementa en todos los demás caracteres después del primero y que se alterna con la clave invertida, como se muestra en el ejemplo 8-43. Puede utilizarse casi cualquier combinación de operaciones para modificar la clave entre cada pasada para dificultar de manera considerable la decodificación del mensaje. En la práctica utilizamos una clave de 128 bits y la técnica de modificación es distinta; no obstante, ésta es básicamente la forma en que se realiza el cifrado. Como el ejemplo 8-42 utiliza una clave de 8 bits, el mensaje cifrado podría descubrirse si se prueban todas las 256 (2^8) claves posibles, pero si se utiliza una clave de 128 bits, se requieren muchos más intentos (2^{128}) para descubrirla (un número de intentos casi imposible de probar por completo).

EJEMPLO 8-42

```
//código que se agrega a la función OnInitDialog en la clase Dlg
    // TODO: agregar aquí inicialización adicional
    Edicion1.SetFocus();

    return FALSE; // Devuelve TRUE a menos que establezca el foco en un
                  control
}

//la función PreTranslateMessage

BOOL CCifradoDlg::PreTranslateMessage(MSG* pMsg)
{
    char ClaveDeCifrado = 0x45;           //Clave aleatoria
    CString temp;
    if ( pMsg->message == WM_KEYDOWN && pMsg->wParam == 13 )
    {
        GetDlgItemText(IDC_EDIT1, temp);
        for ( int a = 0; a < temp.GetLength(); a++ )
        {
            char codigo = temp.GetAt(a);
            __asm
            {
                mov al,codigo
                xor al,ClaveDeCifrado
                mov codigo,al
                inc ClaveDeCifrado
            }
            temp.SetAt(a, codigo);
            Etiqueta1.put_Caption(temp);
        }
        return true;
    }
    return CDialog::PreTranslateMessage(pMsg);
}
```

EJEMPLO 8-43

```
//sólo la parte de lenguaje ensamblador del programa

__asm
{
    mov al,codigo
    xor al,ClaveDeCifrado
    mov codigo,al
    mov eax,a
    shr eax,1
    jc L1
    inc ClaveDeCifrado
    jmp L2
L1:
L2:
}
```

8-6**RESUMEN**

1. El programa ensamblador (ML.EXE) ensambla módulos que contienen variables y segmentos PUBLIC, además de variables EXTRN (externas). El programa enlazador (LINKER.EXE) enlaza los módulos y los archivos de biblioteca para crear un programa en tiempo de ejecución, que se ejecuta desde la línea de comandos del DOS. Este programa por lo general tiene la extensión EXE, pero puede tener también la extensión COM.
2. Las directivas MACRO y ENDM crean un nuevo código de operación para usarlo en los programas. Estas macros son similares a los procedimientos, sólo que no hay llamada ni retorno. En vez de ello, el ensamblador inserta el código de la macro secuencia en un programa cada vez que ésta se invoca. Las macros pueden incluir variables que pasen información y datos a la macro secuencia.
3. Para establecer el enfoque en un objeto se utiliza la variable miembro SetFocus(). Esta variable puede utilizarse casi en todos los objetos.
4. Las funciones SetDlgItemInt y SetDlgItemText se utilizan para colocar datos en un objeto.
5. Para utilizar el controlador del ratón en Windows hay que instalar manejadores para los diversos mensajes de Windows relacionados con el ratón, como WS_MOUSEMOVE, etc.
6. Para convertir de binario a BCD se utiliza la instrucción AAM cuando los números son menores de 100, o mediante varias divisiones entre 10 cuando los números son mayores. Una vez que el número se convierte en BCD, se suma 30H a cada dígito para convertirlo en código ASCII y poder mostrarlo en pantalla.
7. Para convertir un número de ASCII a BCD, se resta 30H a cada dígito. Para obtener el equivalente en binario, se multiplica por 10 y luego se suma cada nuevo dígito.
8. Las tablas de búsqueda se utilizan para la conversión de código con la instrucción XLAT si el código es de 8 bits. Si el código es mayor de ocho bits, se utiliza un procedimiento corto que accede a una tabla de búsqueda. Estas tablas también se utilizan para almacenar direcciones, de manera que puedan seleccionarse distintas partes de un programa o distintos procedimientos.
9. Las instrucciones condicionales de lenguaje ensamblador permiten ensamblar porciones de un programa si se cumple cierta condición. Estas instrucciones son útiles para personalizar el software, de manera que se adapte a cierta aplicación.
10. El sistema de memoria en disco contiene pistas que almacenan la información contenida en sectores. Muchos sistemas de disco almacenan 512 bytes de información por sector. Los datos en el disco se organizan de la siguiente forma: un sector de arranque, una tabla de asignación de archivos, un directorio raíz y un área para almacenar datos. El sector de arranque carga el sistema DOS del disco y lo coloca en el sistema de memoria de la computadora. La FAT o la MFT indica qué sectores están presentes y si contienen datos o no. El directorio raíz contiene nombres de archivos y subdirectorios, por medio de los cuales se accede a todos los archivos en el disco. El área de almacenamiento de datos contiene todos los subdirectorios y archivos de datos.
11. En Visual C++, los archivos se manipulan mediante el objeto Cfile. Para leer un archivo en disco, éste se abre, se lee y después se cierra. Para escribir en un archivo en disco, éste se abre, se escribe en él y después se cierra. Cuando se abre un archivo, el apuntador de archivo direcciona el primer byte del mismo. Para acceder a los datos en otras posiciones, el apuntador de archivo se desplaza mediante la función Seek, antes de leer o escribir datos.
12. Un archivo de acceso secuencial se accede en forma secuencial desde el principio hasta el fin. Un archivo de acceso aleatorio se accede en cualquier punto. Aunque todos los archivos en disco son secuenciales, pueden tratarse como archivos de acceso aleatorio mediante el uso de software.

8-7**PREGUNTAS Y PROBLEMAS**

1. El ensamblador convierte un archivo fuente en un archivo _____.
2. ¿Qué archivos se generan a partir del archivo fuente PRUEBA.ASM, si éste se procesa mediante ML.EXE?

3. El programa enlazador vincula los archivos objeto y los archivos _____ para crear un archivo ejecutable.
4. ¿Qué indica la directiva PUBLIC cuando se coloca en el módulo de un programa?
5. ¿Qué indica la directiva EXTERN cuando se coloca en el módulo de un programa?
6. ¿Qué directiva aparece con las etiquetas definidas como externas?
7. Describa el funcionamiento de un archivo de biblioteca que se enlaza a otros archivos objeto mediante el programa enlazador.
8. ¿Qué directivas de lenguaje ensamblador delinean una macrosecuencia?
9. ¿Qué es una macrosecuencia?
10. ¿Cómo se transfieren los parámetros a una macrosecuencia?
11. Desarrolle una macro llamada SUMA32 que sume el contenido de 32 bits de DX-CX al contenido de 32 bits de BX-AX.
12. ¿Cómo se utiliza la directiva LOCAL dentro de una macrosecuencia?
13. Desarrolle una macro llamada SUMALISTA PARA1,PARA2 que sume el contenido de PARA1 y de PARA2. Cada uno de estos parámetros representa un área de la memoria. El número de bytes a sumar se indica mediante el registro CX antes de llamar a la macro.
14. Desarrolle una macro que sume una lista de datos tipo byte, la cual pueda llamarse mediante la macro SUMAM LISTA, LONGITUD. La etiqueta LISTA es la dirección inicial del bloque de datos y LONGITUD es el número de datos a sumar. El resultado debe ser una suma de 16 bits que se coloque en AX al final de la macro secuencia.
15. ¿Cuál es el propósito de la directiva INCLUDE?
16. Modifique la función del ejemplo 8-12 de manera que sólo filtre los números del 0 al 9 y que ignore todos los demás caracteres.
17. Modifique la función del ejemplo 8-12 de manera que genere un número aleatorio de 8 bits en la variable de clase tipo char, llamada Aleatorio. (Sugerencia: incremente la variable Aleatorio cada vez que se llame a la función PreTranslateMessage, sin importar el mensaje de Windows que se emita.)
18. Modifique el software que desarrolló en la pregunta 17 de manera que genere un número aleatorio entre 9 y 62.
19. Modifique la función que se muestra en el ejemplo 8-15, de manera que los números hexadecimales utilicen las letras minúsculas de la “a” a la “f”, en vez de las letras mayúsculas.
20. Modifique el ejemplo 8-17 de manera que se realice un desplazamiento/desplazamiento cíclico a la derecha o a la izquierda. Para ello, agregue un botón de comando para seleccionar la dirección.
21. ¿Qué manejadores se utilizan para acceder al ratón en el entorno de programación de Visual C++?
22. ¿Cómo se detecta el botón derecho del ratón en un programa?
23. ¿Cómo se detecta un doble clic con el ratón?
24. Desarrolle software que detecte cuando se opriman los botones izquierdo y derecho del ratón al mismo tiempo.
25. ¿Qué color se muestra para el valor 0x00ffff?
26. ¿Qué se muestra cuando se utiliza el número de color 0x80000010 para establecer un color?
27. Cuando se convierte un número de binario a BCD, la instrucción _____ realiza la conversión siempre y cuando el número sea menor que el 100 decimal.
28. ¿Cómo se convierte un número grande (mayor del 100 decimal) de binario a BCD?
29. ¿Cómo podría mostrarse un número binario como número octal?
30. Un dígito BCD se convierte en código ASCII si se le suma un _____.
31. Un número en código ASCII se convierte en BCD si se le resta _____.
32. Desarrolle una función que lea un número ASCII en IDC_EDIT1, mediante el uso de la función GetDlgItemText desde el teclado, y que lo devuelva como número tipo unsigned int. El número en el cuadro de edición es un número octal que se convierte en binario mediante la función.
33. Explique cómo se convierte en binario un número de tres dígitos en código ASCII.
34. Desarrolle una función que convierta todas las letras minúsculas en código ASCII a letras mayúsculas en código ASCII. Su procedimiento no podrá cambiar ningún otro carácter, excepto las letras de la “a” a la “z”, y deberá devolver el carácter convertido como un valor tipo char.

35. Desarrolle una tabla de búsqueda que convierta los datos hexadecimales 00H-FFH en los caracteres en código ASCII que representan a esos dígitos hexadecimales. Asegúrese de mostrar la tabla de búsqueda y cualquier software requerido para la conversión. (Sugerencia: cree una función para realizar la conversión.)
36. Explique el propósito del sector de arranque, de la FAT y del directorio raíz en el sistema FAT.
37. Explique el propósito de la MFT en el sistema de archivos NTFS.
38. La superficie de un disco se divide en pistas, que a su vez se subdividen en _____.
39. ¿Qué es un programa con la secuencia inicial de instrucciones de arranque (bootstrap loader) y dónde se encuentra?
40. ¿Qué es un clúster?
41. El sistema de archivos NTFS por lo general utiliza clústeres de _____ bytes de longitud.
42. ¿Cuál es la máxima longitud de un archivo?
43. ¿Qué código se utiliza para almacenar el nombre de un archivo cuando se utilizan nombres largos para los archivos?
44. Los nombres de archivos del DOS tienen como máximo _____ caracteres de longitud.
45. ¿Cuántos caracteres aparecen por lo general en una extensión?
46. ¿Cuántos caracteres pueden aparecer en un nombre largo de archivo?
47. Desarrolle un programa que abra un archivo llamado PRUEBA.LST, que lea 512 bytes del archivo y los coloque en el arreglo de memoria Arreglo y que cierre el archivo.
48. Muestre cómo cambiar el nombre del archivo PRUEBA.LST por el de PRUEBA.LIS.
49. ¿Cuál es el propósito de la función miembro Remove de CFile?
50. ¿Qué es un control Active X?
51. Escriba un programa que lea cualquier número decimal entre 0 y 2G, y que muestre la versión binaria de 16 bits en pantalla.
52. Escriba un programa que muestre las potencias binarias de 2 (en decimal) en la pantalla, para las potencias de la 0 a la 7. Su pantalla debe mostrar $2^n =$ valor para cada potencia de 2.
53. Utilice la técnica descrita en la pregunta 17 para desarrollar un programa que muestre números aleatorios entre 1 y 47 (o cualesquiera) para la lotería de su estado.
54. Modifique el programa del ejemplo 8-29, de manera que muestre las letras A, b, C, d, E y F para una pantalla hexadecimal de siete segmentos.
55. Modifique el ejemplo 8-42 para cifrar el mensaje mediante un algoritmo que diseñe.
56. Desarrolle una función llamada Descifrado (para un valor CString) para acompañar al cifrado de la pregunta 55.

CAPÍTULO 9

Especificación de hardware del 8086/8088

INTRODUCCIÓN

En este capítulo describiremos las funciones de las terminales de los microprocesadores 8086 y 8088, y proporcionaremos detalles sobre temas de hardware como la generación del reloj, uso de búferes en los buses, cierre de buses, sincronización, estados de espera y la comparación entre la operación en modo mínimo y la operación en modo máximo. Explicaremos primero el funcionamiento de estos microprocesadores debido a su estructura simple, lo cual nos servirá como una introducción a la familia de microprocesadores Intel.

Antes de poder conectar o utilizar una interfaz con cualquier cosa relacionada con el microprocesador, es necesario comprender las funciones de las terminales y la sincronización. Por lo tanto, la información de este capítulo es esencial para tener una completa comprensión de la memoria y el uso de interfaces de E/S, lo cual cubriremos en capítulos posteriores de este libro.

OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

1. Describir la función de cada terminal de los microprocesadores 8086 y 8088.
2. Comprender las características de corriente directa (CD) del microprocesador e indicará su factor de salida para las familias lógicas comunes.
3. Utilizar el chip generador de reloj (8284A) para proveer el reloj al microprocesador.
4. Conectar búferes y cierres a los buses.
5. Interpretar los diagramas de sincronización.
6. Describir los estados de espera y conectar los circuitos requeridos para producir diversos números de esperas.
7. Explicar la diferencia entre la operación en modo mínimo y la operación en modo máximo.

9-1

DISTRIBUCIÓN DE TERMINALES Y LAS FUNCIONES DE CADA TERMINAL

En esta sección explicaremos la función y (en ciertos casos) las múltiples funciones de cada una de las terminales del microprocesador. Además hablaremos sobre las características a CD, lo cual proporcionará una base para comprender las secciones posteriores sobre el uso de búferes y cierres.

La distribución de las terminales (pin-out)

La figura 9-1 ilustra la distribución de las terminales de los microprocesadores 8086 y 8088. Una estrecha comparación nos revela que no hay diferencia entre estos dos microprocesadores; ambos vienen en paquetes **duales en línea** de 40 terminales (DIPs).

Como dijimos en el capítulo 1, el 8086 es un procesador de 16 bits con un bus de datos de 16 bits y el 8088 es un microprocesador de 16 bits con un bus de datos de 8 bits. (Como lo muestran los diagramas de terminales, el 8086 tiene las conexiones de terminal AD₀-AD₁₅ y el 8088 tiene las conexiones de terminal AD₀-AD₇.) La anchura del bus de datos es entonces la única diferencia principal entre estos dos microprocesadores. Esto permite al 8086 transferir datos de 16 bits con más eficiencia.

No obstante, existe una pequeña diferencia en las señales de control. El 8086 tiene una terminal M/IO y el 8088 tiene una terminal IO/M. La otra única diferencia de hardware aparece en la terminal 34 de ambos circuitos integrados: en el 8088 es una terminal SS0 y en el 8086 es una terminal BHE/S7.

Requerimientos de la fuente de poder

Tanto el microprocesador 8086 como el 8088 requieren +5.0 V con una tolerancia en el voltaje de suministro de $\pm 10\%$. El 8086 utiliza una corriente de suministro máxima de 360 mA y el 8088 utiliza una corriente máxima de 340 mA. Ambos microprocesadores operan en temperaturas ambientales de entre 32 °F y 180 °F. Este rango no es lo bastante amplio como para usarse al exterior en el invierno o incluso en el verano, pero hay disponibles versiones de estos microprocesadores con rango de temperatura extendido. También existe una versión CMOS, la cual requiere una corriente de suministro muy baja y tiene un rango de temperatura extendido. Los microprocesadores 80C88 y 80C86 son versiones CMOS que sólo requieren 10 mA de corriente de suministro y funcionan en temperaturas desde los -40 °F hasta los +225 °F.

Características CD

No se debe conectar algo a las terminales del microprocesador sin conocer el requerimiento de corriente de entrada para una terminal de entrada, y la capacidad de control de la corriente de salida para una terminal de salida. Este conocimiento permite al diseñador de hardware seleccionar los componentes de interfaz apropiados para usarlos con el microprocesador, sin el temor de dañar algo.

Características de entrada. Las características de entrada de estos microprocesadores son compatibles con todos los componentes lógicos estándar de la actualidad. La tabla 9-1 muestra los niveles de voltaje de entrada y los requerimientos de corriente de entrada para cualquier terminal de entrada, en cualquiera de los dos microprocesadores. Los niveles de corriente de entrada son muy pequeños, ya que las entradas son las conexiones de compuerta de MOSFETs, y representan sólo corrientes de fuga.

Características de salida. La tabla 9-2 muestra las características de todas las terminales de salida de estos microprocesadores. El nivel de voltaje para el 1 lógico del 8086/8088 es compatible con el de la mayoría de las familias lógicas estándar, pero el nivel de 0 lógico no. Los circuitos lógicos estándar tienen un voltaje máximo de 0.4 V para el 0 lógico, y el 8086/8088 tiene un máximo de 0.45 V. Por lo tanto, hay una diferencia de 0.05 V.

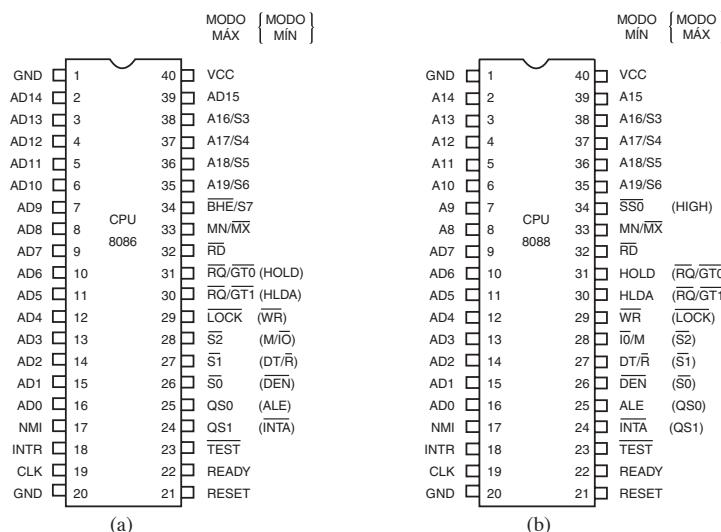


FIGURA 9-1 (a) El diagrama de terminales del microprocesador 8086; (b) el diagrama de terminales del microprocesador 8088.

TABLA 9-1 Características de entrada de los microprocesadores 8086 y 8088.

Nivel lógico	Voltaje	Corriente
0	0.8 V máximo	$\pm 10 \mu\text{A}$ máximo
1	2.0 V mínimo	$\pm 10 \mu\text{A}$ máximo

Esta diferencia reduce la inmunidad al ruido, del nivel estándar de 400 mV (0.8 V – 0.45 V) a 350 mV. (La **inmunidad al ruido** es la diferencia entre los niveles de voltaje de salida y de voltaje de entrada de un 0 lógico.) La reducción en la inmunidad al ruido puede provocar problemas con las conexiones de cable largos o con demasiadas cargas. Por lo tanto, se recomienda conectar no más de 10 cargas de cualquier tipo o combinación a una terminal de salida sin uso de búfer. Si se excede este factor de carga, el ruido empezará a ocasionar problemas de sincronización.

La tabla 9-3 muestra algunas de las familias lógicas más comunes y el factor de salida recomendado para el 8086/8088. La mejor opción de tipos de componentes para la conexión a una terminal de salida del 8086/8088 es un componente lógico LS, 74ALS o 74HC. Algunas de las corrientes de factor de salida se calculan para más de 10 cargas unitarias. Por lo tanto, se recomienda que, si se requiere un factor de salida de más de 10 cargas unitarias, el sistema debe utilizar búferes.

Conexiones de las terminales

- AD₇-AD₀** Las líneas del **bus de dirección/datos** del 8088 son el bus de datos/direcciones multiplexado del 8088 y contienen los ocho bits de más a la derecha de la dirección de memoria o del número de puerto de E/S siempre que la terminal ALE esté activa (1 lógico), o de datos siempre que ALE esté inactiva (0 lógico). Estas terminales se encuentran en su estado de alta impedancia durante una señal de reconocimiento del pedido de obtención del bus (HOLD).
- A₁₅-A₈** El **bus de direcciones** del 8088 proporciona los bits de la mitad superior de la dirección de memoria que se presentan durante un ciclo de bus. Estas conexiones de direcciones entran en su estado de alta impedancia durante una señal de reconocimiento del pedido de obtención del bus (HOLD).
- AD₁₅-AD₈** Las líneas del **bus de dirección/datos** del 8088 forman la parte superior del bus de datos/direcciones multiplexado en el 8086. Estas líneas contienen los bits de dirección A₁₅-A₈ siempre que ALE sea un 1 lógico, y las conexiones del bus de datos D₁₅-D₈ siempre que ALE sea un 0 lógico. Estas terminales entran en un estado de alta impedancia cuando se produce una señal de reconocimiento del pedido de obtención del bus (HOLD).
- A₁₉/S₆-A₁₆/S₃** Los bits del **bus de dirección/estado** se multiplexan para proporcionar las señales de dirección A₁₉-A₁₆ y también los bits de estado S₆-S₃. Estas terminales también obtienen un estado de alta impedancia durante la señal de reconocimiento del pedido de obtención del bus (HOLD).
- El bit de estado S₆ siempre es un 0 lógico, el bit S₅ indica la condición del bit de bandera IF, y los bits S₄ y S₃ muestran cuál segmento se utiliza durante el ciclo de bus actual. En la tabla 9-4 podrá consultar la tabla de verdad de los bits S₄ y S₃. Estos dos bits de estado podrían utilizarse para direccionar cuatro bancos de memoria separados de 1 Mbyte si se decodifican como A₂₁ y A₂₀.
- RD** Siempre que la **señal de lectura** es un 0 lógico, el bus de datos se prepara para recibir datos de la memoria o de los dispositivos de E/S conectados al sistema. Esta terminal

TABLA 9-2 Características de salida de los microprocesadores 8086 y 8088.

Nivel lógico	Voltaje	Corriente
0	0.45 V máximo	2.0 μA máximo
1	2.4 V mínimo	-400 μA máximo

TABLA 9-3 Factor de salida recomendado de cualquier conexión de terminal del 8086/8088.

Familia	Corriente de colector	Corriente de la fuente	Factor de salida
TTL (74)	-1.6 mA	40 μ A	1
TTL (74LS)	-0.4 mA	20 μ A	5
TTL (74S)	-2.0 mA	50 μ A	1
TTL (74ALS)	-0.1 mA	20 μ A	10
TTL (74AS)	-0.5 mA	25 μ A	10
TTL (74F)	-0.5 mA	25 μ A	10
CMOS (74HC)	-10 μ A	10 μ A	10
CMOS (CD)	-10 μ A	10 μ A	10
NMOS	-10 μ A	10 μ A	10

flota hacia su estado de alta impedancia durante una señal de reconocimiento del pedido de obtención del bus (HOLD).

READY	La entrada READY se controla para insertar estados de espera en la sincronización del microprocesador. Si la terminal READY se coloca a un nivel de 0 lógico, el microprocesador entra en estados de espera y permanece inactivo. Si la terminal READY se coloca en un nivel de 1 lógico, no tiene efecto sobre la operación del microprocesador.
INTR	La señal p petición de interrupción se utiliza para solicitar una interrupción de hardware. Si INTR se mantiene en nivel alto cuando IF = 1, el 8086/8088 entra en un ciclo de reconocimiento de interrupción (INTA se activa) una vez que la instrucción actual ha completado su ejecución.
TEST	La terminal Prueba es una entrada que se evalúa mediante la instrucción WAIT. Si TEST es un 0 lógico, la instrucción WAIT funciona como una instrucción NOP y si TEST es un 1 lógico, la instrucción WAIT espera a que TEST sea un 0 lógico. Esta terminal se conecta con más frecuencia al coprocesador numérico.
NMI	La entrada interrupción no enmascarable es similar a INTR, sólo que no comprueba si el bit de bandera IF es un 1 lógico. Si NMI se activa, esta entrada de interrupción utiliza el vector de interrupción 2.
RESET	La entrada reinicio hace que el microprocesador se reinicie si esta terminal se mantiene en nivel alto durante un mínimo de cuatro períodos de reloj. Cada vez que se reinicia el 8086 o el 8088, empieza a ejecutar instrucciones en la posición de memoria FFFF0H y deshabilita las interrupciones futuras al borrar el bit de bandera IF.
CLK	La terminal reloj proporciona la señal de sincronización básica para el microprocesador. La señal del reloj debe tener un ciclo de trabajo del 33% (alto durante un tercio del periodo de reloj y bajo durante dos tercios) para proporcionar una sincronización interna adecuada para el 8086/8088.
V_{CC}	Esta entrada de la fuente de poder proporciona una señal de +5.0 V, ±10% para el microprocesador.
GND	La conexión de tierra es el retorno para la fuente de alimentación. Los microprocesadores 8086/8088 tienen dos terminales etiquetadas como GND; ambas terminales deben conectarse a tierra para que el microprocesador funcione apropiadamente.

TABLA 9-4 La función de los bits de estado S₃ y S₄.

S ₄	S ₃	Función
0	0	Segmento extra
0	1	Segmento de pila
1	0	Segmento de código o ninguno
1	1	Segmento de datos

MN/MX	La terminal de modo mínimo/máximo selecciona la operación en uno de esos dos modos para el microprocesador. Si se selecciona el modo mínimo, la terminal MN/MX debe estar conectada directamente a +5.0 V.
BHE/S₇	La terminal habilita bus alto se utiliza en el 8086 para habilitar los bits más significativos del bus de datos (D ₁₅ -D ₈) durante una operación de lectura o escritura. El estado de S ₇ es siempre un 1 lógico.
Terminales de modo mínimo.	La operación del 8086/8088 en modo mínimo se obtiene mediante la conexión directa de la terminal MN/MX a +5.0 V. No conecte esta terminal a +5.0 V a través de un registro tipo “pull-up”, o no funcionará correctamente.
IO/M o M/IO	La terminal IO/M (8088) o M/IO (8086) selecciona la memoria o la E/S. Esta terminal indica que el bus de direcciones del microprocesador contiene ya sea una dirección de memoria, o la dirección de un puerto de E/S. Esta terminal se encuentra en su estado de alta impedancia durante una señal de reconocimiento del pedido de obtención del bus (HOLD).
WR	La Línea de escritura es un estrobo que indica que el 8086/8088 envía datos a una memoria o a un dispositivo de E/S. Durante el tiempo que WR es un 0 lógico, el bus de datos contiene datos válidos para la memoria o la E/S. Esta terminal flota hacia una alta impedancia durante una señal de reconocimiento del pedido de obtención del bus (HOLD).
INTA	La señal de reconocimiento de interrupción es una respuesta para la terminal de entrada INTR. La terminal INTA se utiliza por lo general como compuerta entre el número de vector de interrupción y el bus de datos, en respuesta a una petición de interrupción.
ALE	La terminal habilita el cierre de dirección que muestra que el bus de direcciones/datos del 8086/8088 contiene información relacionada con una dirección. Esta dirección puede ser de memoria o de un número de puerto de E/S. La señal ALE no flota durante una señal de reconocimiento del pedido de obtención del bus (HOLD).
DT/R	La señal recibe/transmite datos muestra que el bus de datos del microprocesador transmite (DT/R = 1) o recibe (DT/R = 0) datos. Esta señal se utiliza para habilitar los búferes externos del bus de datos.
DEN	La señal habilita bus de datos activa los búferes externos del bus de datos.
HOLD	La señal entrada de pedido de obtención del bus solicita el acceso directo a memoria (DMA). Si la señal HOLD es un 1 lógico, el microprocesador deja de ejecutar software y coloca su bus de direcciones, de datos y de control en el estado de alta impedancia. Si la señal HOLD es un 0 lógico, el microprocesador ejecuta el software de manera normal.
HLDA	La señal reconocimiento del pedido de obtención del bus (HOLD) indica que el 8086/8088 ha entrado al estado HOLD.
SS0	Esta línea de estado es equivalente a la terminal S ₀ en la operación del microprocesador en modo máximo. Esta señal se combina con IO/M y DT/R para decodificar la función del ciclo de bus actual (vea la tabla 9-5).
Terminales de modo máximo.	Para poder llegar al modo máximo, para usarlo con coprocesadores externos, conecte la terminal MN/MX a tierra.
S2, S1 y S0	Los bits de estado indican la función del ciclo de bus actual. Estas señales por lo general se decodifican mediante el controlador de bus 8288, que describiremos más adelante en este capítulo. La tabla 9-6 muestra la función de estos tres bits de estado en el modo máximo.
RQ/GT1 y RQ/GT0	Las terminales solicita/otorga solicitan accesos directos a memoria (DMA) durante la operación en modo máximo. Estas líneas son bidireccionales y se utilizan tanto para solicitar como para otorgar una operación DMA.

TABLA 9-5 Estado del ciclo de bus (8088) mediante el uso de $\overline{SS_0}$.

IO/M	DT/R	$\overline{SS_0}$	Función
0	0	0	Reconocimiento de interrupción
0	0	1	Lectura de memoria
0	1	0	Escritura de memoria
0	1	1	Alto
1	0	0	Obtención del código de operación
1	0	1	Lectura de E/S
1	1	0	Escritura de E/S
1	1	1	Pasivo

TABLA 9-6 Función del bus de control generada por el controlador de bus (8288).

S_2	S_1	S_0	Función
0	0	0	Reconocimiento de interrupción
0	0	1	Lectura de E/S
0	1	0	Escritura de E/S
0	1	1	Alto
1	0	0	Obtención del código de operación
1	0	1	Lectura de Memoria
1	1	0	Escritura de Memoria
1	1	1	Pasivo

TABLA 9-7 Bits de estado de la cola.

QS_1	QS_0	Función
0	0	La cola está inactiva
0	1	Primer byte del código de operación
1	0	La cola está vacía
1	1	Byte subsiguiente del código de operación

LOCK La salida de **bloqueo** se utiliza para bloquear periféricos del sistema. Esta terminal se activa mediante el uso del prefijo LOCK: en cualquier instrucción.

QS₁ y QS₀ Los bits de **estado de la cola** muestran el estado de la cola de instrucciones interna. Estas terminales se proporcionan para que el coprocesador numérico (8087) las utilice. En la tabla 9-7 podrá consultar la operación de estos bits de estado de la cola.

9-2

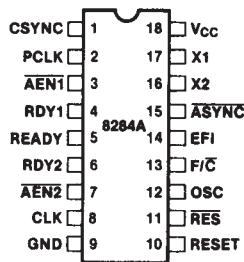
GENERADOR DE RELOJ (8284A)

En esta sección describiremos el generador de reloj (8284A) y la señal RESET; además presentaremos la señal READY para los microprocesadores 8086/8088. (En la sección 9-5 veremos con detalle la función de la señal READY y sus circuitos asociados.)

El generador de reloj 8284A

El 8284A es un componente auxiliar para los microprocesadores 8086/8088. Sin el generador de reloj, muchos circuitos adicionales tendrían que generar el reloj (CLK) en un sistema basado en el microprocesador 8086/8088. El 8284A proporciona las siguientes funciones o señales básicas: generación de reloj, sincronización RESET, sincronización READY y una señal de reloj de periférico al nivel de TTL. La figura 9-2 muestra el diagrama de terminales del generador de reloj 8284A.

FIGURA 9-2 El diagrama de terminales del generador de reloj 8284A.



Funciones de las terminales. El 8284A es un circuito integrado de 18 terminales, diseñado de manera específica para usarse con el microprocesador 8086/8088. A continuación se muestra una lista de cada terminal y su función.

AEN1 y AEN2	Las terminales habilita dirección se proporcionan para calificar las señales de bus listo: RDY ₁ y RDY ₂ , respectivamente. La sección 9-5 ilustra el uso de estas dos terminales, las cuales se utilizan para producir estados de espera, junto con las entradas RDY ₁ y RDY ₂ . Los estados de espera se generan mediante la terminal READY de los microprocesadores 8086/8088, la cual está controlada por estas dos entradas.
RDY₁ y RDY₂	Las entradas bus listo se proporcionan junto con las terminales AEN1 y AEN2 para producir estados de espera en un sistema basado en el microprocesador 8086/8088.
ASYNC	La entrada de selección sincronización READY selecciona una de dos etapas de sincronización para las entradas RDY ₁ y RDY ₂ .
READY	La señal Listo es una terminal de salida que se conecta a la entrada READY del 8086/8088. Esta señal se sincroniza con las entradas RDY ₁ y RDY ₂ .
X₁ y X₂	Las terminales oscilador de cristal se conectan con un cristal externo que se utiliza como fuente de sincronización para el generador de reloj y todas sus funciones.
F/C	La entrada de selección frecuencia/cristal elige la fuente de reloj para el 8284A. Si esta terminal se mantiene en nivel alto, se proporciona un reloj externo a la terminal de entrada EFI; si se mantiene en nivel bajo, el oscilador de cristal interno proporciona la señal de sincronización. La entrada de frecuencia externa se utiliza cuando la terminal F/C se lleva al nivel alto. EFI proporciona la sincronización siempre que la terminal F/C esté en nivel alto.
CLK	La terminal salida de reloj proporciona la señal de entrada CLK a los microprocesadores 8086/8088 y a otros componentes en el sistema. La terminal CLK tiene una señal de salida equivalente a un tercio de la frecuencia del cristal o de la entrada EFI, y tiene un ciclo de trabajo del 33%, como lo requiere el 8086/8088.
PCLK	La señal reloj de periférico equivale a una sexta parte de la frecuencia del cristal o de la entrada EFI, y tiene un ciclo de trabajo del 50%. La salida PCLK proporciona una señal de reloj al equipo periférico en el sistema.
OSC	La salida de oscilador es una señal de nivel TTL que se encuentra a la misma frecuencia que el cristal o la entrada EFI. La salida OSC proporciona una entrada EFI a otros generadores de reloj 8284A en algunos sistemas con varios procesadores.
RES	La señal entrada de reinicio es una entrada activa a baja (active-low) para el 8284A. La terminal RESET por lo general se conecta a una red RC que proporciona la capacidad de reiniciar con el equipo encendido.
RESET	La señal salida de reinicio se conecta a la terminal de entrada RESET del 8086/8088.
CSYNC	La terminal sincronización de reloj se utiliza siempre que la entrada EFI proporciona la sincronización en sistemas con varios procesadores. Si se utiliza el oscilador de cristal interno, esta terminal debe conectarse a tierra.
GND	La terminal de tierra se conecta a tierra.
V_{CC}	Esta terminal fuente de poder se conecta a +5.0 V con una tolerancia de ±10%.

Operación del 8284A

El 8284A es un componente relativamente sencillo. La figura 9-3 muestra el diagrama de sincronización interno del generador de reloj 8284A.

Operación de la sección del reloj. La mitad superior del diagrama lógico representa la sección de reloj y sincronización del generador de reloj 8284A. Como lo indica el diagrama, el oscilador de cristal tiene dos entradas: X₁ y X₂. Si se conecta un cristal a X₁ y X₂, el oscilador genera una señal de onda cuadrada a la misma frecuencia que el cristal. La señal de onda cuadrada se alimenta a una compuerta AND y también a un búfer de inversión que proporciona la señal de salida OSC. Esta señal se utiliza algunas veces como entrada EFI para otros circuitos 8284A en un sistema.

Una inspección de la compuerta AND revela que cuando F/C̄ es un 0 lógico, la salida del oscilador se conduce hacia el contador de división entre 3. Si F/C̄ es un 1 lógico, entonces EFI se conduce hacia el contador.

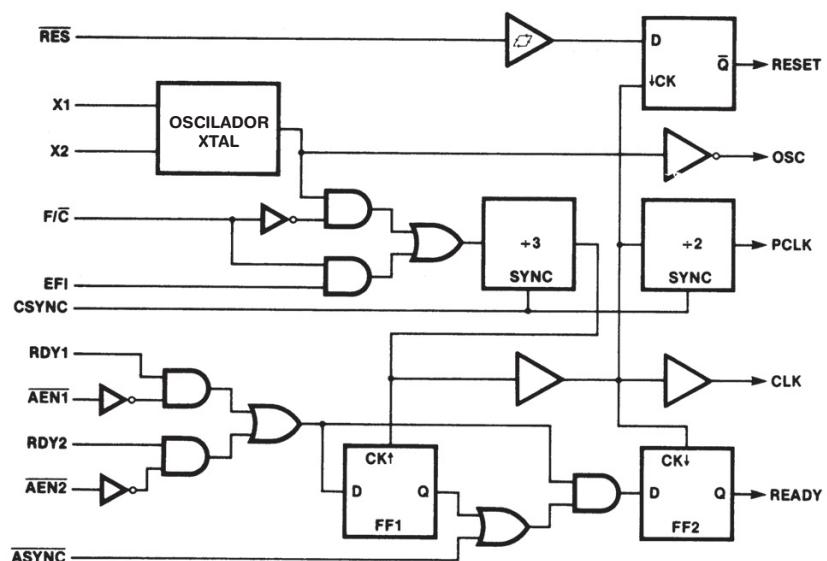
La salida del contador de división entre 3 genera la sincronización para la terminal READY, una señal para otro contador (de división entre 2) y la señal CLK para los microprocesadores 8086/8088. Se utiliza un búfer para la señal CLK antes de que salga del generador de reloj. La salida del primer contador alimenta al segundo. Estos dos contadores en cascada proporcionan la salida de división entre 6 en PCLK, la salida de reloj de periférico.

La figura 9-4 muestra cómo se conecta un 8284A al 8086/8088. Las terminales F/C̄ y CSYNC se conectan a tierra para seleccionar el oscilador de cristal, y un cristal de 15 Mhz proporciona la señal de reloj normal de 5 MHz para el 8086/8088, así como una señal de reloj de periférico de 2.5 MHz.

Operación de la sección de reinicio. Esta sección del 8284A es muy simple: consiste de un búfer de disparador de Schmitt y un solo circuito flip-flop tipo D. El flip-flop tipo D asegura que se cumplan los requerimientos de sincronización de la entrada RESET del 8086/8088. Este circuito aplica la señal RESET al microprocesador en el borde negativo (transición de 1 a 0) de cada reloj. Los microprocesadores 8086/8088 muestrean la señal RESET en el borde positivo (transición de 0 a 1) de los relojes; por lo tanto, este circuito cumple con los requerimientos de sincronización del 8086/8088.

Consulte la figura 9-4. Observe que un circuito RC proporciona un 0 lógico a la terminal de entrada RES̄ cuando se aplica energía por primera vez al sistema. Después de un periodo corto, la entrada RES̄ se convierte en 1 lógico debido a que el capacitor se carga hacia +5.0 V a través de la resistencia. Un interruptor de botón permite que el operador reinicie el microprocesador. Para una sincronización de reinicio correcta se requiere que la entrada RESET se vuelva un 1 lógico antes de cuatro ciclos de reloj, después de aplicar energía al sistema, y que se mantenga en nivel alto durante 50 μ s por lo menos. El flip-flop asegura que la señal RESET se traslade al nivel alto en cuatro ciclos de reloj, y la constante de tiempo RC asegura que permanezca en nivel alto por lo menos durante 50 μ s.

FIGURA 9-3 El diagrama de bloques interno del generador de reloj 8284A.



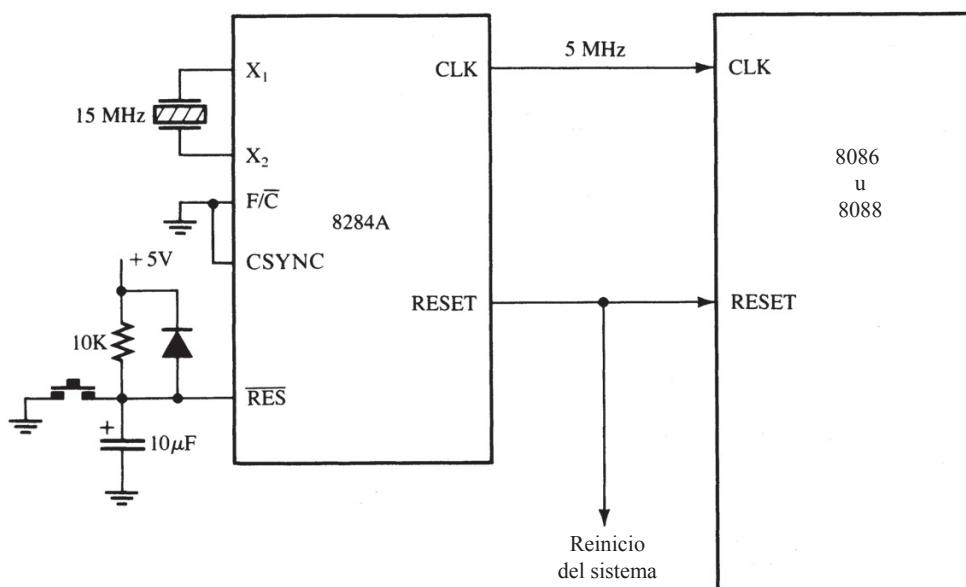


FIGURA 9-4 Diagrama del generador de reloj (8284A) y los microprocesadores 8086 y 8088, en donde se muestra la conexión para las señales de reloj y de reinicio. Un cristal de 15 MHz proporciona el reloj de 5 MHz para el microprocesador.

9-3

USO DE BÚFERES Y CIERRES EN LOS BUSES

Antes de poder utilizar los microprocesadores 8086/8088 con la memoria o las interfaces de E/S, sus buses multiplexados deben demultiplexarse. En esta sección proporcionaremos los detalles requeridos para demultiplexar los buses e ilustraremos cómo se utilizan búferes en los buses para sistemas muy grandes. (Como el factor de salida máximo es de 10, el sistema debe utilizar búferes si contiene más de 10 componentes adicionales.)

Demultiplexión de los buses

El bus de direcciones/datos en el 8086/8088 se multiplexa (comparte) para reducir el número de terminales requeridas para el circuito integrado del microprocesador 8086/8088. Desafortunadamente, esto impone en el diseñador de hardware la tarea de extraer o demultiplexar la información de estas terminales multiplexadas.

¿Por qué no dejar los buses multiplexados? La memoria y la E/S requieren que la dirección permanezca válida y estable durante un ciclo de lectura o de escritura. Si los buses se multiplexan, la dirección cambia en la memoria y la E/S, lo cual hace que lean o escriban datos en las posiciones incorrectas.

Todos los sistemas computacionales tienen tres buses: (1) un bus de direcciones que proporciona la dirección de memoria o el número de puerto de E/S para la memoria o las operaciones de E/S, respectivamente; (2) un bus de datos que transfiere datos entre el microprocesador y la memoria, o entre el microprocesador y los dispositivos de E/S en el sistema; y (3) un bus de control que proporciona señales de control para la memoria y las operaciones de E/S. Estos buses deben estar presentes para poder crear una interfaz entre la memoria y la E/S.

Demultiplexión del 8088. La figura 9-5 muestra el microprocesador 8088 y los componentes requeridos para demultiplexar sus buses. En este caso se utilizan dos cierres transparentes 74LS373 o 74LS573 para demultiplexar las conexiones AD₇-AD₀ del bus de direcciones/datos y las conexiones de dirección/estado multiplexadas A₁₉/S₁₆-A₁₆/S₃.

Estos cierres transparentes, que son como alambres siempre que la terminal de habilitación de cierre de dirección (ALE) se vuelve un 1 lógico, pasan las entradas a las salidas. Después de un periodo corto la terminal ALE regresa a su condición de 0 lógico, lo cual hace que los cierres recuerden las

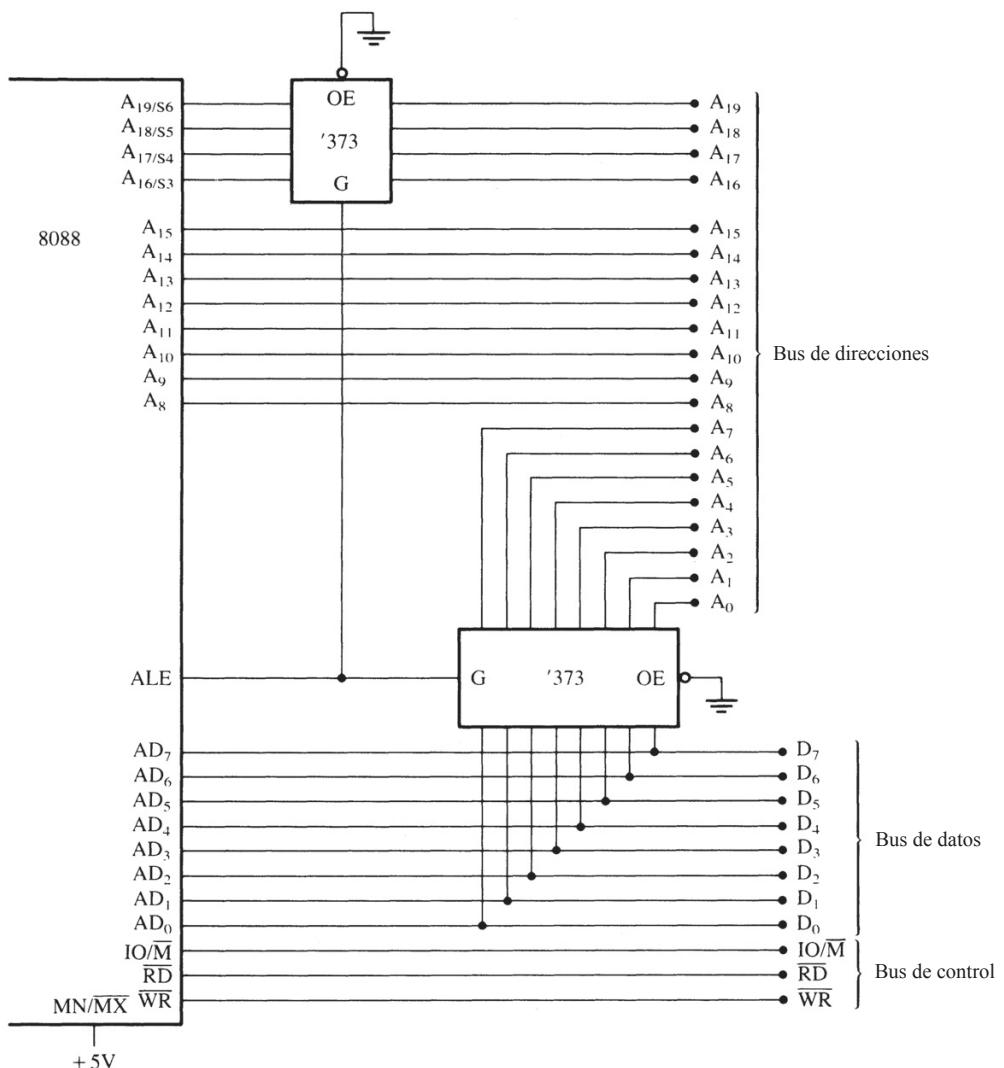


FIGURA 9-5 El microprocesador 8088 se muestra con un bus de dirección demultiplexado. Éste es el modelo que se utiliza para construir muchos sistemas basados en el 8088.

entradas al momento del cambio para un 0 lógico. En este caso, las conexiones A₇-A₀ se almacenan en el cierre inferior y las conexiones A₁₉-A₁₆ se almacenan en el cierre superior.

Esto produce un bus de direcciones separado con las conexiones A₁₉-A₀. Estas conexiones de dirección permiten al 8088 direccionar 1 Mbyte de espacio de memoria. El hecho de que el bus de datos esté separado le permite conectarse con cualquier dispositivo periférico o componente de memoria de 8 bits.

Demultiplexión del 8086. Al igual que el 8088, el 8086 requiere buses de control de direcciones, datos y control separados. Su principal diferencia es en el número de terminales multiplexadas. En el 8088, sólo AD₇-AD₀ y A₁₉/S₆-A₁₆/S₃ se multiplexan. En el 8086, las terminales que se multiplexan son AD₁₅-AD₀, A₁₉/S₆-A₁₆/S₃ y BHE/S₇. Todas estas señales deben demultiplexarse.

La figura 9-6 muestra un 8086 demultiplexado con los tres buses: de direcciones (A₁₉-A₀, y BHE), de datos (D₁₅-D₀) y de control (M/IO, RD y WR).

Este circuito que se muestra en la figura 9-6 es casi idéntico al que se muestra en la figura 9-5, con la excepción de que se ha agregado un cierre adicional al 74LS373 superior para demultiplexar las terminales AD₁₅-AD₈ del bus de direcciones/datos y se ha agregado una entrada BHE/S₇ al 74LS373 superior.

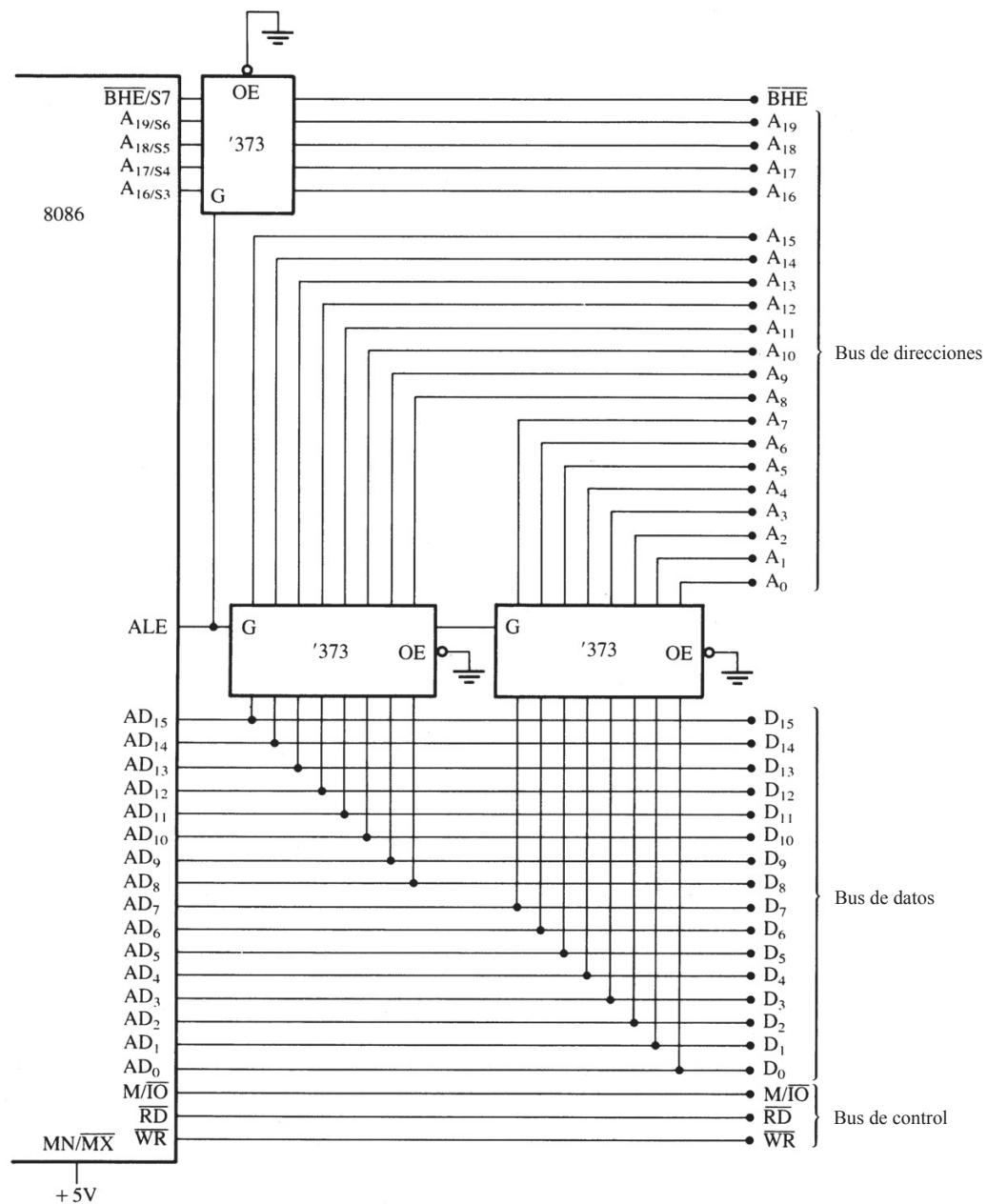


FIGURA 9-6 El microprocesador 8086 se muestra con un bus de direcciones demultiplexado. Éste es el modelo que se utiliza para construir muchos sistemas basados en el 8086.

para seleccionar el banco de memoria de alto orden en el sistema de memoria de 16 bits del 8086. Aquí, la memoria y el sistema de E/S ven al 8086 como un dispositivo con un bus de direcciones de 20 bits (A₁₉-A₀), un bus de datos de 16 bits (D₁₅-D₀) y un bus de control de tres líneas (M/I/O, RD y WR).

El sistema de búferes

Si se conectan más de 10 cargas unitarias a la terminal de cualquier bus, todo el sistema 8086 u 8088 debe utilizar búferes. Las terminales demultiplexadas ya utilizan búferes mediante los cierres 74LS373 o 74LS573, los cuales se han diseñado para controlar los buses de alta capacitancia que se encuentran

en los sistemas de microcomputadoras. Las corrientes de salida del búfer se han incrementado de manera que puedan controlarse más cargas unitarias TTL. Una salida de 0 lógico proporciona hasta 32 mA de corriente de colector, y una salida de 1 lógico proporciona hasta 5.2 mA de corriente de colector.

Una señal con uso completo de búferes introducirá un retraso de sincronización en el sistema. Esto no produce dificultad a menos que se utilice la memoria o los dispositivos de E/S, que funcionan casi a la velocidad máxima del bus. En la sección 9-4 hablaremos con más detalle sobre este problema y los retrasos de tiempo implicados.

El 8088 con uso completo de búferes. En la figura 9-7 se muestra un microprocesador 8088 con uso completo de búferes. Observe que las ocho terminales de dirección restantes ($A_{15}-A_8$) utilizan un búfer octal 74LS244; los ocho pines del bus de datos (D_7-D_0) utilizan un búfer de bus octal bidireccional

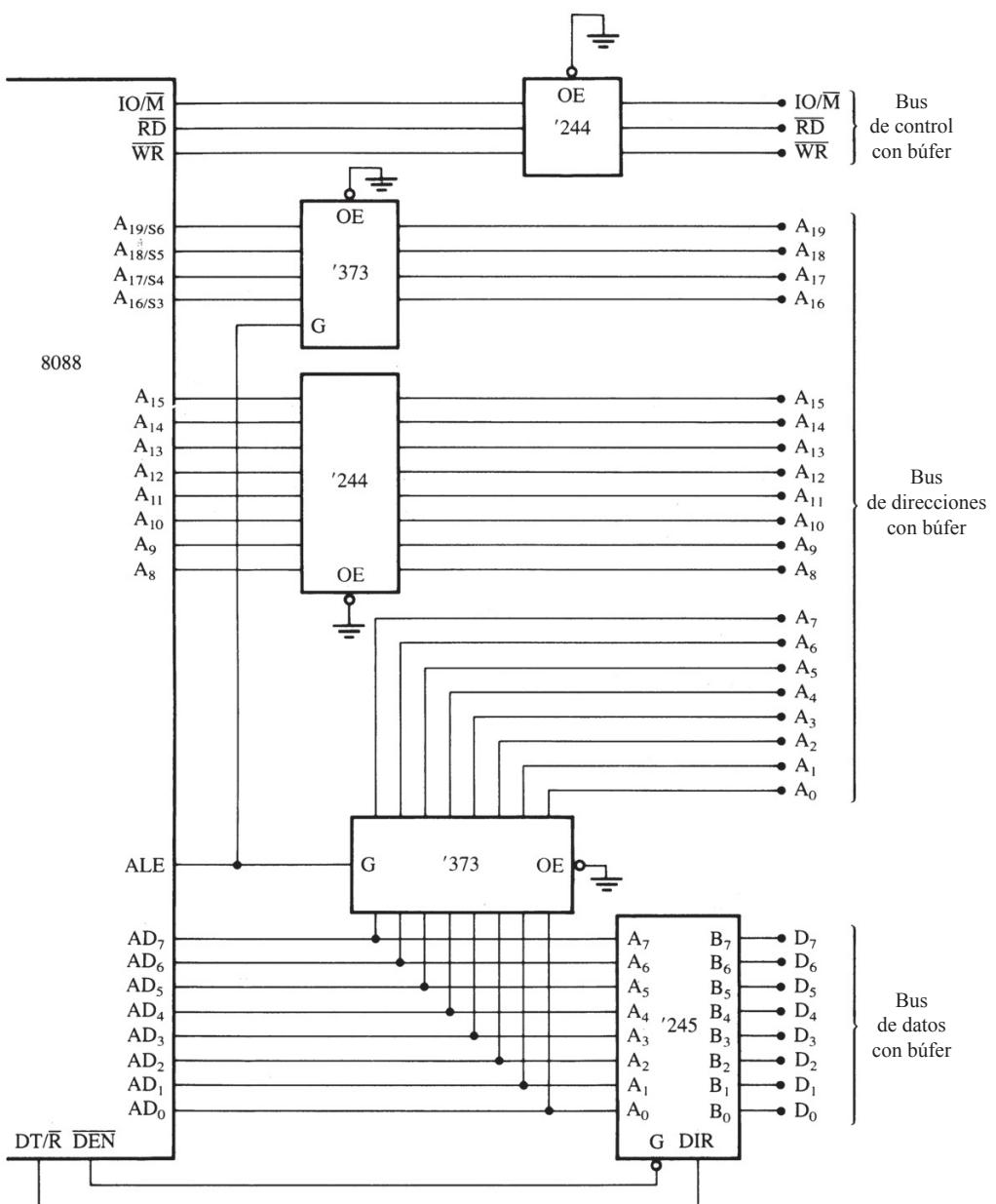


FIGURA 9-7 Un microprocesador 8088 con uso completo de búferes.

74LS245; y las señales del bus de control (M/\overline{IO} , \overline{RD} y \overline{WR}) utilizan un búfer 74LS244. Un sistema 8088 con uso completo de búferes requiere dos 74LS244, un 74LS245 y dos 74LS373. La dirección del 74LS245 se controla mediante la señal DT/R y se habilita y deshabilita mediante la señal DEN .

El 8086 con uso completo de búferes. La figura 9-8 muestra un microprocesador 8086 con uso completo de búferes. Sus terminales de dirección ya utilizan búfer mediante los cierres de dirección 74LS373; su bus de datos emplea dos búferes de bus octales bidireccionales; y las señales del bus de

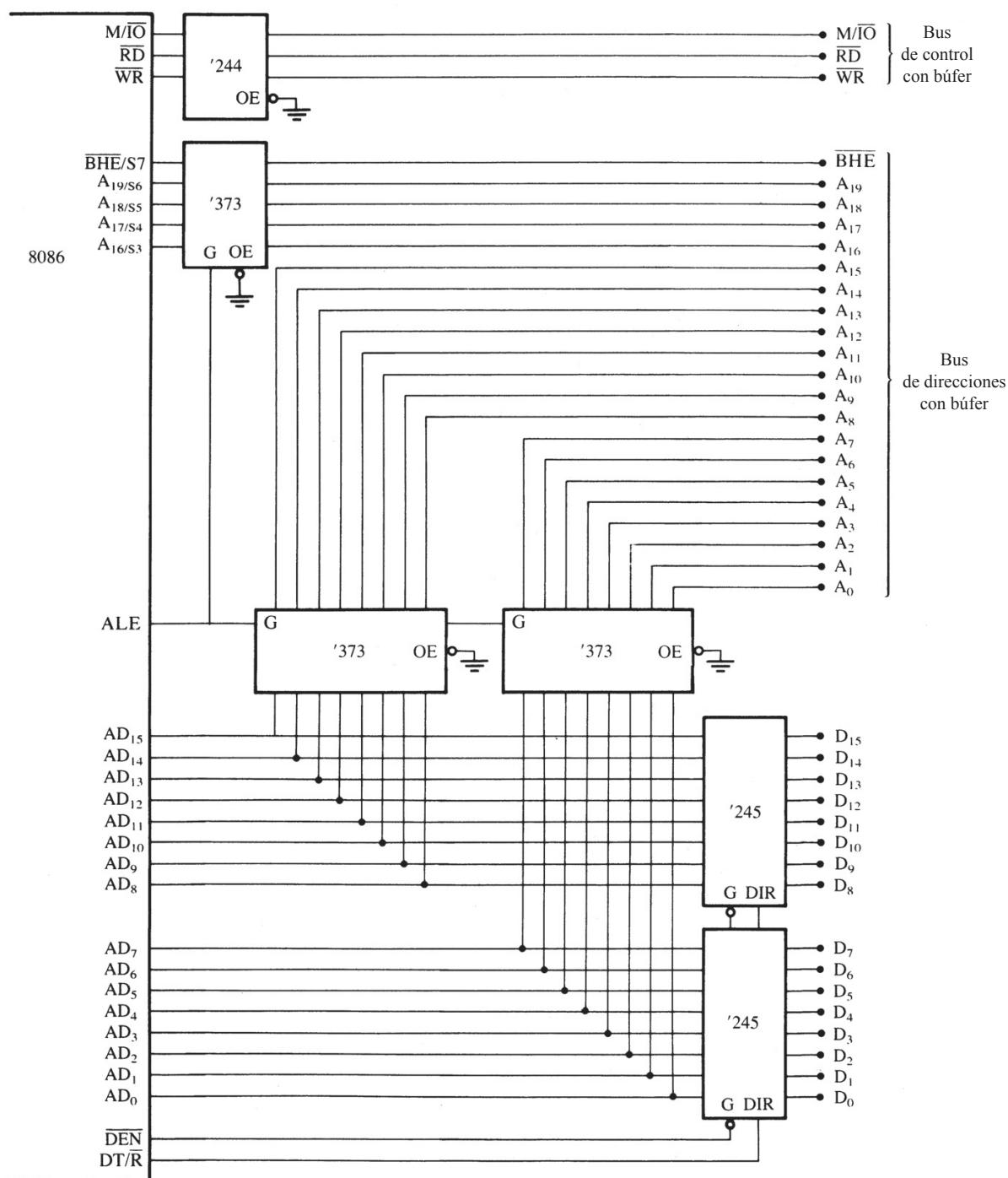


FIGURA 9-8 Un microprocesador 8086 con uso completo de búferes.

control (M/\overline{IO} , \overline{RD} y \overline{WR}) utilizan un búfer 74LS244. Un sistema 8086 con uso completo de búferes requiere un 74LS244, dos 74LS245 y tres 74LS373. El 8086 requiere un búfer más que el 8088 debido a las conexiones extra del bus de datos ($D_{15}-D_8$). También tiene una señal BHE que utiliza búferes para la selección del banco de memoria.

9-4

SINCRONIZACIÓN DEL BUS

Es esencial comprender el funcionamiento de la sincronización del bus del sistema antes de elegir una memoria o un dispositivo de E/S para comunicarse con los microprocesadores 8086 u 8088. En esta sección proporcionaremos la información sobre la operación de las señales del bus y la sincronización básica de lectura y escritura del 8086/8088. Es importante observar que en esta sección sólo hablaremos sobre los tiempos que afectan a la interfaz con la memoria y la E/S.

Operación básica del bus

Los tres buses del 8086 y del 8088 (de direcciones, de datos y de control) funcionan de igual forma que los de cualquier otro microprocesador. Si se escriben datos en la memoria (vea la sincronización simplificada para la escritura en la figura 9-9), el microprocesador envía la dirección de memoria en el bus de direcciones, envía en el bus de datos la información que se va a escribir en la memoria y emite una señal de escritura (WR) a memoria y la señal $IO/\overline{M} = 0$ para el 8088, o $M/\overline{IO} = 1$ para el 8086. Si los datos se leen de la memoria (vea la sincronización simplificada para la lectura en la figura 9-10), el microprocesador envía la dirección de memoria en el bus de direcciones, emite una señal de lectura de memoria (\overline{RD}) y acepta la información a través del bus de datos.

Sincronización en general

Los microprocesadores 8086/8088 utilizan la memoria y la E/S en períodos llamados **ciclos de bus**. Cada ciclo de bus equivale a cuatro períodos de reloj del sistema (estados T). Los microprocesadores más recientes dividen el ciclo de bus en dos períodos de reloj. Si el reloj se opera a 5 MHz (la frecuencia básica de operación para estos dos microprocesadores), un ciclo de bus del 8086/8088 se completa en 800 ns. Esto significa que el microprocesador lee o escribe datos entre sí mismo y la memoria o la E/S a una velocidad máxima de 1.25 millones de veces por segundo. (Debido a la cola interna, el 8086/8088 puede ejecutar 2.5 millones de instrucciones por segundo [MIPS] en ráfagas.) Otras versiones disponibles de estos microprocesadores operan a velocidades de transferencia mucho mayores debido a que utilizan frecuencias de reloj más altas.

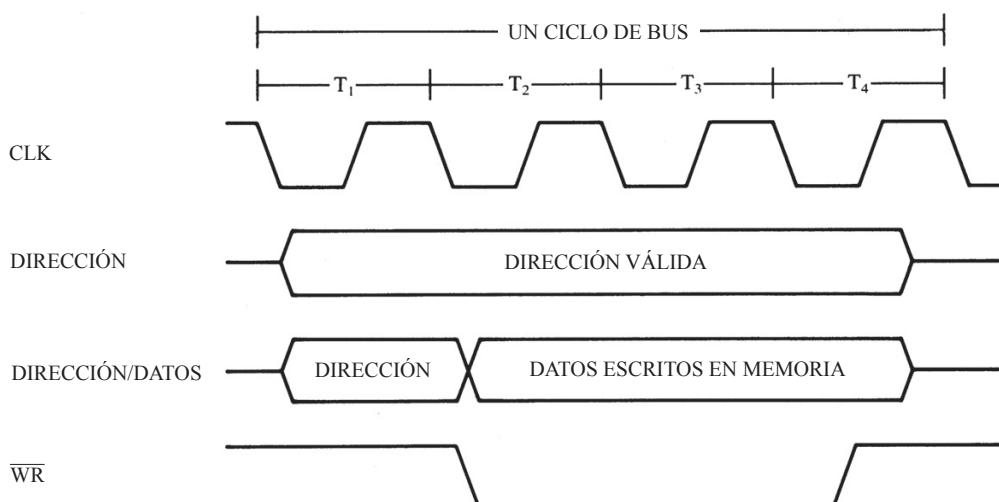


FIGURA 9-9 El ciclo de bus de escritura del 8086/8088 simplificado.

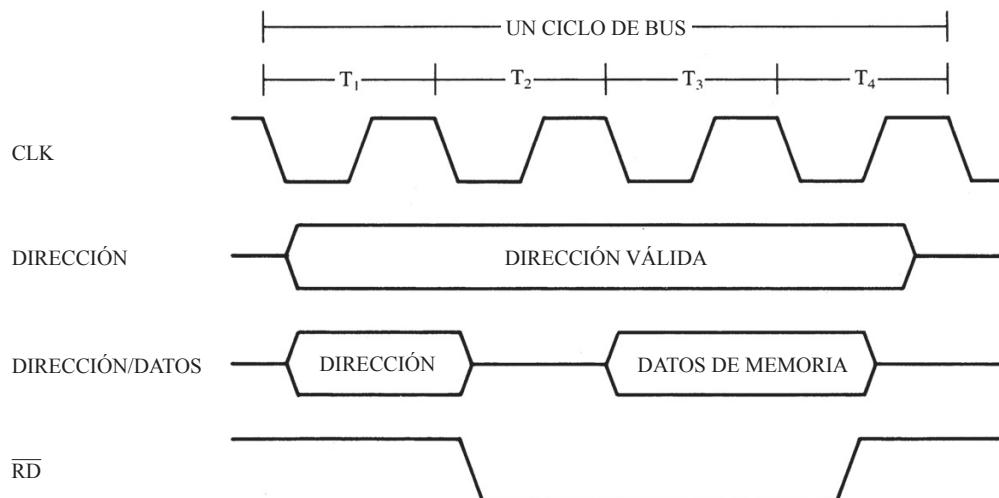


FIGURA 9-10 El ciclo de bus de lectura del 8086/8088 simplificado.

Durante el primer periodo de reloj en un ciclo de bus, el cual se conoce como T₁, ocurren muchas cosas. La dirección de la memoria o la ubicación de la E/S se envían a través de las conexiones del bus de direcciones y del bus de direcciones/datos. (El bus de direcciones/datos está multiplexado y algunas veces contiene información de direccionamiento de memoria, otras veces contiene datos.) Durante T₁ también se envían las señales de control ALE, DT/R y IO/M (8088) o M/I/O (8086). La señal IO/M o M/I/O indica si el bus de direcciones contiene una dirección de memoria o un número de dispositivo (puerto) de E/S.

Durante T₂, los microprocesadores 8086/8088 emiten la señal RD o WR, DEN y en caso de una escritura, los datos a escribir aparecen en el bus de datos. Estos eventos hacen que la memoria o el dispositivo de E/S empiece a realizar una lectura o una escritura. La señal DEN activa los búferes del bus de datos (en caso de que estén presentes en el sistema), de manera que la memoria o el dispositivo de E/S pueda recibir los datos a escribir, o para que el microprocesador pueda aceptar los datos leídos de la memoria o del dispositivo de E/S para una operación de lectura. Si el ciclo de bus es de escritura, los datos se envían a la memoria o al dispositivo de E/S a través del bus de datos.

La señal READY se muestrea al final de T₂, como se muestra en la figura 9-11. Si READY está en nivel bajo en este momento, T₃ se convierte en un estado de espera (T_W). (En la sección 9-5 proporcionaremos más detalles.) Este periodo de reloj se proporciona para dar tiempo a la memoria para que acceda a los datos. Si el ciclo de bus es de lectura, el bus de datos se muestrea al final de T₃.

En T₄, todas las señales del bus se desactivan para prepararse para el siguiente ciclo de bus. Éste es también el momento en el que el 8086/8088 muestrea las conexiones del bus de datos para ver si hay datos que se lean de memoria de un dispositivo de E/S. Además, en este punto el borde de caída de la señal WR transfiere datos a la memoria o al dispositivo de E/S, que se activa y realiza la escritura cuando la señal WR regresa a un nivel de 1 lógico.

Sincronización de lectura

La figura 9-11 también describe la sincronización de lectura para el microprocesador 8088. La sincronización de lectura del 8086 es idéntica, sólo que el 8086 tiene 16 bits en el bus de datos, en vez de ocho. Si analiza con cuidado este diagrama de sincronización podrá identificar todos los eventos principales que se describen para cada estado T.

El elemento más importante que se incluye en el diagrama de sincronización de lectura es la cantidad de tiempo permitido para que la memoria o el dispositivo de E/S lean los datos. La memoria se selecciona en base a su tiempo de acceso, que es la cantidad fija de tiempo que el microprocesador le permite para acceder a los datos para la operación de lectura. Por lo tanto, es en extremo importante que la memoria seleccionada cumpla con las limitaciones del sistema.

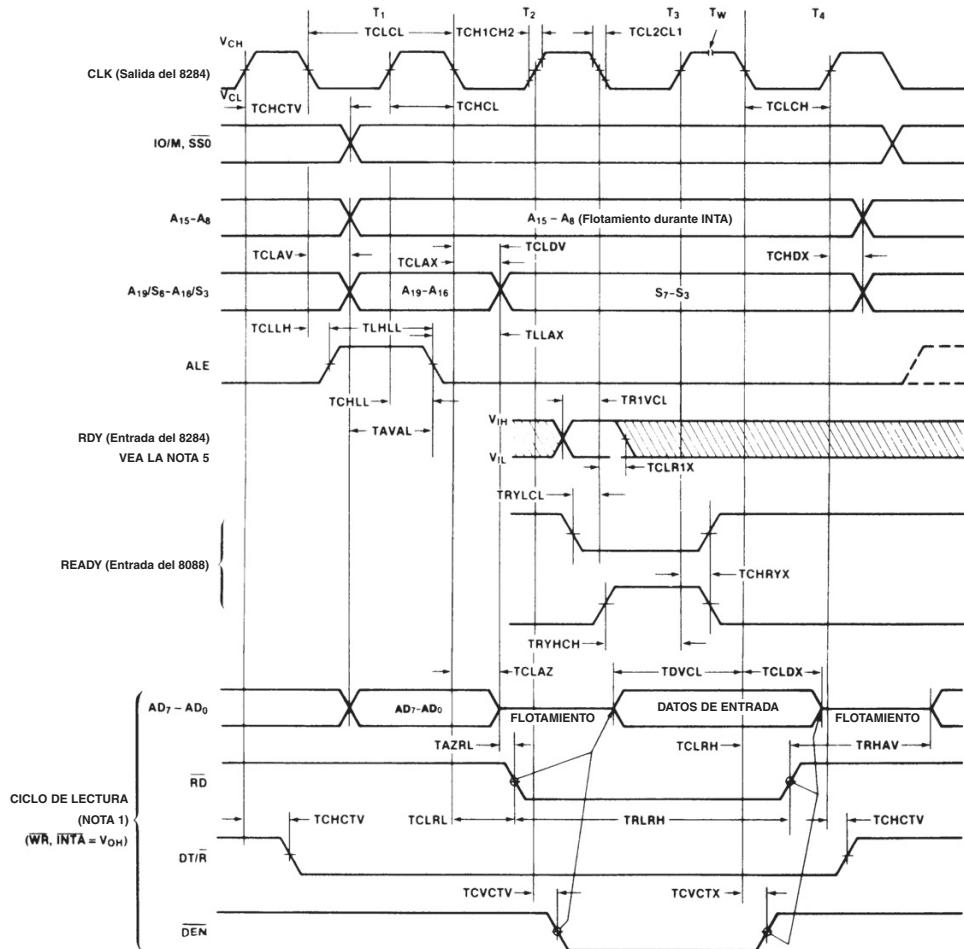


FIGURA 9-11 La sincronización del bus del 8088 en modo mínimo para una operación de lectura.

El diagrama de sincronización del microprocesador no proporciona un listado para el tiempo de acceso a la memoria. En vez de ello, es necesario combinar varios tiempos para llegar al tiempo de acceso. Para encontrar el tiempo de acceso a la memoria en este diagrama, primero localice el punto en T₃ en el que se muestran los datos. Si examina con cuidado el diagrama de sincronización, observará una línea que se extiende desde el extremo de T₃ hacia el bus de datos. Al final de T₃, el microprocesador muestrea el bus de datos.

El tiempo de acceso a la memoria empieza cuando la dirección aparece en el bus de direcciones de memoria y continúa hasta que el microprocesador muestrea los datos de memoria en T₃. Entre estos tiempos transcurren aproximadamente tres estados T. (En la figura 9-12 podrá ver los siguientes tiempos.) La dirección no aparece sino hasta que el tiempo T_{CLAV} debe restarse de los tres estados de reloj (600 ns) que separan la aparición de la dirección (T₁) y el muestreo de los datos (T₃). También debe restarse otro tiempo: el tiempo de preparación de los datos (T_{DVCL}), el cual ocurre antes de T₃. El tiempo de acceso a la memoria es, por lo tanto, tres estados de reloj menos la suma de T_{CLAV} y T_{DVCL}. Como T_{DVCL} es de 30 ns con un reloj de 5 MHz, el tiempo de acceso a memoria permitido es de sólo 460 ns (tiempo de acceso = 600 ns - 110 ns - 30 ns).

Los dispositivos de memoria que se seleccionen para conectarse al 8086/8088 que opera a 5 MHz deben poder acceder a los datos en menos de 460 ns, debido al retraso de tiempo que se introduce por los decodificadores de direcciones y los búferes en el sistema. Debe existir por lo menos un margen de entre 30 y 40 ns para la operación de estos circuitos. Por ende, la velocidad de memoria no debe ser menor de 420 ns para que pueda operar de forma correcta con los microprocesadores 8086/8088.

FIGURA 9-12
Características CA
del 8088.

CARACTERISTICAS DE A.C. (8088: $T_A = 0^\circ\text{C}$ a 70°C , $V_{CC} = 5 \text{ V} \pm 10\%$)*

(8088-2: $T_A = 0^\circ\text{C}$ a 70°C , $V_{CC} = 5 \text{ V} \pm 5\%$)*

REQUERIMIENTOS DE SINCRONIZACIÓN DEL SISTEMA DE MÍNIMA COMPLEJIDAD

Símbolo	Parámetro	8088		8088-2		Unidades	Condiciones de prueba
		Mín.	Máx.	Mín.	Máx.		
TCLCL	Periodo de ciclo CLK	200	500	125	500	ns	
TCLCH	Tiempo bajo CLK	118		68		ns	
TCHCL	Tiempo alto CLK	69		44		ns	
TCH1CH2	Tiempo elevación CLK		10		10	ns	De 1.0 V a 3.5 V
TCL2CL1	Tiempo caída CLK		10		10	ns	De 3.5 V a 1.0 V
TDVCL	Datos en tiempo de preparación	30		20		ns	
TCLDX	Datos en tiempo de espera (hold)	10		10		ns	
TR1VCL	Tiempo de preparación RDY hacia 8284 (vea notas 1, 2)	35		35		ns	
TCLR1X	Tiempo de espera (hold) RDY hacia 8284 (vea notas 1, 2)	0		0		ns	
TRYHCH	Tiempo de preparación READY hacia 8088	118		68		ns	
TOHRYX	Tiempo de espera (hold) READY hacia 8088	30		20		ns	
TRYLCL	READY inactiva para CLK (vea nota 3)	-8		-8		ns	
THVCH	Tiempo de preparación HOLD	35		20		ns	
TINVCH	Tiempo de preparación INTR, NMI, TEST (vea nota 2)	30		15		ns	
TILIH	Tiempo de elevación de entrada (excepto CLK)		20		20	ns	De 0.8 V a 2.0 V
TIHIL	Tiempo de caída de entrada (excepto CLK)		12		12	ns	De 2.0 V a 0.8 V

CARACTERISTICAS DE A.C. (continuación)

RESPUESTAS DE SINCRONIZACIÓN

Símbolo	Parámetro	8088		8088-2		Unidades	Condiciones de prueba
		Mín.	Máx.	Mín.	Máx.		
TCLAV	Retraso válido de dirección	10	110	10	60	ns	
TCLAX	Tiempo de espera (hold) de dirección	10		10		ns	
TCLAZ	Retraso de flotamiento de dirección	TCLAX	80	TCLAX	50	ns	
TLHLL	Anchura ALE	TCLCH-20		TCLCH-10		ns	
TCLLH	Retraso activo ALE		80		50	ns	
TCHLL	Retraso inactivo ALE		85		55	ns	
TLLAX	Tiempo de espera (hold) de dirección para ALE inactiva	TCHCL-10		TCHCL-10		ns	
TCLDV	Retraso válido de datos	10	110	10	60	ns	
TCHDX	Tiempo de espera (hold) de datos	10		10		ns	
TWHDX	Tiempo de espera (hold) de datos después de WR	TCLCH-30		TCLCH-30		ns	
TCVCTV	Retraso activo de control 1	10	110	10	70	ns	
TCHCTV	Retraso activo de control 2	10	110	10	60	ns	
TCVCTX	Retraso inactivo de control	10	110	10	70	ns	
TAZRL	Flotamiento de dirección para READ activo	0		0		ns	
TCLRL	Retraso activo de RD	10	165	10	100	ns	
TCLRH	Retraso inactivo de RD	10	150	10	80	ns	
TRHAV	RD inactiva para siguiente dirección activa	TCLCL-45		TCLCL-40		ns	
TCLHAV	Retraso válido de HLDA	10	160	10	100	ns	
TRLRH	Anchura de RD	2TCLCL-75		2TCLCL-50		ns	
TWLWH	Anchura de WR	2TCLCL-60		2TCLCL-40		ns	
TAVAL	Dirección válida para ALE baja	TCLCH-60		TCLCH-40		ns	
TOLOH	Tiempo de elevación de salida		20		20	ns	De 0.8 V a 2.0 V
TOHOL	Tiempo de caída de salida		12		12	ns	De 2.0 V a 0.8 V

$C_L=20-100\text{pF}$ para todas las salidas del 8088 además de las cargas internas

El otro factor de sincronización que puede afectar a la operación de la memoria es la anchura del estrobo \overline{RD} . En el diagrama de sincronización, el estrobo de lectura se proporciona como T_{RLRH} . El tiempo para este estrobo es de 325 ns (a una velocidad de reloj de 5 MHz), que es lo suficientemente amplio como para casi todos los dispositivos de memoria fabricados con un tiempo de acceso de 400 ns o menos.

Sincronización de escritura

La figura 9-13 muestra el diagrama de sincronización de escritura para el microprocesador 8088. (De nuevo, el 8086 es casi idéntico, por lo que no necesitamos presentarlo en un diagrama de sincronización separado.)

Las principales diferencias entre la sincronización de lectura y la de escritura son mínimas. El estrobo RD se sustituye por el estrobo WR, el bus de datos contiene información para la memoria en vez de contener la información que viene de la memoria, y DT/R permanece en 1 lógico en vez de 0 lógico durante el ciclo de bus.

Al crear interfaces con algunos dispositivos de memoria, la sincronización puede ser muy imprescindible entre el punto en el que WR se convierte en un 1 lógico y el tiempo en donde los datos se quitan del bus de datos. Éste es el caso ya que, como recordará, los datos de memoria se escriben durante el borde de caída del estrobo WR. De acuerdo con el diagrama de sincronización, este periodo crítico es T_{WHDX} o 88 ns cuando el 8088 se opera con un reloj de 5 MHz. El tiempo de espera (hold) es mucho menor que esto; de hecho, por lo general es de 0 ns para los dispositivos de memoria. La anchura del estrobo WR es T_{WLWH} o 340 ns a una velocidad de reloj de 5 MHz. Esta velocidad es compatible con la mayoría de los dispositivos de memoria que tienen un tiempo de acceso de 400 ns o menos.

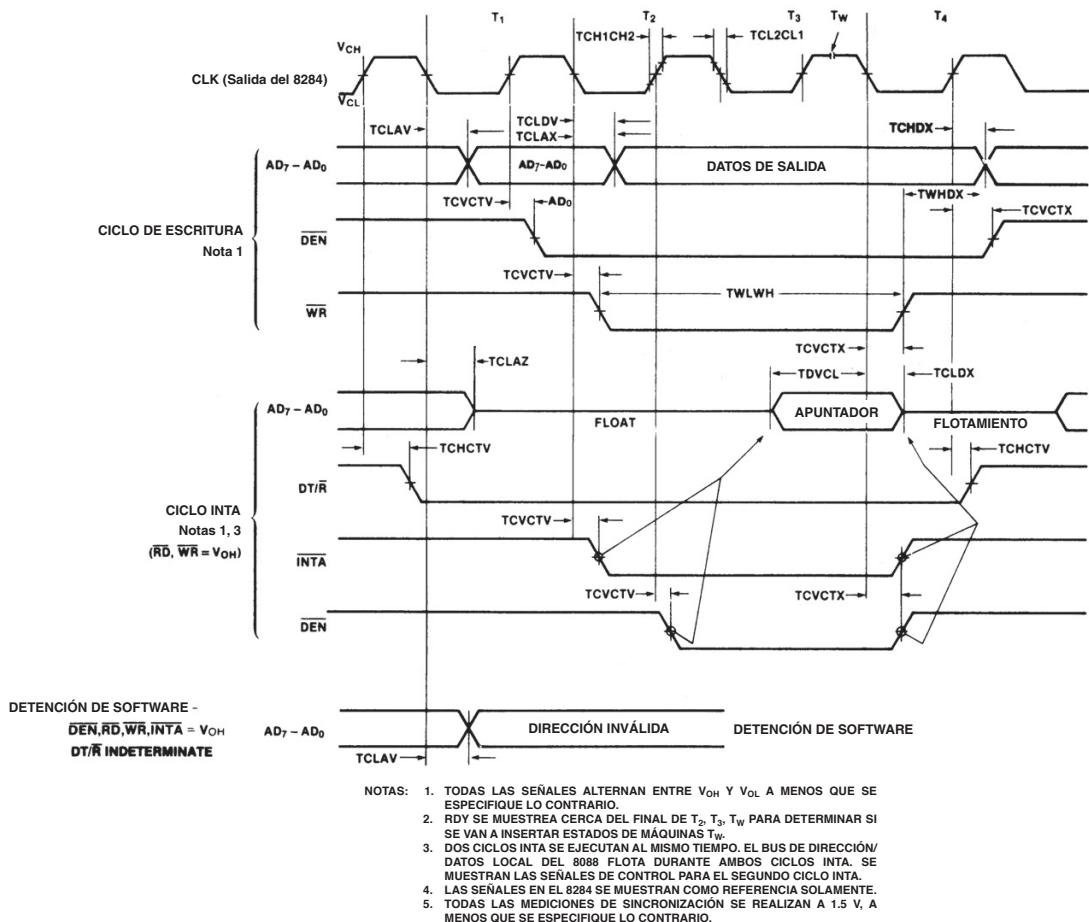


FIGURA 9-13 Sincronización de escritura en el bus del 8088 en modo mínimo.

9-5**READY Y EL ESTADO DE ESPERA**

Como lo mencionamos en secciones anteriores, la entrada READY produce estados de espera para los componentes más lentos de memoria y de E/S. Un estado de espera (T_w) es un periodo de reloj adicional que se inserta entre T_2 y T_3 para alargar el ciclo de bus. Si se inserta un estado de espera, entonces el tiempo de acceso a la memoria (por lo general de 460 ns con un reloj de 5 MHz) se extiende un periodo de reloj (200 ns) hasta 660 ns.

En esta sección hablaremos sobre los circuitos de sincronización de la señal READY dentro del generador de reloj 8284A, mostraremos cómo insertar uno o más estados de espera de manera selectiva en el ciclo de bus y examinaremos la entrada READY así como los tiempos de sincronización que requiere.

La entrada READY

Esta entrada se muestrea al final de T_2 y (si se aplica) se vuelve a muestrear a mitad de T_w . Si READY es un 0 lógico al final de T_2 , se retrasa T_3 y se inserta T_w entre T_2 y T_3 . A continuación, READY se muestrea a mitad de T_w para determinar si el siguiente estado va a ser T_w o T_3 . Se evalúa para ver si es un 0 lógico en la transición de 1 a 0 del reloj al final de T_2 , y se evalúa para ver si es un 1 lógico en la transición de 0 a 1 del reloj a mitad de T_w .

La entrada READY para el 8086/8088 tiene ciertos requerimientos estrictos de sincronización. El diagrama de sincronización de la figura 9-14 muestra cómo READY provoca un estado de espera (T_w), junto con los tiempos requeridos de preparación y de espera (hold) del reloj del sistema. El requerimiento de sincronización para esta operación se cumple mediante los circuitos de sincronización internos para READY del generador de reloj 8284A. Cuando el 8284A se utiliza para READY, la entrada RDY (entrada READY para el 8284A) ocurre al final de cada estado T.

RDY y el 8284A

RDY es la entrada READY sincronizada para el generador de reloj 8284A. En la figura 9-15 se muestra el diagrama de sincronización para esta entrada. Aunque difiere de la sincronización para la entrada

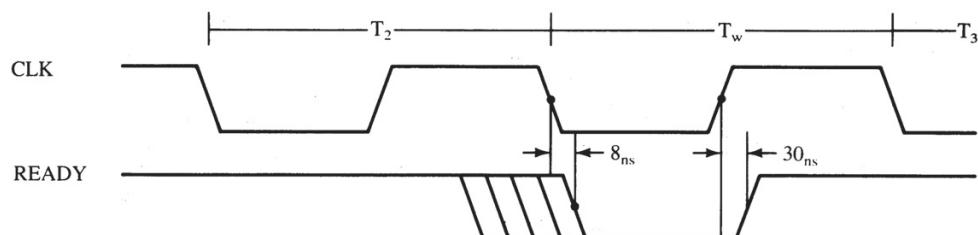


FIGURA 9-14 Sincronización de la entrada READY del 8086/8088.

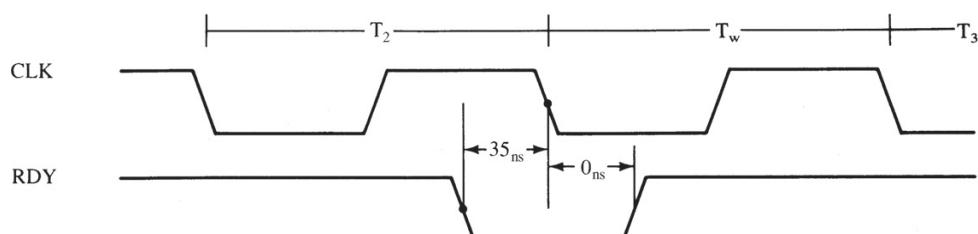
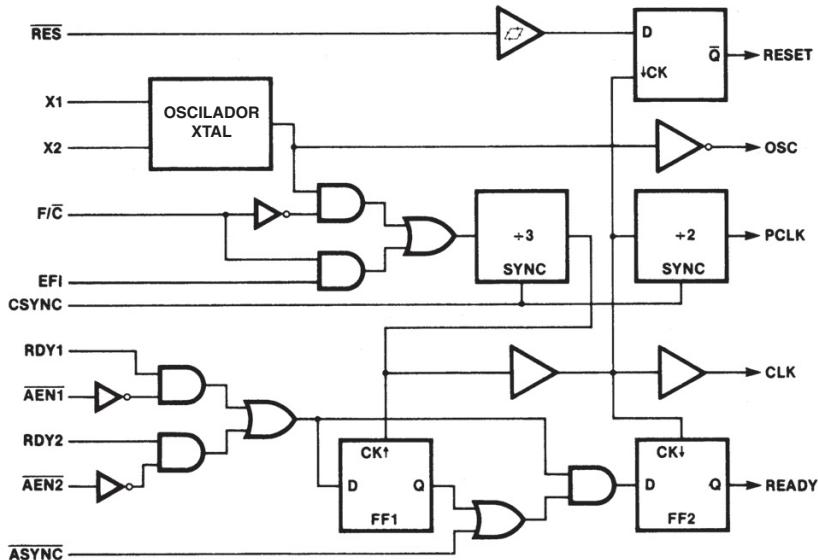


FIGURA 9-15 Sincronización de la entrada RDY del 8284A.

FIGURA 9-16 Diagrama de bloques internos del generador de reloj 8284A. (Cortesía de Intel Corporation.)



READY que va al 8086/8088, los circuitos internos del 8284A garantizan la precisión de la sincronización de READY que se proporciona a los microprocesadores 8086/8088.

La figura 9-16 muestra de nuevo la estructura interna del 8284A. La mitad inferior de este diagrama corresponde al circuito de sincronización de READY. En el extremo izquierdo se aplica un AND a las entradas RDY₁ y AEN₁, al igual que con las entradas RDY₂ y AEN₂. Despues se aplica un OR a las salidas de las compuertas AND para generar la entrada a una o dos etapas de sincronización. Para obtener un 1 lógico en las entradas a los flip-flops, la salida del AND entre RDY₁ y AEN₁ debe estar activa, o la salida del AND entre RDY₂ y AEN₂ debe estar activa.

La entrada ASYNC selecciona una etapa de sincronización cuando es un 1 lógico y dos etapas cuando es un 0 lógico. Si se selecciona una etapa, entonces se impide que la señal RDY llegue a la terminal READY del 8086/8088 hasta el siguiente borde negativo del reloj. Si se seleccionan dos etapas, el primer borde positivo del reloj captura RDY en el primer flip-flop. La salida de este flip-flop se alimenta al segundo flip-flop, por lo que en el siguiente borde negativo del reloj el segundo flip-flop captura RDY.

La figura 9-17 muestra un circuito utilizado para introducir casi cualquier número de estados de espera para los microprocesadores 8086/8088. Aquí, un registro de desplazamiento en serie de ocho bits (74LS164) desplaza un 0 lógico para uno o más períodos de reloj, desde una de sus salidas Q a través de la entrada RDY₁ del 8284A. Con el puente apropiado, este circuito puede proporcionar varios números de estados de espera. Observe también cómo el registro de desplazamiento se regresa a su punto de inicio. La salida del registro se fuerza al nivel alto cuando las terminales RD, WR e INTA son todas 1s lógicos. Estas tres señales se quedan en nivel alto hasta el estado T₂. Por lo que el registro de desplazamiento se mueve por primera vez cuando llega el borde positivo de T₂. Si se desea un estado de espera, la salida Q_B se conecta a la compuerta OR. Si se desean dos estados de espera se conecta la salida Q_C, y así sucesivamente.

En la figura 9-17 podrá ver que este circuito no siempre genera estados de espera. Se habilita desde la memoria sólo para dispositivos de memoria que requieren la inserción de estados de espera. Si la señal de selección de un dispositivo de memoria es un 0 lógico, se selecciona el dispositivo; entonces este circuito generará un estado de espera.

La figura 9-18 muestra el diagrama de sincronización para este generador de estados de espera del registro de desplazamiento cuando se cablea para insertar un estado de espera. El diagrama de sincronización también ilustra el contenido interno de los flip-flops del registro de desplazamiento, para presentar una vista más detallada de su operación. En este ejemplo se genera un estado de espera.

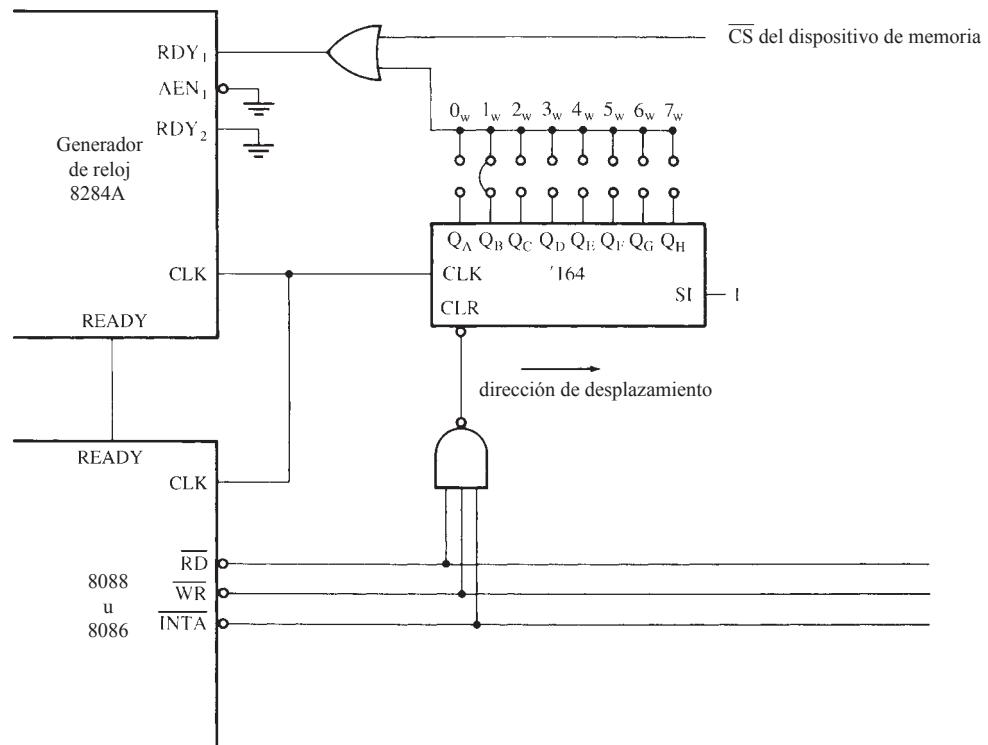


FIGURA 9-17 Un circuito que produce entre 0 y 7 estados de espera.

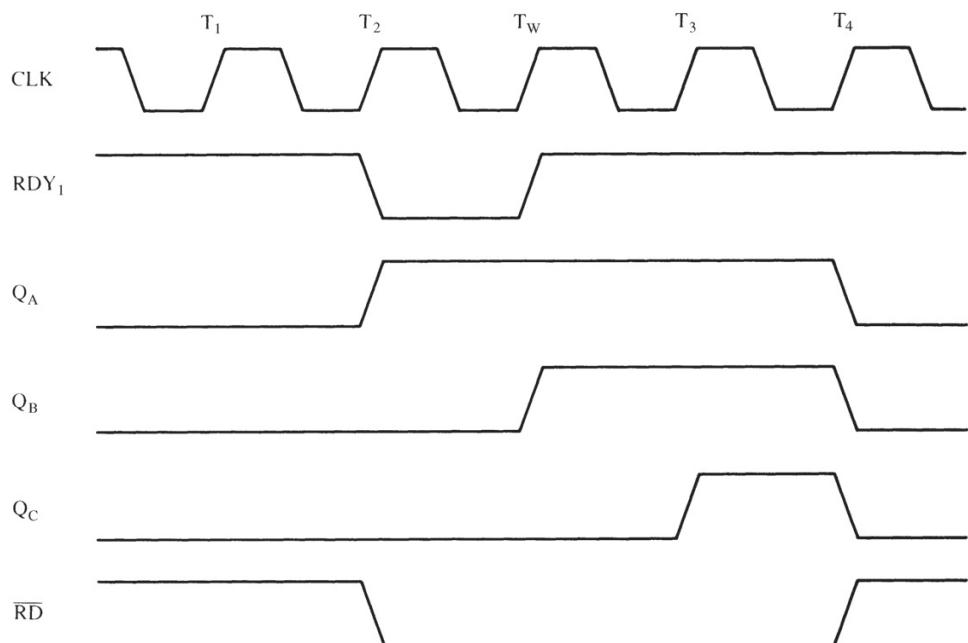


FIGURA 9-18 Sincronización de la generación de estados de espera del circuito de la figura 9-17.

9-6

COMPARACIÓN ENTRE MODO MÍNIMO Y MODO MÁXIMO

Hay dos modos de operación disponibles para los microprocesadores 8086/8088: modo máximo y modo mínimo. La operación en modo mínimo se obtiene mediante la conexión de la terminal de selección de modo **MN/MX** a +5.0 V, y el modo mínimo se selecciona mediante la conexión a tierra de esta terminal. Ambos modos permiten distintas estructuras de control para los microprocesadores 8086/8088. El modo de operación que se proporciona mediante el modo mínimo es similar al del 8085A, el más reciente microprocesador de ocho bits de Intel. El modo máximo es único y está diseñado para utilizarse siempre que exista un coprocesador en un sistema. Hay que tener en cuenta que el modo máximo se retiró de la familia Intel desde el microprocesador 80286.

Operación en modo mínimo

Este modo de operación es la manera menos costosa de operar los microprocesadores 8086/8088 (en la figura 9-19 podrá ver el sistema 8088 en modo mínimo). Cuesta menos debido a que todas las señales de control para la memoria y la E/S se generan mediante el microprocesador. Estas señales de control son idénticas a las del microprocesador 8085A de Intel, uno de los primeros microprocesadores de ocho bits. El modo mínimo permite utilizar periféricos de ocho bits del 8085A con el 8086/8088 sin ninguna consideración especial.

Operación en modo máximo

Este modo de operación difiere del modo mínimo en cuanto a que algunas de las señales de control deben generarse de forma externa. Para ello se requiere agregar un controlador de bus externo: el controlador de bus 8288 (en la figura 9-20 podrá ver el sistema 8088 en modo máximo). No hay suficientes terminales en el 8086/8088 para el control del bus durante el modo máximo ya que las nuevas terminales y las nuevas características han sustituido algunas de estas terminales. El modo máximo se utiliza solamente cuando el sistema contiene coprocesadores externos, como el coprocesador aritmético 8087.

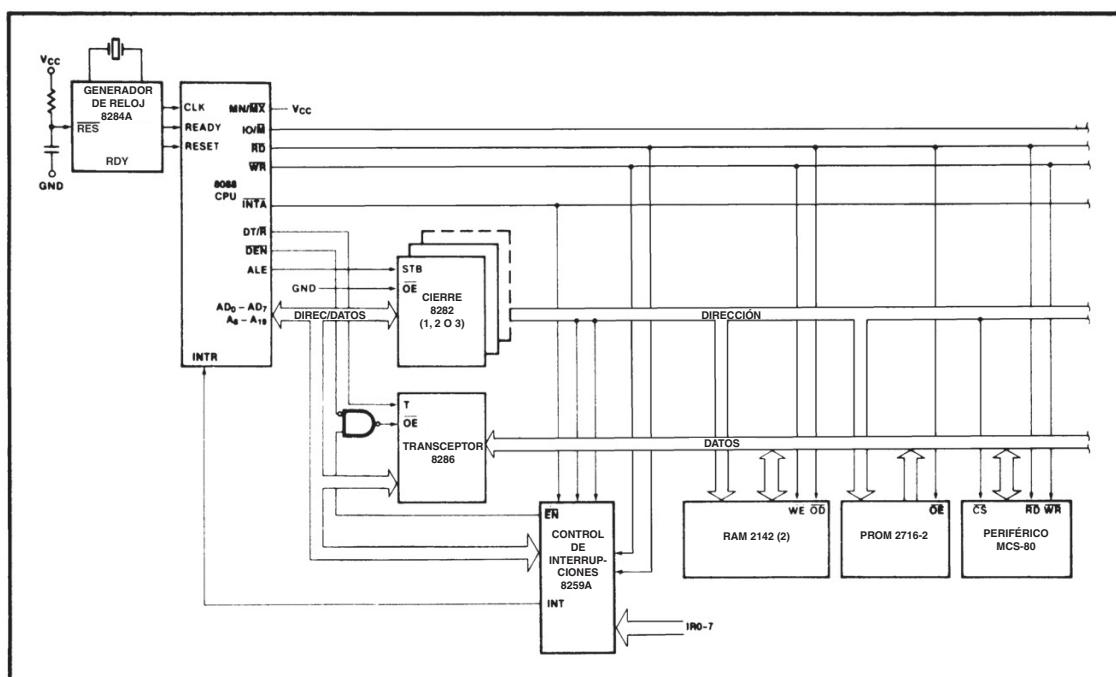


FIGURA 9-19 El sistema 8088 en modo mínimo.

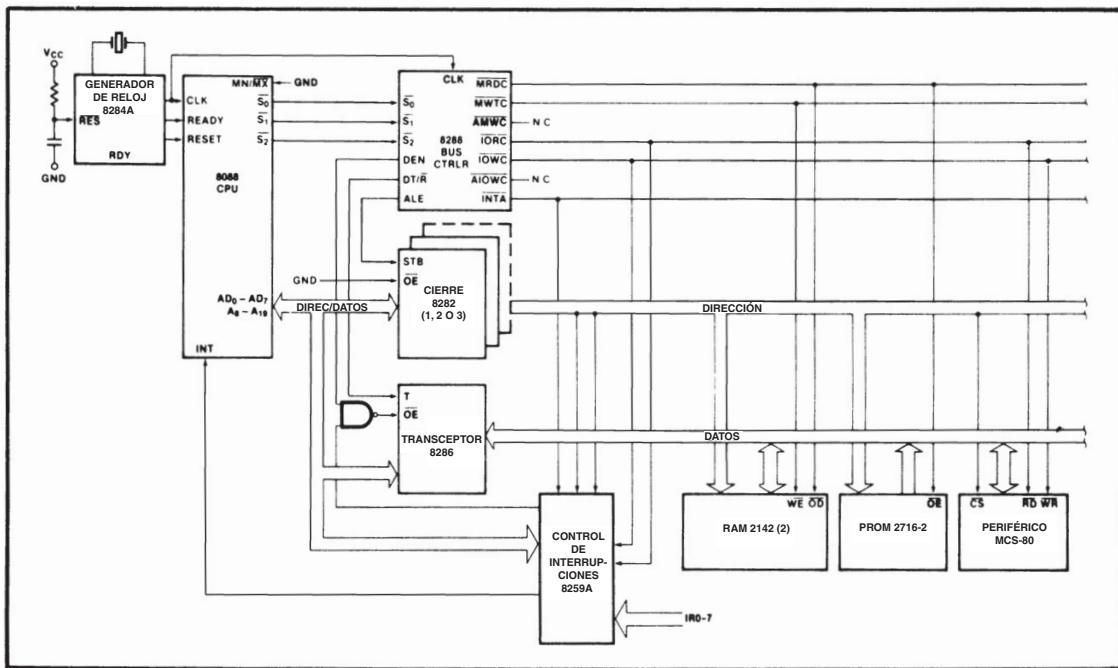


FIGURA 9-20 El sistema 8088 en modo máximo.

El controlador de bus 8288

Un sistema 8086/8088 que se opera en modo máximo debe tener un controlador de bus 8288 para proporcionar las señales que se eliminan del 8086/8088 debido a la operación en modo máximo. La figura 9-21 ilustra el diagrama de bloques y de terminales del controlador de bus 8288.

Observe que el bus de control desarrollado por el controlador de bus 8288 contiene señales separadas para la E/S ($\overline{\text{IORC}}$ e $\overline{\text{IOWC}}$) y la memoria ($\overline{\text{MRDC}}$ y $\overline{\text{MWTC}}$). También contiene estrobos avanzados de escritura de memoria ($\overline{\text{AMWC}}$) y de E/S ($\overline{\text{AIOWC}}$), y la señal $\overline{\text{INTA}}$. Estas señales sustituyen las señales de modo mínimo ALE, WR, IO/M, DT/R, DEN e INTA, las cuales se pierden cuando los microprocesadores 8086/8088 se operan en el modo máximo.

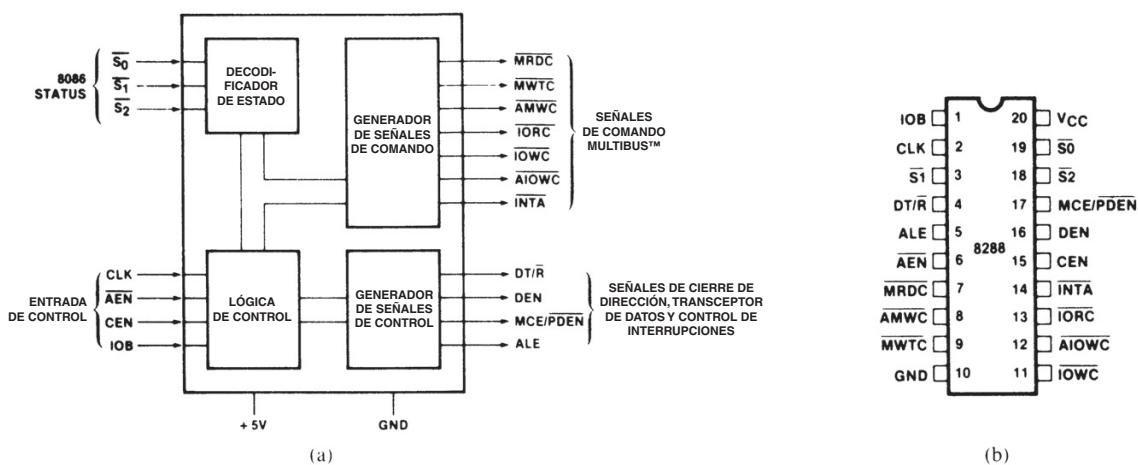


FIGURA 9-21 El controlador de bus 8288; (a) diagrama de bloques y (b) de terminales.

Funciones de las terminales

La siguiente lista proporciona una descripción de cada terminal del controlador de bus 8288.

S₂, S₁ y S₀	Las entradas de estado se conectan a las terminales de salida de estado en el microprocesador 8086/8088. Estas tres señales se decodifican para generar las señales de sincronización para el sistema.
CLK	La entrada reloj proporciona la sincronización interna y debe conectarse a la terminal de salida CLK del generador de reloj 8284A.
ALE	La salida habilita cierre de dirección se utiliza para demultiplexar el bus de direcciones/datos.
DEN	La terminal habilita bus de datos controla los búferes del bus de datos bidireccionales en el sistema. Observe que ésta es una terminal de salida activa en nivel alto, lo cual es la polaridad opuesta de la señal DEN que se encuentra en el microprocesador cuando se opera en el modo mínimo.
DT/R	El 8288 envía la señal transmite/recibe datos para controlar la dirección de los búferes del bus de datos bidireccionales.
AEN	La entrada habilita dirección hace que el 8288 habilite las señales de control de la memoria.
CEN	La entrada habilita control habilita las terminales de salida de comando en el 8288.
IOB	La entrada modo de bus de E/S selecciona la operación en modo de bus de E/S o en modo de bus de sistema.
AIOWC	La señal escritura de E/S avanzada es una salida de comando que se utiliza para proporcionar E/S con una señal de control de escritura de E/S avanzada.
IORC	La salida comando de lectura de E/S proporciona la señal de control de lectura a la E/S.
IOWC	La salida comando de escritura de E/S proporciona la señal de escritura principal a la E/S.
AMWT	La terminal de control escritura de memoria avanzada proporciona una señal de escritura anticipada o avanzada a la memoria.
MWTC	La terminal de control escritura de memoria proporciona la señal de control de escritura normal a la memoria.
MRDC	La terminal de control lectura de memoria proporciona una señal de control de lectura a la memoria.
INTA	La salida reconocimiento de interrupción reconoce una entrada de petición de interrupción que se aplica a la terminal INTR.
MCE/PDEN	La salida datos maestros en cascada/de periférico selecciona la operación en cascada para un controlador de interrupciones si IOB está conectada a tierra, y habilita los transceptores del bus de E/S si IOB está en nivel alto.

1. Las principales diferencias entre el 8086 y el 8088 son: (1) un bus de datos de ocho bits en el 8088 y un bus de datos de 16 bits en el 8086, (2) una terminal SS0 en el 8088 en lugar de BHE/S7 en el 8086, y (3) una terminal IO/M en el 8088 en vez de una terminal M/IO en el 8086.
2. Tanto el 8086 como el 8088 requieren un solo suministro de energía de +5.0 V con una tolerancia de $\pm 10\%$.
3. Los microprocesadores 8086/8088 son compatibles con TTL si la cifra de inmunidad al ruido se reduce a 350 mV, en comparación con los 400 mV usuales.

4. Los microprocesadores 8086/8088 pueden controlar las siguientes cargas unitarias: un 74XX, cinco 74LSXX, un 74SXX, 10 74ALSXX y 10 74HCXX.
5. El generador de reloj 8284A proporciona el reloj del sistema (CLK), la sincronización de READY y la sincronización de RESET.
6. La frecuencia de operación estándar de 5 MHz de los microprocesadores 8086/8088 se obtiene mediante la conexión de un cristal de 15 MHz al generador de reloj 8284A. La salida PCLK contiene una señal compatible con TTL, equivalente a un medio de la frecuencia de CLK.
7. Cada vez que se reinician los microprocesadores 8086/8088, empiezan a ejecutar software en la posición de memoria FFFF0H (FFFF:0000) con la terminal de petición de interrupciones desabilitada.
8. Como los buses del 8086/8088 se multiplexan y la mayoría de los dispositivos de memoria y de E/S no, el sistema debe demultiplexarse antes de conectarse con la memoria o la E/S. La demultiplexión se realiza mediante un cierre de ocho bits, cuyo pulso de reloj se obtiene de la señal ALE.
9. En un sistema grande los buses deben utilizar búferes, ya que los microprocesadores 8086/8088 son capaces de controlar sólo 10 cargas unitarias, y los sistemas grandes por lo general tienen muchas más.
10. La sincronización del bus será un concepto muy importante en los capítulos restantes del libro. Un ciclo de bus que consiste de cuatro periodos de reloj actúa como la sincronización básica del sistema. Cada ciclo de bus puede leer o escribir datos entre el microprocesador y la memoria o entre el microprocesador y el sistema de E/S.
11. Un ciclo de bus se divide en cuatro etapas, o períodos T: el microprocesador utiliza T_1 para enviar la dirección a la memoria o a la E/S, junto con la señal ALE a los demultiplexores; T_2 se utiliza para enviar datos a la memoria para una escritura, y para evaluar la terminal READY y activar las señales de control \overline{RD} o \overline{WR} ; T_3 otorga tiempo a la memoria para acceder a los datos y permite transferirlos entre el microprocesador y la memoria o la E/S; y T_4 es en donde se escriben los datos.
12. Los microprocesadores 8086/8088 permiten a la memoria y la E/S 460 ns para acceder a los datos cuando se operan con un reloj de 5 MHz.
13. Los estados de espera (T_W) alargan el ciclo de bus uno o más períodos de reloj para permitir a la memoria y a la E/S un tiempo de acceso adicional. Los estados de espera se insertan mediante el control de la entrada READY que va al 8086/8088. READY se muestra al final de T_2 y durante T_W .
14. La operación en modo mínimo es similar a la del microprocesador Intel 8085A. La operación en modo máximo es nueva y está diseñada específicamente para la operación del coprocesador aritmético 8087.
15. El controlador de bus 8288 debe utilizarse en el modo máximo para proporcionar las señales del bus de control a la memoria y la E/S. Esto se debe a que la operación en modo máximo del 8086/8088 elimina algunas de las líneas de señales de control del sistema y las sustituye con señales de control para los coprocesadores. El 8288 reconstruye estas señales de control que se eliminan.

9-8**PREGUNTAS Y PROBLEMAS**

1. Liste las diferencias entre los microprocesadores 8086 y 8088.
2. ¿Es el 8086/8088 compatible con TTL? Explique su respuesta.
3. ¿Cuál es el factor de salida del 8086/8088 para los siguientes dispositivos?:
 (a) 74XXX TTL.
 (b) 74ALSXXX TTL.
 (c) 74HCXXX CMOS.
4. ¿Qué información aparece en el bus de direcciones/datos del 8088 cuando ALE está activa?
5. ¿Cuáles son los fines de los bits de estado S_3 y S_4 ?
6. ¿Qué condición indica un 0 lógico en la terminal RD del 8086/8088?

7. Explique la operación de la terminal TEST y de la instrucción WAIT.
8. Describa la señal que se aplica a la terminal de entrada CLK de los microprocesadores 8086/8088.
9. ¿Qué modo de operación se selecciona cuando MN/MX se conecta a tierra?
10. ¿Qué indica la señal de estrobo WR que proviene del 8086/8088 sobre la operación del 8086/8088?
11. ¿Cuándo flota ALE a su estado de alta impedancia?
12. Cuando DT/R es un 1 lógico, ¿qué condición indica sobre la operación del 8087/8088?
13. ¿Qué ocurre cuando la entrada HOLD para el 8086/8088 se coloca en su nivel 1 lógico?
14. ¿Qué terminales del 8086/8088 en modo mínimo se decodifican para descubrir si el procesador se detiene?
15. Explique la operación de la terminal LOCK.
16. ¿Qué condiciones indican las terminales QS₁ y QS₀ sobre el 8086/8088?
17. ¿Cuáles son las tres tareas de limpieza que proporciona el generador de reloj 8284A?
18. ¿Por cuál factor divide el generador de reloj 8284A la frecuencia de salida del oscilador de cristal?
19. Si la terminal F/C se coloca en un nivel de 1 lógico, el oscilador de cristal se deshabilita. ¿En dónde se conecta la señal de entrada de sincronización al 8284A bajo esta condición?
20. La salida PCLK del 8284A es de _____ MHz si el oscilador de cristal opera a 14 MHz.
21. La entrada RES que va al 8284A se coloca en un nivel de _____ lógico para reiniciar el 8086/8088.
22. ¿Qué conexiones de bus en el microprocesador 8086 se demultiplexan de manera usual?
23. ¿Qué conexiones de bus en el microprocesador 8088 se demultiplexan de manera usual?
24. ¿Qué circuito integrado TTL se utiliza a menudo para demultiplexar los buses en el 8086/8088?
25. ¿Cuál es el propósito de la señal BHE demultiplexada en el microprocesador 8086?
26. ¿Por qué se requieren búferes con frecuencia en un sistema basado en el 8086/8088?
27. ¿Qué señal del 8086/8088 se utiliza para seleccionar la dirección de los flujos de datos que pasan por el búfer de bus bidireccional 74LS245?
28. Un ciclo de bus equivale a _____ períodos de reloj.
29. Si la entrada CLK que va al 8086/8088 es de 4 MHz, ¿cuál es la longitud de un ciclo de bus?
30. ¿Cuáles son las dos operaciones del 8086/8088 que ocurren durante un ciclo de bus?
31. ¿Cuántos MIPS es capaz de obtener el 8086/8088 cuando opera con un reloj de 10 MHz?
32. Describa en forma breve el propósito de cada uno de los estados T que se muestran:
 - (a) T₁.
 - (b) T₂.
 - (c) T₃.
 - (d) T₄.
 - (e) T_w.
33. ¿Cuánto tiempo se permite para el acceso a memoria cuando el 8086/8088 se opera con un reloj de 5 MHz?
34. ¿Qué anchura tiene DEN si el 8088 se opera con un reloj de 5 MHz?
35. Si la terminal READY se conecta a tierra, introducirá _____ estados en el ciclo de bus del 8086/8088.
36. ¿Qué hace la entrada ASYNC que va al 8284A?
37. ¿Qué niveles lógicos deben aplicarse a AEN1 y RDY₁ para obtener un 1 lógico en la terminal READY? (Suponga que AEN₂ está en un nivel de 1 lógico.)
38. Compare la operación en modo mínimo con la de modo máximo del 8086/8088.
39. ¿Qué función principal proporciona el controlador de bus 8288 cuando se utiliza con la operación en modo máximo del 8086/8088?

CAPÍTULO 10

Interfaz de memoria

INTRODUCCIÓN

Todo sistema basado en microprocesador, ya sea simple o complejo, tiene un sistema de memoria. La familia Intel de microprocesadores no es distinta de cualquier otra en este caso. Casi todos los sistemas contienen dos tipos principales de memoria: memoria de sólo lectura (ROM) y memoria de acceso aleatorio (RAM), también conocida como memoria de lectura/escritura. La ROM contiene software y datos permanentes del sistema, mientras que la RAM contiene datos temporales y software de aplicaciones. En este capítulo explicaremos cómo crear interfaces entre ambos tipos de memoria y la familia Intel de microprocesadores. Demostraremos la interfaz de memoria para un bus de datos de 8, 16, 32 y 64 bits mediante el uso de varios tamaños de direcciones de memoria. Esto permite crear una interfaz entre casi cualquier microprocesador y cualquier sistema de memoria.

OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

1. Decodificar la dirección de memoria y utilizar las salidas del decodificador para seleccionar varios componentes de la memoria.
2. Utilizar dispositivos lógicos programables (PLDs) para decodificar direcciones de memoria.
3. Explicar cómo crear una interfaz entre la RAM y la ROM con un microprocesador.
4. Explicar cómo utilizar el código de corrección de errores (ECC) con la memoria.
5. Crear una interfaz entre la memoria y un bus de datos de 8, 16, 32 y 64 bits.
6. Explicar la operación de un controlador de RAM dinámica.
7. Crear una interfaz entre la RAM dinámica y el microprocesador.

10-1

DISPOSITIVOS DE MEMORIA

Antes de tratar de crear una interfaz entre la memoria y el microprocesador, es esencial comprender completamente la operación de los componentes de la memoria. En esta sección explicaremos las funciones de los cuatro tipos comunes de memoria: memoria de sólo lectura (ROM), memoria flash (EEPROM), memoria estática de acceso aleatorio (SRAM) y memoria dinámica de acceso aleatorio (DRAM).

Conexiones de las terminales de memoria

Las conexiones de las terminales que son comunes para todos los dispositivos de memoria son las entradas de dirección, las salidas o entradas/salidas de datos, cierto tipo de entrada de selección y por lo menos una entrada de control que se utiliza para seleccionar una operación de lectura o de escritura. En la figura 10-1 podrá ver los dispositivos de memoria ROM y RAM genéricos.

Conecciones de dirección. Todos los dispositivos de memoria tienen entradas de dirección que seleccionan una posición dentro del dispositivo de memoria. Casi siempre las entradas de dirección son etiquetadas desde A_0 , la entrada de dirección menos significativa, hasta A_n , en donde el subíndice n puede ser cualquier valor, pero siempre se etiqueta como uno menos que el número total de terminales de dirección. Por ejemplo, un dispositivo de memoria con 10 terminales de dirección tiene sus terminales de dirección etiquetadas desde A_0 hasta A_9 . El número de terminales de dirección que tiene un dispositivo de memoria se determina sobre la base del número de posiciones de memoria que contiene.

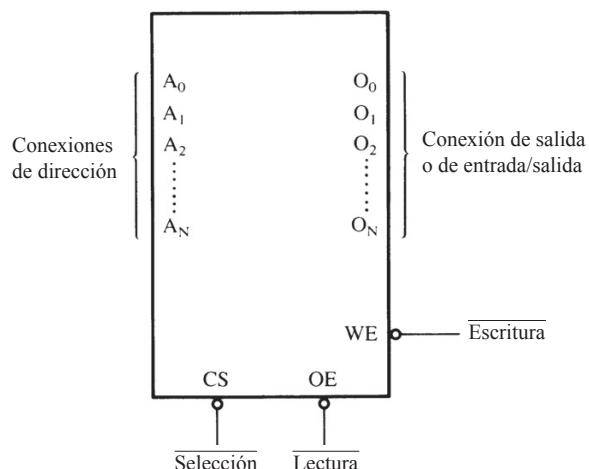
En la actualidad, los dispositivos de memoria más comunes tienen entre 1 K (1024) y 512 M (536,870,912) posiciones de memoria, aunque ya existen dispositivos con 1 G o más posiciones de memoria. Un dispositivo con 1 K de memoria tiene 10 terminales de dirección (A_0-A_9); por lo tanto, se requieren 10 terminales de dirección para seleccionar cualquiera de sus 1024 posiciones de memoria. Se requiere un número binario de 10 bits (1024 combinaciones distintas) para seleccionar cualquier posición en un dispositivo con 1024 posiciones. Si un dispositivo de memoria tiene 11 conexiones de dirección (A_0-A_{11}), cuenta con 2048 (2 K) posiciones internas de memoria. Por lo tanto, el número de posiciones de memoria puede extrapolarse basándose en el número de terminales de dirección. Por ejemplo, un dispositivo de memoria de 4 K tiene 12 conexiones de dirección, un dispositivo de 8 K tiene 13, y así sucesivamente. Un dispositivo que contiene 1 M posiciones requiere de una dirección de 20 bits (A_0-A_{19}).

El número 400H representa una sección de 1 Kbyte del sistema de memoria. Si un dispositivo de memoria se decodifica para comenzar en la dirección de memoria 10000H y es un dispositivo de 1 K, su última posición se encuentra en la dirección 103FFH; una posición menos que 400H. Otro número hexadecimal importante a recordar es 1000H, ya que equivale a 4 K. Un dispositivo de memoria que contiene una dirección inicial de 14000H y que tiene 4 Kbytes termina en la posición 14FFFFH; una posición menos que 1000H. Un tercer número a recordar es 64 K, o 10000H. Una memoria que empieza en la posición 30000H y termina en la posición 3FFFFH es una memoria de 64 Kbytes. Por último y como 1 M de memoria es una cantidad común, tenemos que 1 M de memoria contiene 100000H posiciones de memoria.

Conecciones de datos. Todos los dispositivos de memoria tienen un conjunto de salidas o entradas/salidas de datos. El dispositivo que se muestra en la figura 10-1 tiene un conjunto común de conexiones de entrada/salida (E/S). En la actualidad, muchos dispositivos de memoria tienen terminales de E/S comunes bidireccionales.

Las conexiones de datos están en los puntos en lo que se introducen datos para su almacenamiento, o se extraen para su lectura. Las terminales de datos en los dispositivos de memoria se etiquetan desde D_0 hasta D_7 para un dispositivo de memoria de ocho bits de anchura. En este dispositivo de memoria de ejemplo hay ocho conexiones de E/S, lo que significa que el dispositivo de memoria almacena ocho bits de datos en cada una de sus posiciones de memoria. Por lo general, a un dispositivo de memoria de ocho bits se le conoce como memoria **tipo byte**. Aunque la mayoría de los dispositivos de la actualidad son de ocho bits, algunos son de 16 bits, de cuatro bits o incluso de un bit.

FIGURA 10-1 Un seudocomponente de memoria en el que se muestran las conexiones de dirección, datos y control.



Los listados de los catálogos de dispositivos de memoria a menudo hacen referencia a las posiciones de memoria multiplicadas por los bits de cada posición. Por ejemplo, el fabricante por lo común lista un dispositivo de memoria con 1 K posiciones de memoria y ocho bits en cada posición como $1\text{ K} \times 8$. Un dispositivo de $16\text{ K} \times 1$ es un dispositivo de memoria que contiene 16 K posiciones de memoria de 1 bit cada una. A menudo, los dispositivos de memoria se clasifican de acuerdo con la capacidad total de bits. Por ejemplo, un dispositivo de memoria de $1\text{ K} \times 8$ bits se lista algunas veces como un dispositivo de memoria de 8 K, o una memoria de $64\text{ K} \times 4$ se lista como un dispositivo de 256 K. Estas variaciones se dan entre los distintos fabricantes.

Conexiones de selección. Cada dispositivo de memoria tiene una entrada (algunas veces más de una) que selecciona o habilita el dispositivo de memoria. Por lo general, a este tipo de entrada se le conoce como **selección de chip** ($\overline{\text{CS}}$), **habilitación de chip** ($\overline{\text{CE}}$) o simplemente como entrada de **selección** ($\overline{\text{S}}$). Es común que la memoria RAM tenga cuando menos una entrada $\overline{\text{CS}}$ o $\overline{\text{S}}$, y que la ROM tenga cuando menos una entrada $\overline{\text{CE}}$. Si la entrada $\overline{\text{CE}}$, $\overline{\text{CS}}$ o $\overline{\text{S}}$ está activa (un 0 lógico en este caso, debido a la barra superior), el dispositivo de memoria realiza una operación de lectura o de escritura; si está inactiva (un 1 lógico en este caso), el dispositivo de memoria no puede realizar una lectura o escritura debido a que está desconectado o deshabilitado. Si hay más de una conexión $\overline{\text{CS}}$ presente, todas deben activarse para leer o escribir datos.

Conexiones de control. Todos los dispositivos de memoria cuentan con algún tipo de entrada o entradas de control. Por lo general, una ROM tiene sólo una entrada de control, mientras que una RAM tiene una o dos entradas de control.

La entrada de control que se encuentra con más frecuencia en una ROM es la conexión **habilita salida** ($\overline{\text{OE}}$) o **compuerta** ($\overline{\text{G}}$), la cual permite que los datos fluyan de las terminales de datos de salida de la ROM. Si $\overline{\text{OE}}$ y la entrada de selección ($\overline{\text{CE}}$) están activas, la salida está habilitada; si $\overline{\text{OE}}$ está inactiva, la salida está deshabilitada en su estado de alta impedancia. La conexión $\overline{\text{OE}}$ habilita y deshabilita un conjunto de búferes de tres estados que se encuentran dentro del dispositivo de memoria y deben estar activos para poder leer datos.

Un dispositivo de memoria RAM tiene una o dos entradas de control. Si sólo hay una entrada de control, a menudo se le conoce como $\text{R}/\overline{\text{W}}$. Esta terminal selecciona una operación de lectura o una operación de escritura sólo si el dispositivo está seleccionado sobre la base de la entrada de selección ($\overline{\text{CS}}$). Si la RAM tiene dos entradas de control, por lo general se etiquetan como $\overline{\text{WE}}$ (o $\overline{\text{W}}$) y $\overline{\text{OE}}$ (o $\overline{\text{G}}$). Aquí, $\overline{\text{WE}}$ (**habilita lectura**) debe estar activa para realizar una escritura en memoria, y $\overline{\text{OE}}$ debe estar activa para realizar una operación de lectura de memoria. Cuando están presentes estos dos controles ($\overline{\text{WE}}$ y $\overline{\text{OE}}$), nunca deben estar activos al mismo tiempo. Si ambas entradas de control están inactivas (1 lógico) nunca se escriben ni se leen datos, y las conexiones de datos están en su estado de alta impedancia.

Memoria ROM

La **memoria de sólo lectura** (ROM) almacena en forma permanente programas y datos que están residentes para el sistema y no deben cambiar cuando se desconecta la fuente de energía. La ROM se programa de manera permanente de forma que los datos siempre estén presentes, aún y cuando se desconecte la fuente de energía. A este tipo de memoria se le conoce comúnmente como **memoria no volátil**, debido a que su contenido *no* cambia incluso si se desconecta la energía.

En la actualidad, la ROM está disponible en muchas formas. Un dispositivo denominado ROM se compra en cantidades masivas de un fabricante y se programa durante su fabricación. La EPROM (**memoria de sólo lectura programable borrable**) es un tipo de ROM que se utiliza con más frecuencia cuando el software debe cambiarse a menudo o cuando hay muy pocas en demanda para que la ROM sea económica. Para que una ROM sea práctica, por lo general debemos comprar cuando menos 10,000 dispositivos para recuperar el cargo de programación de la fábrica. Una EPROM se programa en el campo, en un dispositivo llamado *programador de EPROM*. La EPROM también puede borrarse si se expone a una luz ultravioleta de alta intensidad durante 20 minutos aproximadamente, dependiendo del tipo de EPROM.

También hay dispositivos de memoria PROM, aunque no son tan comunes en la actualidad. La PROM (**memoria de sólo lectura programable**) también se programa en el campo mediante el proceso de quemar pequeños fusibles de NI-cromo u óxido de silicio; pero una vez que se programa, no puede borrarse.

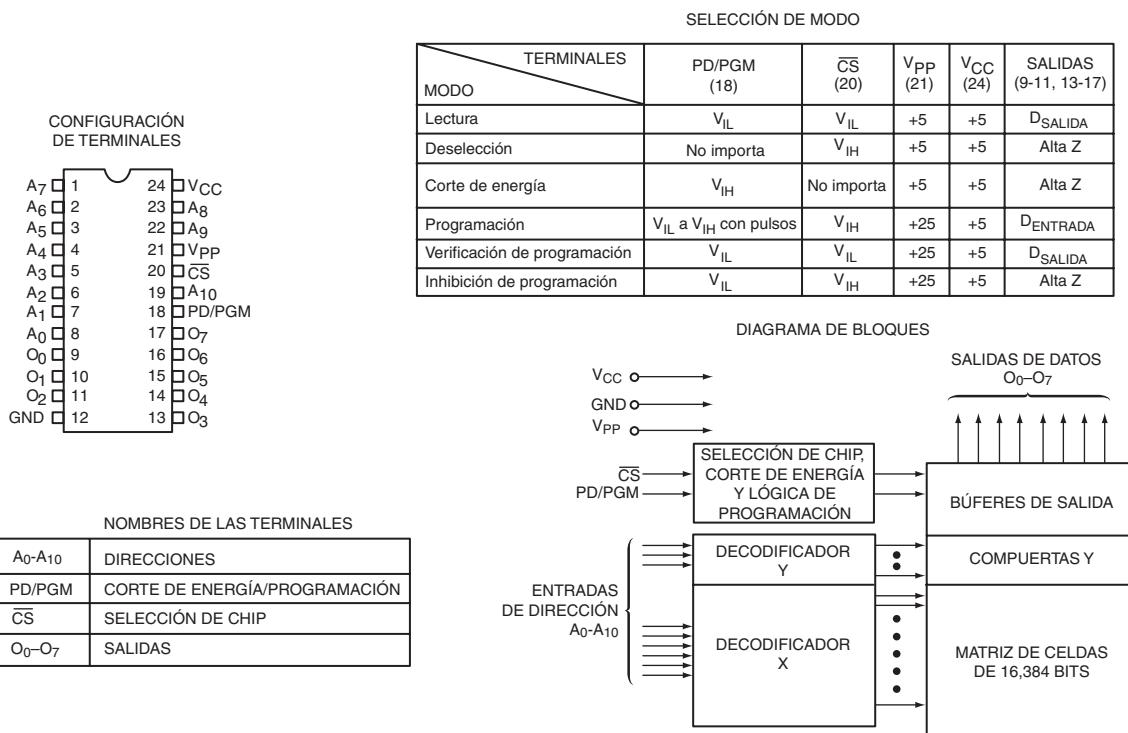


FIGURA 10-2 El diagrama de terminales de la EPROM 2716, 2K × 8. (Cortesía de Intel Corporation.)

Otro tipo más reciente de memoria llamado **memoria principalmente de lectura** (RMM) es la **memoria Flash**.¹ A este tipo de memoria también se le conoce como EEPROM (**ROM programable borrable por electricidad**), EAROM (**ROM alterable por electricidad**) o NOVRAM (**RAM no volátil**). Por lo general, estos dispositivos de memoria pueden borrarse en el sistema mediante electricidad, pero requieren más tiempo para borrarse que una RAM normal. El dispositivo de memoria Flash se utiliza para almacenar información de configuración en sistemas tales como la tarjeta de vídeo en la computadora. Lo único que no ha sustituido aún es la EPROM en la mayoría de los sistemas computacionales para la memoria del BIOS. Algunos sistemas contienen una contraseña almacenada en el dispositivo de memoria Flash. Esta memoria ha hecho su mayor impacto en las tarjetas de memoria para las cámaras digitales y la memoria en los reproductores de audio MP3.

La figura 10-2 muestra la EPROM 2716, que representa a la mayoría de las EPROMs comunes. Este dispositivo contiene 11 entradas de dirección y ocho salidas de datos. La EPROM 2716 es un dispositivo de memoria de sólo lectura de 2 K × 8. La serie 27XXX de EPROMs consta de los siguientes números de pieza: 2704 (512 × 8), 2708 (1 K × 8), 2716 (2 K × 8), 2732 (4 K × 8), 2764 (8 K × 8), 27128 (16 K × 8), 27256 (32 K × 8), 27512 (64 K × 8) y 271024 (128 K × 8). Cada una de estas piezas contiene terminales de dirección, ocho conexiones de datos, una o más entradas de selección de chip (CE) y una terminal de habilitación de la salida (OE).

La figura 10-3 muestra el diagrama de sincronización para la EPROM 2716. Los datos aparecen en las conexiones de salida sólo después de que se coloca un 0 lógico en las conexiones de las terminales CE y OE. Si CE y OE no son ambas 0 lógico, las conexiones de salida de datos permanecen en sus estados de alta impedancia o desconectadas. Observe que la terminal V_{PP} debe colocarse en el nivel de 1 lógico para que puedan leerse datos de la EPROM. En algunos casos, la terminal V_{PP} está en la misma posición que la terminal WE en la SRAM. Esto permite que un solo zócalo almacene ya sea una EPROM o una SRAM. Un ejemplo es la EPROM 27256 y la SRAM 62256, ambos dispositivos de 32 K × 8 que tienen el mismo diagrama de terminales, excepto por V_{PP} en la EPROM y WE en la SRAM.

¹ La memoria Flash es marca registrada de Intel Corporation..

Características de C.A.

$T_A = 0^\circ\text{C}$ a 70°C , $V_{CC}^{[1]} = +5V \pm 5\%$, $V_{PP}^{[2]} = V_{CC} \pm 0.6V^{[3]}$

Símbolo	Parámetro	Límites			Unidad	Condiciones de prueba
		Mín.	Tipo ^[4]	Máx.		
t_{ACC1}	Retraso de Dirección a Salida			250 450	ns	$\text{PD/PGM} = \overline{\text{CS}} = V_{IL}$
t_{ACC2}	Retraso de PD/PGM a Salida			280 450	ns	$\overline{\text{CS}} = V_{IL}$
t_{CO}	Retraso de Selección de chip a Salida			120	ns	$\text{PD/PGM} = V_{IL}$
t_{PF}	Flotamiento de PD/PGM a Salida	0		100	ns	$\overline{\text{CS}} = V_{IL}$
t_{DF}	Flotamiento de Deselección de chip a Salida	0		100	ns	$\text{PD/PGM} = V_{IL}$
t_{OH}	Retención (hold) de Dirección a Salida	0			ns	$\text{PD/PGM} = \overline{\text{CS}} = V_{IL}$

Capacitancia^[5] $T_A = 25^\circ\text{C}$, $f = 1\text{ MHz}$

Símbolo	Parámetro	Tipo	Máx.	Unidad	Condiciones
$C_{ENTRADA}$	Entrada de capacitancia	4	6	pF	$V_{ENTRADA} = 0V$
C_{SALIDA}	Salida de capacitancia	8	12	pF	$V_{SALIDA} = 0V$

Condiciones de prueba de C.A.:

Carga de salida: 1 compuerta TTL y $C_L = 100\text{ pF}$
 Tiempos de elevación y caída de la entrada: $\leq 20\text{ ns}$
 Niveles del pulso de entrada: 0.8V a 2.2V
 Nivel de referencia de medición de sincronización:
 Entradas 1 V y 2 V
 Salidas 0.8 V y 2 V

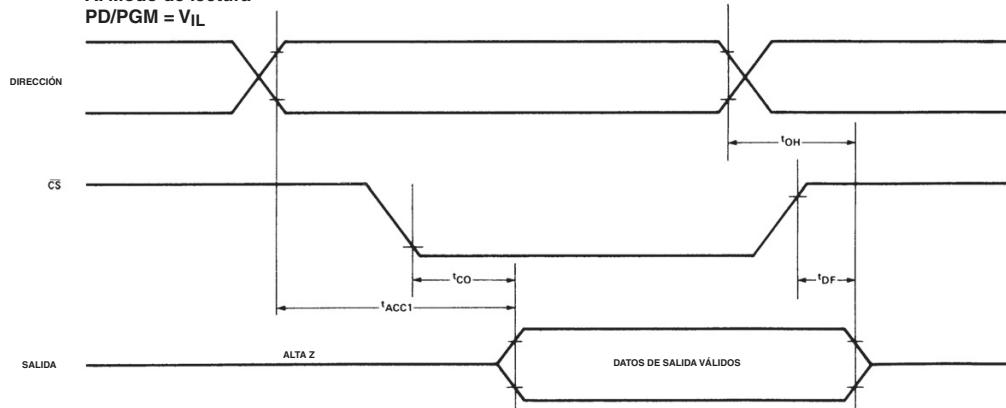
FORMAS DE ONDA**A. Modo de lectura**

FIGURA 10-3 El diagrama de sincronización de las características de CA de la EPROM 2716.
 (Cortesía de Intel Corporation.)

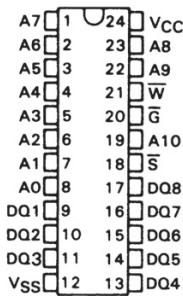
Una importante pieza de información que proporcionan el diagrama de sincronización y la ficha técnica es el tiempo de acceso a la memoria: el tiempo que requiere la memoria para leer información. Como se muestra en la figura 10-3, el tiempo de acceso a memoria (T_{ACC}) se mide desde que aparece la dirección en las entradas de dirección hasta que aparecen los datos en las conexiones de salida. Esto se basa en la suposición de que la entrada $\overline{\text{CE}}$ se va al nivel bajo al mismo tiempo que las entradas de dirección se vuelven estables. Además, $\overline{\text{OE}}$ debe ser un 0 lógico para que se activen las conexiones de salida. La velocidad básica de esta EPROM es de 450 ns. (Recuerde que el 8086/8088 que opera con un reloj de 5 MHz permite que la memoria de 460 ns acceda a los datos.) Este tipo de componente de memoria requiere que los estados de espera operen en forma correcta con los microprocesadores 8086/8088, debido a su tiempo de acceso considerablemente largo. Si no se desean estados de espera, hay disponibles versiones de mayor velocidad de la EPROM por un costo adicional. En la actualidad, la memoria EPROM está disponible con tiempos de acceso de hasta 100 ns. Es obvio que se requieren estados de espera en los microprocesadores modernos para cualquier dispositivo EPROM.

Dispositivos de RAM estática (SRAM)

Los dispositivos de memoria RAM estática retienen los datos mientras se aplique energía CD. Como no se requiere ninguna acción especial (excepto la energía) para retener los datos almacenados, estos

FIGURA 10-4 El diagrama de terminales de la RAM estática (SRAM) TMS4016 de $2\text{ K} \times 8$. (Cortesía de Texas Instruments Incorporated.)

TMS4016...ENCAPSULADO NL
(VISTA SUPERIOR)



NOMENCLATURA DE TERMINALES	
AO – A10	Direcciones
DQ1 – DQ8	Datos de entrada/Datos de salida
G-bar	Habilitación de la salida
S-bar	Selección de chip
Vcc	Suministro de +5 V
Vss	Tierra
W-bar	Habilitación de lectura

dispositivos se llaman **memoria estática**. También se les llama **memoria volátil** porque no retienen los datos sin energía. La principal diferencia entre una ROM y una RAM es que se puede escribir en la RAM bajo condiciones normales de operación, mientras que una ROM se programa fuera de la computadora y, por lo general, sólo se lee. La SRAM, que almacena datos temporales, se utiliza cuando el tamaño de la memoria de lectura/escritura es relativamente pequeño. En la actualidad, se considera que una memoria es pequeña si tiene menos de 1 Mbyte.

La figura 10-4 muestra la SRAM 4016, que es una memoria de lectura/escritura de $2\text{ K} \times 8$. Este dispositivo tiene 11 entradas de dirección y ocho conexiones de datos de entrada/salida. Este dispositivo representa a todos los dispositivos SRAM, excepto por el número de conexiones de dirección y de datos.

Las entradas de control de esta RAM son un poco distintas de las que presentamos anteriormente. La terminal \overline{OE} se etiqueta como \overline{G} , la terminal \overline{CS} es \overline{S} y la terminal \overline{WE} es \overline{W} . A pesar de las designaciones alteradas, las terminales de control funcionan de igual forma que las que describimos anteriormente. Otros fabricantes manejan esta popular SRAM con los números de pieza 2016 y 6116.

La figura 10-5 muestra el diagrama de sincronización para la SRAM 4016. Como la sincronización del ciclo de lectura nos indica, el tiempo de acceso es $t_{a(A)}$. En la versión más lenta de la 4016 este tiempo es de 250 ns, que es lo bastante rápido como para conectarse directamente con un 8088 o con un 8086 que opere a 5 MHz sin estados de espera. De nuevo, es importante recordar que debe verificarse el tiempo de acceso para determinar la compatibilidad de los componentes de memoria con el microprocesador.

La figura 10-6 muestra el diagrama de terminales de la RAM estática 62256, de $32\text{ K} \times 8$. Este dispositivo viene empaquetado en un circuito integrado de 28 terminales y está disponible con tiempos de acceso de 120 ns o de 150 ns. Otros dispositivos SRAM comunes están disponibles en tamaños de $8\text{ K} \times 8$, $128\text{ K} \times 8$, $256\text{ K} \times 8$, $512\text{ K} \times 8$ y $1\text{ M} \times 8$, con tiempos de acceso de hasta 1.0 ns para la SRAM que se utiliza en los sistemas de memoria caché de las computadoras.

Memoria RAM dinámica (DRAM)

La RAM estática más grande disponible actualmente es de $1\text{ M} \times 8$. Por otro lado, la RAM dinámica está disponible en tamaños mucho más grandes: hasta $512\text{ M} \times 1$. En cuanto a las demás características, la DRAM es en esencia igual que la SRAM, sólo que retiene los datos por sólo 2 o 4 ms en un capacitor integrado. Después de 2 o 4 ms, el contenido de la DRAM debe reescribirse (*refrescarse*) ya que los capacitores, que almacenan un 1 o un 0 lógico, pierden su carga.

En vez de requerir la casi imposible tarea de leer el contenido de cada posición de memoria con un programa y luego renovarla, el fabricante ha diseñado la construcción interna de la DRAM de manera distinta a la SRAM. En la DRAM, todo el contenido de la memoria se refresca con 256 lecturas en un intervalo de 2 o 4 ms. La operación de refrescar también se produce durante una escritura, una lectura o durante un ciclo especial para refrescar. En la sección 10-6 proporcionaremos mucha más información sobre el proceso de refrescar la DRAM.

Otra desventaja de la memoria DRAM es que requiere muchas terminales de dirección por lo cual los fabricantes han decidido multiplexar las entradas de dirección. La figura 10-7 muestra una DRAM TMS4464 de 64 K × 4, la cual almacena 256 K bits de datos. Observe que sólo contiene ocho entradas

características eléctricas en el rango recomendado de temperatura de operación al aire libre (a menos que se indique lo contrario)

PARÁMETRO	CONDICIONES DE PRUEBA	MÍN	TIPO†	MÁX	UNIDAD
V_{OH} Voltaje de nivel alto	$I_{OH} = -1 \text{ mA}, V_{CC} = 4.5 \text{ V}$	2.4			V
V_{OL} Voltaje de nivel bajo	$I_{OL} = 2.1 \text{ mA}, V_{CC} = 4.5 \text{ V}$			0.4	V
I_I Corriente de entrada	$V_I = 0 \text{ V a } 5.5 \text{ V}$			10	μA
I_{OZ} Corriente de salida en estado desconectado	$S \text{ o } G \text{ a } 2 \text{ V o } W \text{ a } 0.8 \text{ V}, V_O = 0 \text{ V a } 5.5 \text{ V}$			10	μA
I_{CC} Corriente de suministro de V_{CC}	$I_O = 0 \text{ mA}, V_{CC} = 5.5 \text{ V}, T_A = 0^\circ\text{C}$ (peor caso)	40		70	mA
C_I Capacitancia de entrada	$V_I = 0 \text{ V}, f = 1 \text{ MHz}$			8	pF
C_O Capacitancia de salida	$V_O = 0 \text{ V}, f = 1 \text{ MHz}$			12	pF

†Todos los valores típicos son a $V_{CC} = 5 \text{ V}, T_A = 25^\circ\text{C}$

requerimientos de sincronización en los rangos recomendados de suministro de voltaje y de temperatura de operación al aire libre

PARÁMETRO	TMS4016-12	TMS4016-15	TMS4016-20	TMS4016-25	UNIDAD
					MÍN
$t_{C(rd)}$ Tiempo de ciclo de lectura	120	150	200	250	ns
$t_{C(wr)}$ Tiempo de ciclo de escritura	120	150	200	250	ns
$t_w(W)$ Anchura de pulso de escritura	60	80	100	120	ns
$t_{SU(A)}$ Tiempo de preparación de dirección	20	20	20	20	ns
$t_{SU(S)}$ Tiempo de preparación de selección de chip	60	80	100	120	ns
$t_{SU(D)}$ Tiempo de preparación de datos	50	60	80	100	ns
$t_h(A)$ Tiempo de retención de dirección	0	0	0	0	ns
$t_h(D)$ Tiempo de retención de datos	5	10	10	10	ns

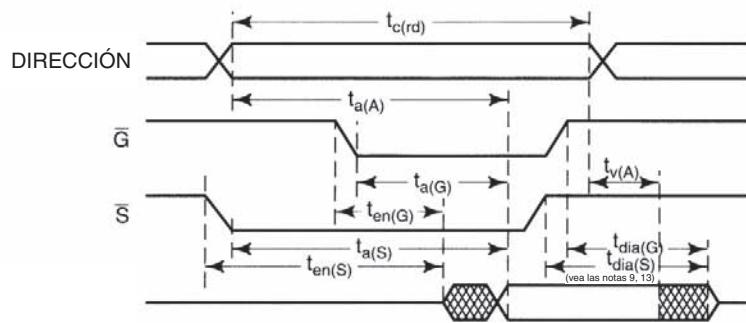
características de conmutación en el rango recomendado de voltaje, $T_A = 0^\circ\text{C}$ a 70°C

PARÁMETRO	TMS4016-12	TMS4016-15	TMS4016-20	TMS4016-25	UNIDAD
					MÍN
$t_{a(A)}$ Tiempo de acceso de dirección	120	150	200	250	ns
$t_{a(S)}$ Tiempo de acceso de selección de chip en nivel bajo	60	75	100	120	ns
$t_{a(G)}$ Tiempo de acceso de habilitación de salida en nivel bajo	50	60	80	100	ns
$t_{V(A)}$ Datos de salida válidos después del cambio de dirección	10	15	15	15	ns
$t_{dis(S)}$ Tiempo de deshabilitación de salida después de selección de chip en nivel alto	40	50	60	80	ns
$t_{dis(G)}$ Tiempo de deshabilitación de salida después de habilitación de salida en nivel alto	40	50	60	80	ns
$t_{dis(W)}$ Tiempo de deshabilitación de salida después de habilitación de escritura en nivel bajo	50	60	60	80	ns
$t_{en(S)}$ Tiempo de habilitación de salida después de selección de chip en nivel bajo	5	5	10	10	ns
$t_{en(G)}$ Tiempo de habilitación de salida después de habilitación de salida en nivel bajo	5	5	10	10	ns
$t_{en(W)}$ Tiempo de habilitación de salida después de habilitación de escritura en nivel alto	5	5	10	10	ns

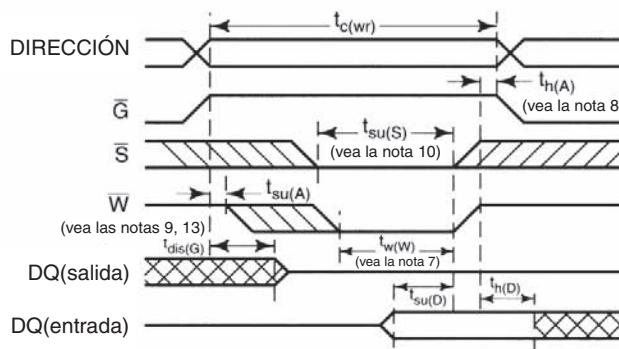
NOTAS: 3. $C_L = 100 \text{ pF}$ para todas las mediciones, excepto para $t_{dis(W)}$ y $t_{en(W)}$.
 $C_L = 5 \text{ pF}$ para $t_{dis(W)}$ y $t_{en(W)}$.
4. Los parámetros $t_{dis(W)}$ y $t_{en(W)}$ se muestran y no se evalúan al 100%.

FIGURA 10-5 (a) Las características de CA de la SRAM TMS4016. (b) Los diagramas de sincronización de la SRAM TMS4016. (Cortesía de Texas Instruments Incorporated.)

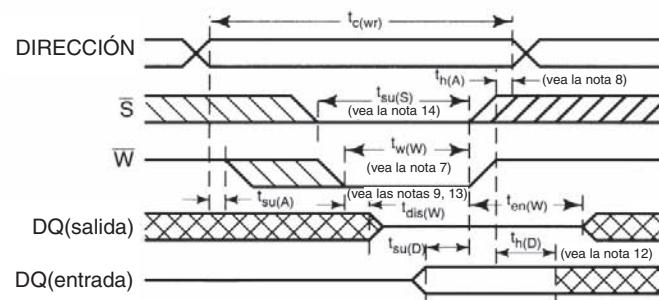
forma de onda de la sincronización del ciclo de lectura (vea la nota 5)



forma de onda de la sincronización del ciclo de escritura núm. 1 (vea la nota 6)



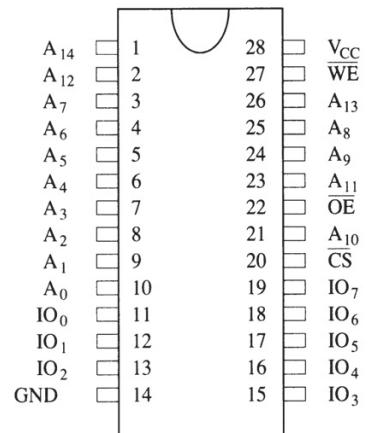
forma de onda de la sincronización del ciclo de escritura núm. 2 (vea las notas 6 y 11)



NOTAS:

5. \overline{W} es Ciclo de lectura en nivel alto.
6. \overline{W} debe estar en nivel alto durante todas las transiciones de dirección.
7. Se produce una escritura durante el traslape de una \overline{S} en bajo y una \overline{W} en bajo.
8. $t_{h(A)}$ se mide desde la primera S o W que sube a nivel alto, hasta el final del ciclo de escritura.
9. Durante este periodo, las terminales de E/S se encuentran en el estado de salida, de manera que no deben aplicarse las señales de entrada de la fase opuesta a las salidas.
10. Si la transición S en bajo ocurre de manera simultánea con las transiciones \overline{W} en bajo o después de la transición \overline{W} , la salida permanece en estado de alta impedancia.
11. G está en nivel bajo continuo ($G = V_{IL}$).
12. Si S está en nivel bajo durante este periodo, las terminales de E/S están en el estado de salida. No deben aplicarse las señales de entrada de datos de la fase opuesta a las salidas.
13. La transición se mide a ± 200 mV del voltaje en estado estable.
14. Si la transición S en bajo ocurre antes de la transición \overline{W} en bajo, entonces no deben aplicarse las señales de entrada de datos de la fase opuesta a las salidas para la duración de $t_{dis(W)}$ después de la transición \overline{W} en bajo.

FIGURA 10-6 Diagrama de terminales de la RAM estática 62256 de 32 K × 8.

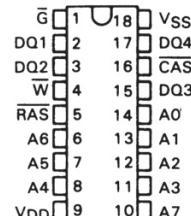


FUNCIÓN DE LAS TERMINALES

A ₀ - A ₁₄	Direcciones
IO ₀ - IO ₇	Conexiones de datos
CS	Selección de chip
OE	Habilitación de salida
WE	Habilitación de escritura
V _{CC}	Suministro de +5 V
GND	Tierra

FIGURA 10-7 Diagrama de terminales de la RAM dinámica (DRAM) TMS4464 de 64 K × 4. (Cortesía de Texas Instruments Incorporated.)

TMS4464...ENCAPSULADO JL O NL
(VISTA SUPERIOR)



(a)

NOMENCLATURA DE TERMINALES

A ₀ -A ₇	Entradas de dirección
CAS	Estrobo de dirección de columna
DQ1-DQ4	Datos de entrada/Datos de salida
G	Habilitación de salida
RAS	Estrobo de dirección de fila
V _{DD}	Suministro de +5 V
V _{SS}	Tierra
W	Habilitación de escritura

(b)

de dirección en donde debería contener 16; el número requerido para direccionar 64 K posiciones de memoria. La única forma en que pueden forzarse 16 bits de dirección en ocho terminales de dirección es en dos incrementos de 8 bits. Esta operación requiere dos terminales especiales: **estrobo de dirección de columna (CAS)** y **estrobo de dirección de fila (RAS)**. En primer lugar, los bits A₀-A₇ se colocan en las terminales de dirección y se aplican por medio de estrobo en un cierre de fila interno mediante RAS como la dirección de fila. A continuación, los bits A₈-A₁₅ se colocan en las mismas ocho entradas

de dirección y se aplican por medio de estrobo a un cierre de columna interno mediante $\overline{\text{CAS}}$ como la dirección de columna (en la figura 10-8 podrá ver esta sincronización). La dirección de 16 bits que se guarda en estos cierres internos direcciona el contenido de una de las posiciones de memoria de 4 bits. Observe que $\overline{\text{CAS}}$ también realiza la función de la entrada de selección de chip para la DRAM.

La figura 10-9 muestra un conjunto de multiplexores que se utilizan para aplicar mediante estrobo las direcciones de columna y de fila en las ocho terminales de dirección, en un par de DRAMs TMS4464. Aquí, la señal $\overline{\text{RAS}}$ no sólo aplica por medio de estrobo la dirección de fila en las DRAMs,

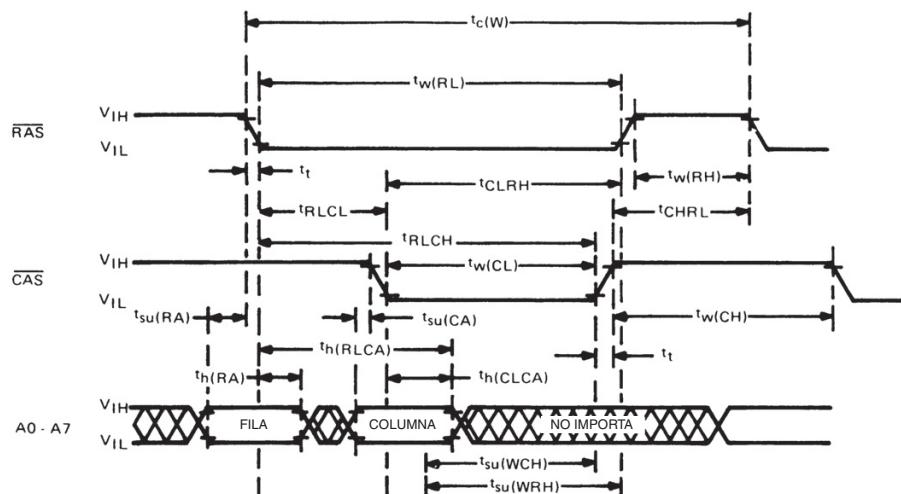


FIGURA 10-8 $\overline{\text{RAS}}$, $\overline{\text{CAS}}$ y la sincronización de las entradas de dirección para la DRAM TMS4464. (Cortesía de Texas Instruments Incorporated.)

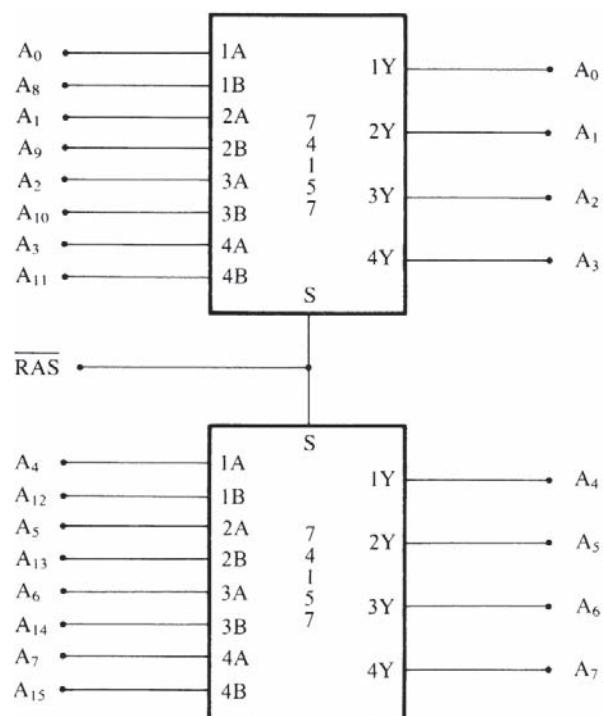
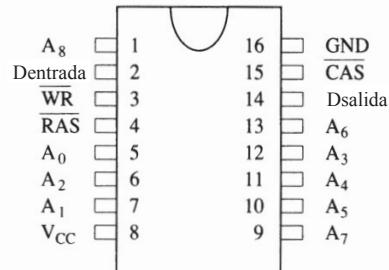


FIGURA 10-9 Multiplexor de dirección para la DRAM TMS4464.

FIGURA 10-10 La RAM dinámica 41256 organizada como un dispositivo de memoria de $256\text{ K} \times 1$.



FUNCIONES DE LAS TERMINALES

A ₀ - A ₈	Direcciones
Dentrada	Datos de entrada
Dsalida	Datos de salida
<u>CAS</u>	Estrobo de dirección de columna
<u>RAS</u>	Estrobo de dirección de fila
<u>WR</u>	Habilitación de escritura
V _{CC}	Suministro de +5 V
GND	Tierra

sino que también selecciona qué parte de la dirección se aplica a las entradas de dirección. Esto es posible gracias al tiempo de retraso de larga propagación de los multiplexores. Cuando RAS es un 1 lógico, las entradas B se conectan a las salidas Y de los multiplexores; cuando la entrada RAS se vuelve un 0 lógico, las entradas A se conectan a las salidas Y. Como el cierre de dirección de fila interno se dispara en los bordes, captura la dirección de fila antes de que la dirección en las entradas cambie a la dirección de columna. En la sección 10-6 proporcionaremos más detalles sobre la DRAM y la creación de interfaces para este tipo de memoria.

Al igual que con la SRAM, la terminal R/W escribe datos en la DRAM cuando es un 0 lógico, pero no hay una terminal de habilitación o que tenga la etiqueta G. Tampoco hay entrada S (selección) para la DRAM. Como se mencionó antes, la entrada CAS selecciona la DRAM. Si está seleccionada, se escribe en la DRAM cuando R/W = 0 y se lee cuando R/W = 1.

La figura 10-10 muestra el diagrama de terminales de la RAM dinámica 41256. Este dispositivo está organizado como una memoria de $256\text{ K} \times 1$ y requiere tan sólo 70 ns para acceder a los datos.

Las DRAMs más recientes son más grandes y están organizadas como memorias de $16\text{ M} \times 1$, de $256\text{ M} \times 1$ y de $512\text{ M} \times 1$. Se tiene previsto lanzar al mercado una memoria de $1\text{ G} \times 1$, la cual se encuentra en etapas de planeación. Por lo general la memoria DRAM se coloca en pequeños tableros de circuitos llamados *SIMMs* (*Módulos de memoria sencillos en línea*). La figura 10-11 muestra los diagramas de terminales de los dos SIMMs más comunes. Las organizaciones más comunes de un SIMM de 30 terminales son de $1\text{ M} \times 8$ o de $1\text{ M} \times 9$ y de $4\text{ M} \times 8$ o de $4\text{ M} \times 9$. La figura 10-11 ilustra una memoria de $4\text{ M} \times 9$. El noveno bit es el bit de paridad. También se muestra un SIMM más reciente de 72 terminales. Por lo común, los SIMMs de 72 terminales se organizan como memorias de $1\text{ M} \times 32$ o de $1\text{ M} \times 36$ (con paridad). Otros tamaños disponibles son $2\text{ M} \times 32$, $4\text{ M} \times 32$, $8\text{ M} \times 32$ y $16\text{ M} \times 32$. Estas memorias también están disponibles con paridad. La figura 10-11 muestra un SIMM de $4\text{ M} \times 36$, que tiene 16 Mbytes de memoria.

En la actualidad hay cada vez más sistemas que utilizan microprocesadores Pentium-Pentium 4. Estos microprocesadores tienen un bus de datos de 64 bits, las cuales descartan el uso de los SIMMs de 8 bits que describimos aquí. Incluso hasta los SIMMs de 72 terminales son inefficientes ya que deben usarse en pares para obtener una conexión de datos de 64 bits. En la actualidad, los DIMMs (*Módulos de memoria duales en línea*) de 64 bits se han convertido en el estándar en la mayoría de los sistemas. La memoria en estos módulos se organiza de manera que haya 64 bits de anchura. Los tamaños comunes disponibles son 16 Mbytes ($2\text{ M} \times 64$), 32 Mbytes ($4\text{ M} \times 64$), 64 Mbytes ($8\text{ M} \times 64$), 128 Mbytes ($16\text{ M} \times 64$), 256 Mbytes ($32\text{ M} \times 64$) y 512 Mbytes ($64\text{ M} \times 64$). En la figura 10-12 se ilustra el diagrama

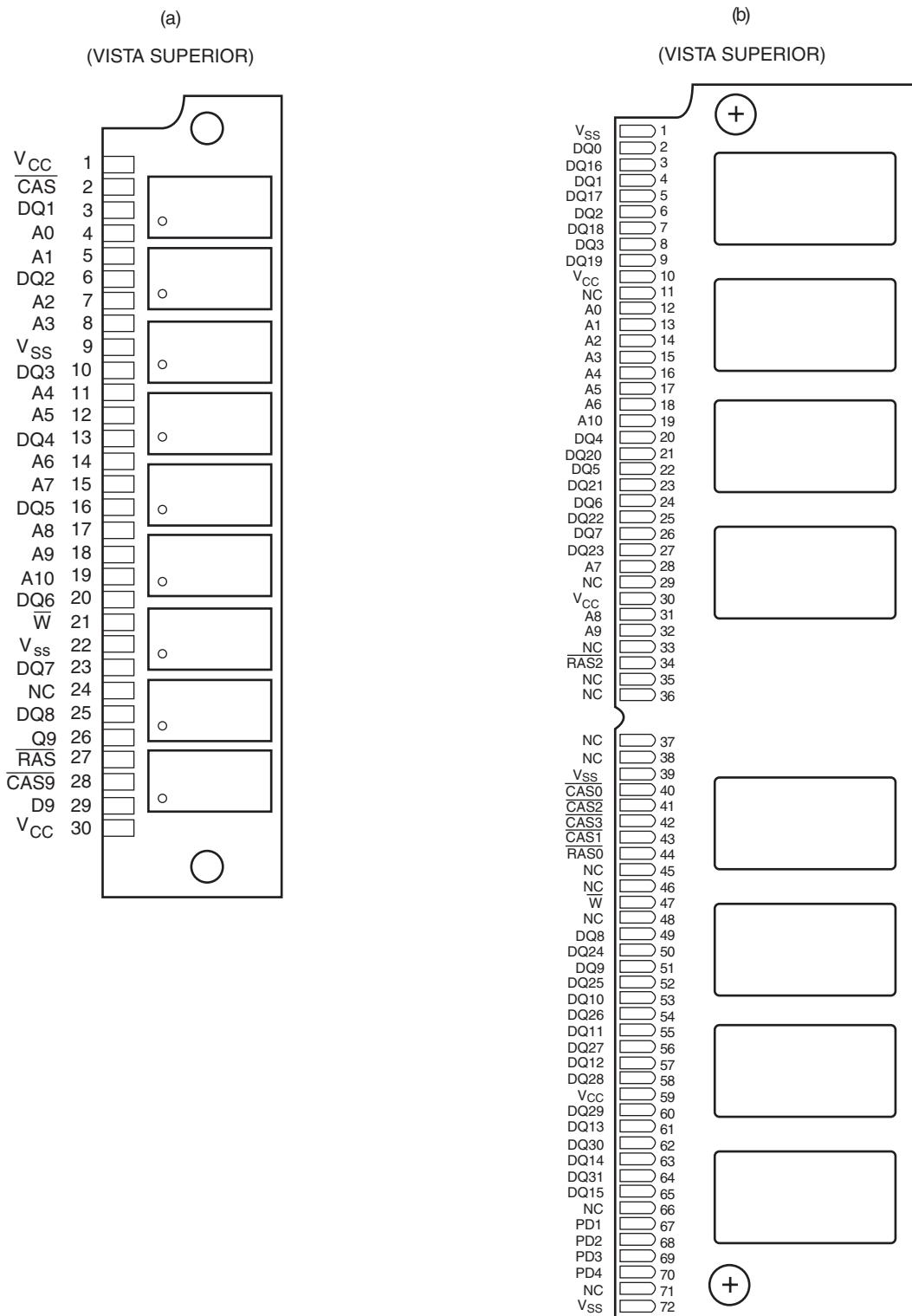


FIGURA 10-11 Los diagramas de terminales de la SIMM de 30 terminales y la de 72 terminales. (a) Un SIMM de 30 terminales organizado como 4 M × 9 y (b) un SIMM de 72 terminales organizado como 4 M × 36.

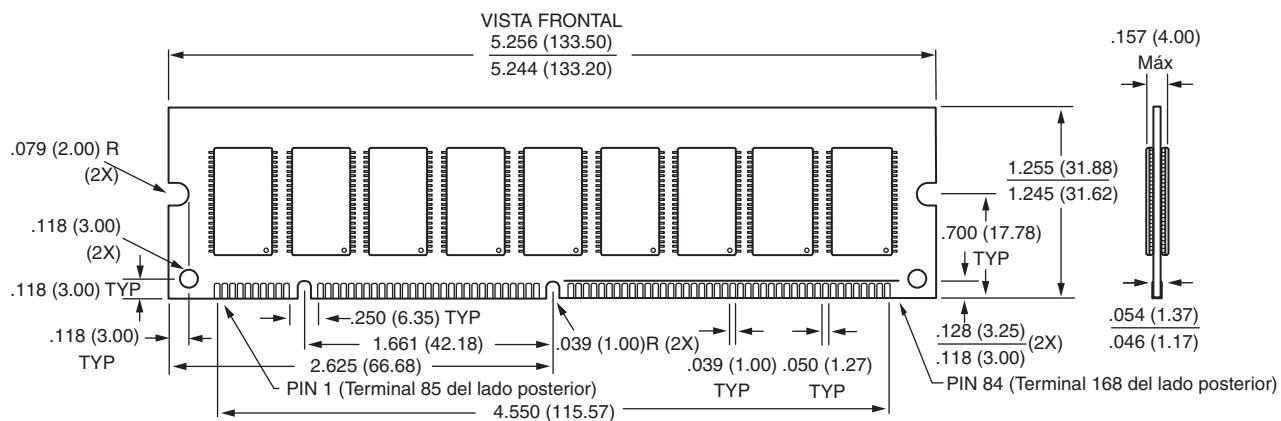


FIGURA 10-12 El diagrama de terminales de un DIMM de 168 terminales.

de terminales del DIMM. Este módulo está disponible en los formatos DRAM, EDO, SDRAM y DDR (velocidad doble de datos), con o sin una EPROM. La EPROM proporciona información al sistema sobre el tamaño y la velocidad del dispositivo de memoria para las aplicaciones de “conectar y usar” (plug-and-play).

Una nueva adición en el mercado de las memorias es el módulo de memoria RIMM de RAMBUS Corporation. Al igual que la SDRAM, el RIMM contiene 168 terminales, pero cada terminal es de dos niveles, con lo que el número total de conexiones se eleva a 336. La SDRAM más veloz que está disponible en la actualidad es la PC-400 o 500 DDR, la cual opera a una velocidad de 4.4 Gbytes por segundo. En comparación, el RIMM de 800 MHz opera a una velocidad de 3.2 Gbytes por segundo y ya no se soporta en algunos sistemas. La RDRAM tuvo una vida bastante corta en el volátil mercado de la computación. El módulo RIMM está organizado como un dispositivo de 32 bits. Esto significa que para utilizarse como memoria en un Pentium 4, debe utilizarse en pares. Intel aclama que el sistema Pentium 4 que utiliza módulos RIMM es 300% más rápido que un Pentium III que utiliza memoria PC-100. De acuerdo con RAMBUS, el RIMM actual de 800 MHz ha incrementado su velocidad hasta 1200 MHz, pero aún no es lo bastante rápido como para acaparar una buena parte del mercado.

En la actualidad, la DRAM más reciente es el dispositivo de memoria DDR (**velocidad doble de datos**). La memoria DDR transfiere datos en cada borde del reloj, con lo que opera al doble de velocidad de la SDRAM. Esto no afecta el tiempo de acceso para la memoria, por lo que aún se requieren muchos estados de espera para operar este tipo de memoria, pero puede ser mucho más veloz que la memoria SDRAM normal, incluyendo a la RDRAM.

10-2

DECODIFICACIÓN DE DIRECCIONES

Para poder conectar un dispositivo de memoria con el microprocesador, es necesario decodificar la dirección que éste envía. El proceso de decodificación hace que la memoria funcione en una sola sección o partición del mapa de memoria. Sin un decodificador de direcciones, sólo podría conectarse un dispositivo a un microprocesador, lo que lo haría prácticamente inútil. En esta sección describiremos unas cuantas de las técnicas comunes de decodificación de direcciones, así como los decodificadores que se incluyen en muchos sistemas.

¿Por qué decodificar la memoria?

Cuando el microprocesador 8088 se compara con la EPROM 2716, se puede ver una diferencia en el número de conexiones de dirección: la EPROM tiene 11 conexiones y el microprocesador tiene 20. Esto significa que el microprocesador envía una dirección de memoria de 20 bits cada vez que lee o

escribe datos. Como la EPROM sólo tiene 11 terminales de dirección, hay una discordancia que debe corregirse. Si sólo se conectan 11 terminales de dirección del 8088 a la memoria, el 8088 sólo verá 2 Kbytes de memoria en vez del 1 Mbyte que “espera” que contenga la memoria. El decodificador corrige esta discordancia mediante la decodificación de las terminales de dirección que no se conectan al componente de memoria.

Decodificador simple de compuerta NAND

Cuando se utiliza la EPROM de $2\text{ K} \times 8$, las conexiones de dirección A₁₀-A₀ del 8088 se conectan en las entradas de dirección A₁₀-A₀ de la EPROM. Las nueve terminales de dirección restantes (A₁₉-A₁₁) se conectan a las entradas de un decodificador de compuerta AND (vea la figura 10-13). El decodificador selecciona la EPROM de una de las secciones de 2 Kbytes del sistema de memoria de 1 Mbyte en el microprocesador 8088.

En este circuito, una sola compuerta NAND decodifica la dirección de memoria. La salida de la compuerta NAND es un 0 lógico siempre que todas las terminales de dirección del 8088 que están unidas a sus entradas (A₁₉-A₁₁) son 1 lógico. La salida de 0 lógico activa a baja del decodificador de compuerta NAND se conecta a la terminal de entrada $\overline{\text{CE}}$ que selecciona (habilita) la EPROM. Recuerde que siempre que $\overline{\text{CE}}$ es un 0 lógico, los datos se leen de la EPROM sólo si $\overline{\text{OE}}$ también es un 0 lógico. La terminal $\overline{\text{OE}}$ se activa mediante la señal RD del 8088 o la señal MRDC (**control de lectura de memoria**) de otros miembros de la familia.

Si la dirección binaria de 20 bits que decodifica la compuerta NAND se escribe de manera que los nueve bits de más a la izquierda tengan el valor de 1 y los 11 bits de más a la derecha tengan el valor de “no importa” (X), el rango de direcciones actual de la EPROM no puede determinarse. (Un valor de “no importa” es un 1 o un 0 lógico, lo que sea apropiado.)

El ejemplo 10-1 muestra cómo se determina el rango de direcciones para esta EPROM si se anotan los bits de dirección (A₁₉-A₁₁) decodificados en forma externa y los bits de dirección decodificados por la EPROM (A₁₀-A₀) con el valor “no importa”. En realidad no nos importa el valor de las terminales de dirección en la EPROM, ya que se decodifican en forma interna. Como lo muestra el ejemplo, los valores “no importa” se escriben primero como 0s para localizar la dirección más baja y luego como 1s para localizar la dirección más alta. El ejemplo 10-1 también muestra estos límites binarios como direcciones hexadecimales. Aquí, la EPROM de 2 K se decodifica en las posiciones de memoria FF800H-FFFFFH. Observe que ésta es una sección de 2 Kbytes de la memoria y también se ubica en la posición de reinicio para el 8086/8088 (FFFF0H), el lugar más probable para una EPROM en un sistema.

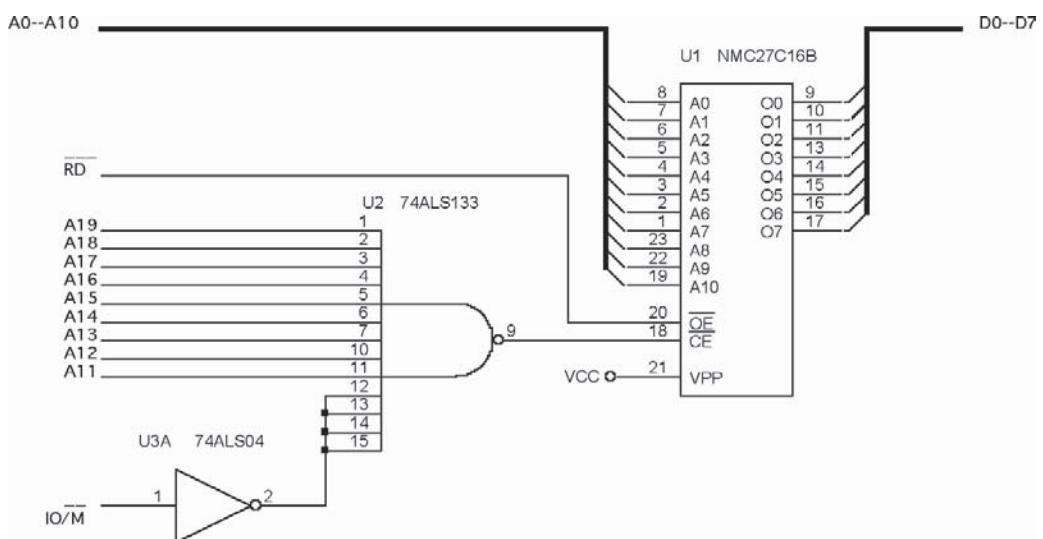


FIGURA 10-13 Un decodificador de compuerta NAND simple que selecciona una EPROM 2716 para la posición de memoria FF800H-FFFFFH.

EJEMPLO 10-1

```

1111 1111 1XXX XXXX XXXX
          o
1111 1111 1000 0000 0000 = FF8000
          a
1111 1111 1111 1111 1111 = FFFFFFFF

```

Aunque este ejemplo sirve para ilustrar la decodificación, las compuertas NAND se utilizan raras veces para decodificar memoria, ya que cada dispositivo de memoria requiere su propio decodificador de compuerta NAND. Debido al excesivo costo del decodificador de compuerta NAND y los inversores que se requieren a menudo, hay que encontrar una alternativa para esta opción.

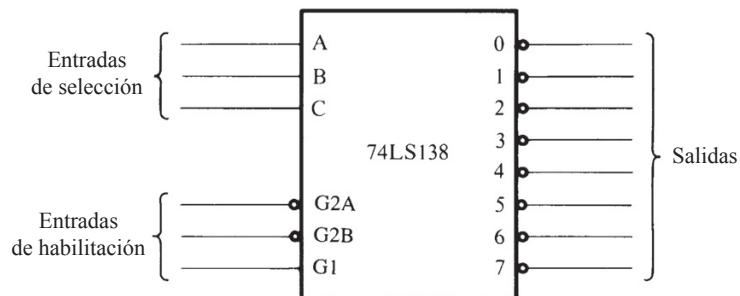
El decodificador de 3 a 8 líneas (74LS138)

Uno de los decodificadores de circuito integrado más comunes que se encuentran en muchos sistemas basados en microprocesador es el decodificador 74LS138 de 3 a 8 líneas. La figura 10-14 muestra este decodificador y su tabla de verdad.

La tabla de verdad muestra que sólo una de las ocho salidas se va al nivel bajo en un momento dado. Para que cualquiera de las salidas del decodificador se vaya al nivel bajo, deben estar activas las tres entradas de habilitación ($\overline{G2A}$, $\overline{G2B}$ y $G1$). Para que se activen, las entradas $\overline{G2A}$ y $\overline{G2B}$ deben estar en nivel bajo (0 lógico) y $G1$ debe estar en nivel alto (1 lógico). Una vez que se habilita el 74LS138, las entradas de dirección (C , B y A) seleccionan la terminal de salida que se va al nivel bajo. ¡Imagine ocho entradas CE de una EPROM conectadas a las ocho salidas del decodificador! Éste es un dispositivo muy poderoso, ya que selecciona ocho dispositivos de memoria distintos al mismo tiempo. Aún en la actualidad este dispositivo sigue teniendo una amplia aplicación.

FIGURA 10-14

El decodificador 74LS138 de 3 a 8 líneas y la tabla de función.



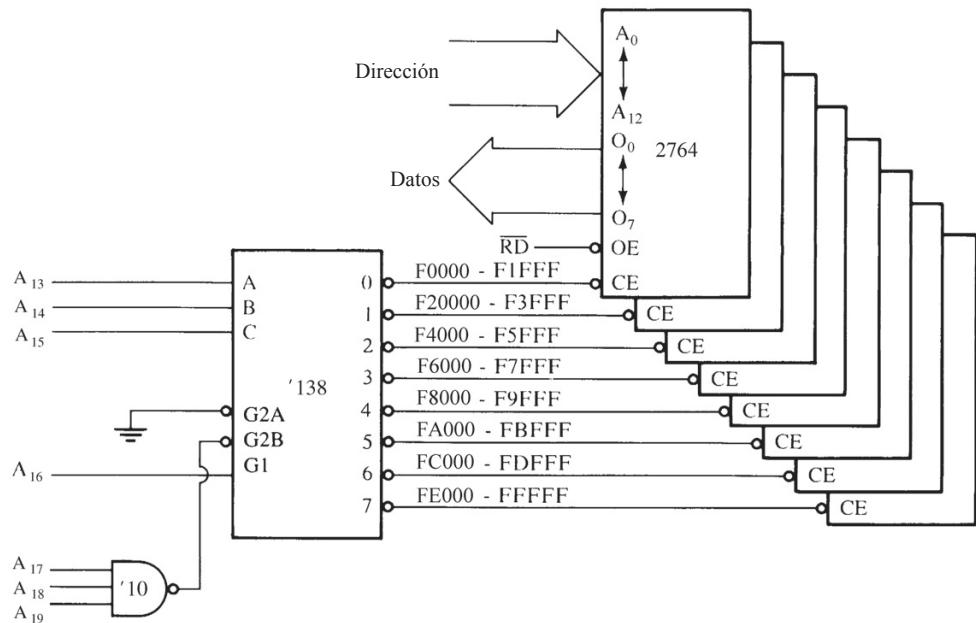


FIGURA 10-15 Un circuito que utiliza ocho EPROMs 2764 para una sección de memoria de 64 K × 8 en un sistema basado en el microprocesador 8088. Las direcciones que se seleccionan en este circuito son F0000H-FFFFFH.

Circuito decodificador de ejemplo. Observe que las salidas del decodificador que se muestra en la figura 10-15 se conectan a ocho dispositivos de memoria EPROM 2764 distintos. Aquí, el decodificador selecciona ocho bloques de 8 Kbytes de memoria, para una capacidad total de memoria de 64 Kbytes. Esta figura también muestra el rango de direcciones de cada dispositivo de memoria y las conexiones comunes a éstos. Observe que todas las conexiones de dirección del 8088 se conectan a este circuito. Observe también que las salidas del decodificador se conectan a las entradas \overline{CE} de las EPROMs y que la señal \overline{RD} del 8088 se conecta a las entradas \overline{OE} de las EPROMs. Esto permite habilitar sólo el EPROM seleccionado y enviar sus datos al microprocesador por medio del bus de datos, siempre que \overline{RD} se vuelva un 0 lógico.

En este circuito se conecta una compuerta NAND de tres entradas a los bits de dirección $A_{19}-A_{17}$. Cuando las tres entradas de dirección están en nivel alto, la salida de esta compuerta NAND se va al nivel bajo y habilita la entrada $G2B$ del 74LS138. La entrada $G1$ se conecta de manera directa a A_{16} . En otras palabras, para poder habilitar este decodificador las cuatro primeras conexiones de dirección ($A_{19}-A_{16}$) deben estar todas en nivel alto.

Las entradas de dirección C, B y A se conectan a las terminales de dirección del microprocesador $A_{15}-A_{13}$. Estas tres entradas de dirección determinan cuál terminal de salida se va al nivel bajo y cuál EPROM se selecciona cuando el 8088 envía una dirección de memoria dentro de este rango al sistema de memoria.

El ejemplo 10-2 muestra cómo se determina el rango de direcciones de todo el decodificador. Observe que el rango comprende las posiciones F0000H-FFFFFH. Este rango abarca 64 Kbytes de la memoria.

EJEMPLO 10-2

1111 XXXX XXXX XXXX XXXX

○

1111 0000 0000 0000 0000 = F0000H
a
1111 1111 1111 1111 1111 = FFFFFH

¿Cómo es posible determinar el rango de direcciones de cada dispositivo de memoria que se conecta a las salidas del decodificador? De nuevo, se anota el patrón de bits; esta vez las entradas de dirección C, B, A contienen el valor “no importa”. El ejemplo 10-3 muestra cómo la salida 0 del decodificador se obliga a ir al nivel bajo para seleccionar la EPROM conectada a esa terminal. Aquí C, B y A se muestran como 0s lógicos.

EJEMPLO 10-3

CBA	
1111 000X XXXX XXXX XXXX	
○	
1111 0000 0000 0000 0000 = F0000H	
a	
1111 0001 1111 1111 1111 = F1FFFH	

Si se requiere el rango de direcciones de la EPROM que está conectada a la salida 1 del decodificador, se determina de la misma forma que la de la salida 0. La única diferencia es que ahora las entradas C, B y A contienen 001 en vez de 000 (vea el ejemplo 10-4). Los rangos de dirección de salida restantes se determinan de la misma forma, mediante la sustitución de la dirección binaria de la terminal de salida en C, B y A.

EJEMPLO 10-4

CBA	
1111 001X XXXX XXXX XXXX	
○	
1111 0010 0000 0000 0000 = F2000H	
a	
1111 0011 1111 1111 1111 = F3FFFH	

El decodificador dual de 2 a 4 líneas (74LS139)

Otro decodificador que se utiliza en algunas aplicaciones es el decodificador 74LS139 dual de 2 a 4 líneas. La figura 10-16 muestra el diagrama de terminales y la tabla de verdad para este decodificador. El 74LS139 contiene dos decodificadores separados de 2 a 4 líneas, cada uno con sus propias conexiones de dirección, habilitación y salida.

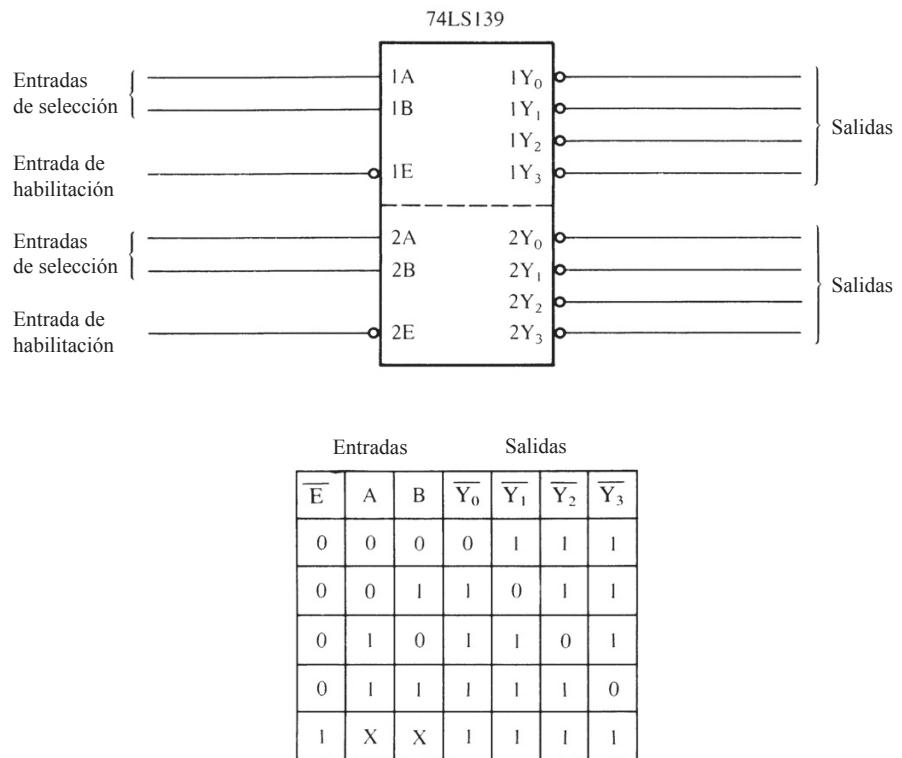
En la figura 10-17 aparece un decodificador más complicado que utiliza el decodificador 74LS139. Este circuito utiliza una EPROM de $128\text{ K} \times 8$ (271000) y una SRAM de $128\text{ K} \times 8$ (621000). La EPROM se decodifica en las posiciones de memoria E0000H-FFFFFH y la SRAM se decodifica en las direcciones 00000H-1FFFFH. Esto es bastante común en un pequeño sistema integrado, en donde la EPROM se encuentra en la parte superior del espacio de memoria y la SRAM en la parte inferior.

La salida $\overline{Y_0}$ del decodificador U1A activa la SRAM siempre que los bits de dirección A_{17} y A_{18} son 0 lógico si la señal IO/\overline{M} es un 0 lógico y la línea de dirección A_{19} es un 0 lógico. Esto selecciona la SRAM para cualquier dirección entre 00000H y 1FFFFH. El segundo decodificador (U1B) es un poco más complicado, ya que la compuerta NAND (U4B) selecciona el decodificador cuando IO/\overline{M} es un 0 lógico, siempre que A_{19} sea un 1 lógico. Esto selecciona la EPROM para las direcciones E0000H hasta la FFFFFH.

Decodificadores PLD programables

En esta sección explicaremos el uso del dispositivo lógico programable (PLD) como un decodificador. Existen tres dispositivos SPLD (**PLD simple**) que funcionan de la misma forma pero que tienen distintos nombres: PLA (**arreglo lógico programable**), PAL (**lógica de arreglo programable**) y GAL (**lógica de arreglo con compuerta**). Aunque estos dispositivos han existido desde mediados de la década de 1970, comenzaron su aparición en el sistema de memoria y en los diseños digitales a principios de

FIGURA 10-16 El diagrama de terminales y la tabla de verdad del decodificador 74LS139 dual de 2 a 4 líneas.



la década de 1990. El PAL y el PLA se programan mediante fusibles, de igual forma que la PROM, y algunos dispositivos PLD pueden borrarse, de igual forma que las EPROMs. En esencia, los tres dispositivos son arreglos de elementos lógicos programables.

También hay otros PLDs disponibles, como los CPLDs (**dispositivos de lógica compleja programables**), los FPGAs (**arreglos de compuerta programables en el campo**) y los FPICs (**interconexión programable en el campo**). Estos tipos de PLDs son mucho más complejos que los SPLDs que se utilizan con más frecuencia para diseñar un sistema completo. Si el asunto en cuestión es decodificar direcciones entonces se utiliza el SPLD, y si el asunto es sobre un sistema completo entonces se utilizan el CPLD o el FPIC para implementar el diseño. A estos dispositivos también se les conoce como ASIC (**circuito integrado de aplicación específica**).

Arreglos lógicos programables combinatorios. Uno de los dos tipos básicos de PAL es el arreglo lógico programable combinatorio. La estructura interna de este dispositivo es como un arreglo programable de circuitos lógicos combinatorios. La figura 10-18 muestra la estructura interna del PAL16L8 que está construido con una lógica de compuertas AND/OR. Este dispositivo que representa a un PLD tiene 10 entradas fijas, dos salidas fijas y seis terminales que pueden programarse como entradas o salidas. Cada señal de salida se genera mediante una compuerta OR de siete entradas que tiene una compuerta AND conectada a cada entrada. Las salidas de las compuertas OR pasan a través de un inversor de tres estados que define cada salida como una función AND/NOR. En un principio, todos los fusibles conectan a todas las conexiones verticales/horizontales que se muestran en la figura 10-18. La programación se lleva a cabo mediante el proceso de quemar fusibles para conectar varias entradas al arreglo de compuertas OR. La función AND cableada se lleva a cabo en cada conexión de entrada, lo cual nos permite tener un término de producto de hasta 16 entradas. Una expresión lógica que utilice el PAL16L8 puede tener hasta siete términos de producto con hasta 16 entradas con la función NOR aplicada entre ellas para generar la expresión de salida. Este dispositivo es ideal como decodificador de direcciones de memoria debido a su estructura. También es ideal debido a que las salidas son activadas a baja.

Por fortuna, no tenemos que seleccionar los fusibles por número para la programación, como era costumbre cuando se introdujo este dispositivo por primera vez. Un PAL se programa mediante un paquete de software tal como PALASM, el programa ensamblador del PAL. El diseño de los PLDs más

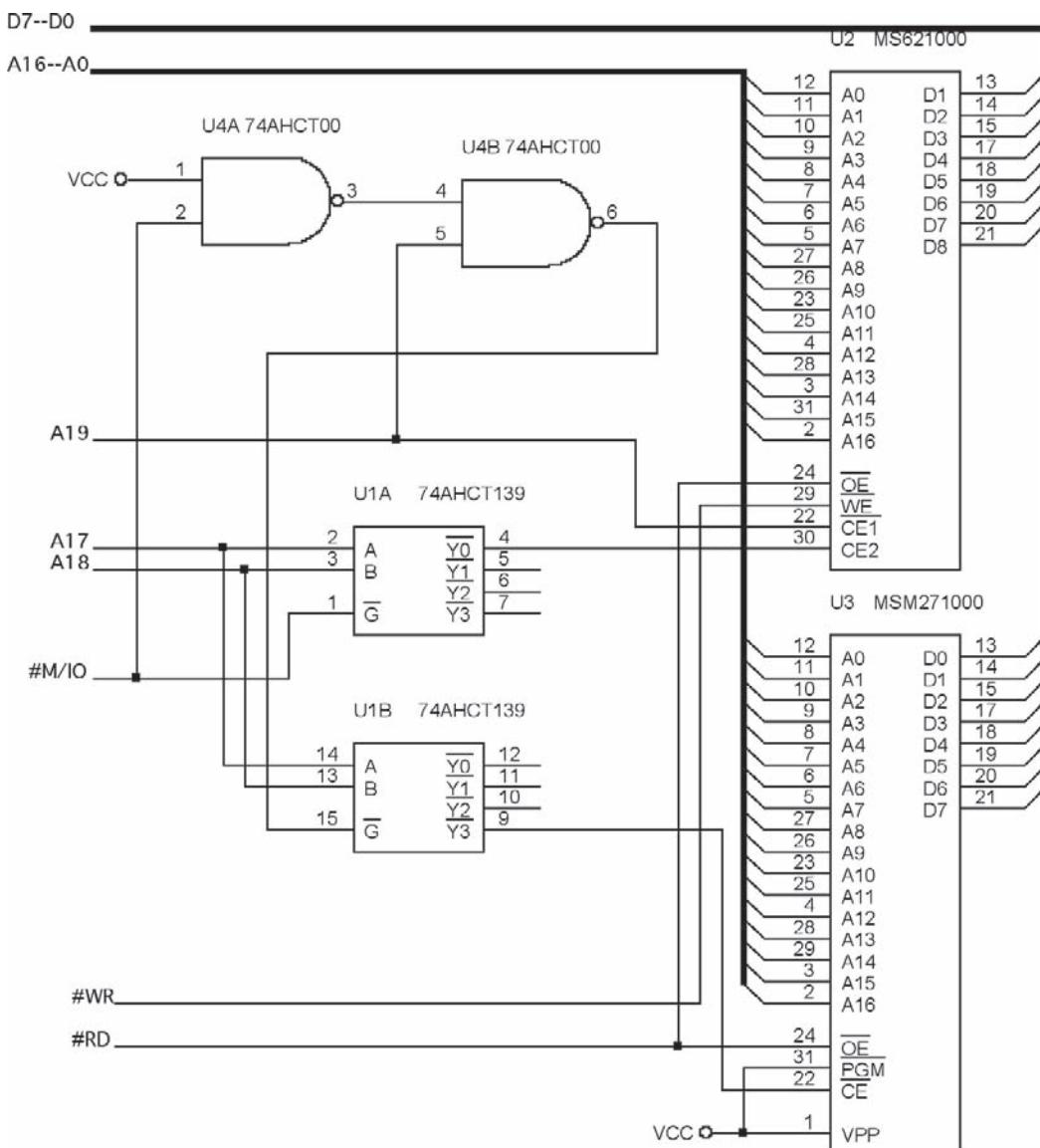


FIGURA 10-17 Ejemplo de un sistema de memoria construido con un 74HCT139.

recientes se realiza mediante el uso de HDL (**lenguaje de descripción de hardware**) o VHDL (**HDL verilog**). El lenguaje VHDL y su sintaxis son el estándar industrial actual para programar dispositivos PLD. El ejemplo 10-5 muestra un programa que decodifica las mismas áreas de memoria que se muestran en la figura 10-17. Este programa se desarrolló mediante el uso de un editor de texto tal como EDIT, disponible en el sistema Microsoft DOS versión 7.1 con XP, o el Bloc de Notas en Windows XP. Este programa también puede desarrollarse mediante el uso de un editor que se incluye en cualquiera de los diversos paquetes de programación para PLDs. Varios editores tratan de facilitar la tarea de definir las terminales, pero creemos que es más fácil utilizar el Bloc de notas y el listado como se muestra.

EJEMPLO 10-5

-- código VHDL para el decodificador de la figura 10-17

```
library ieee;
use ieee.std_logic_1164.all;
```

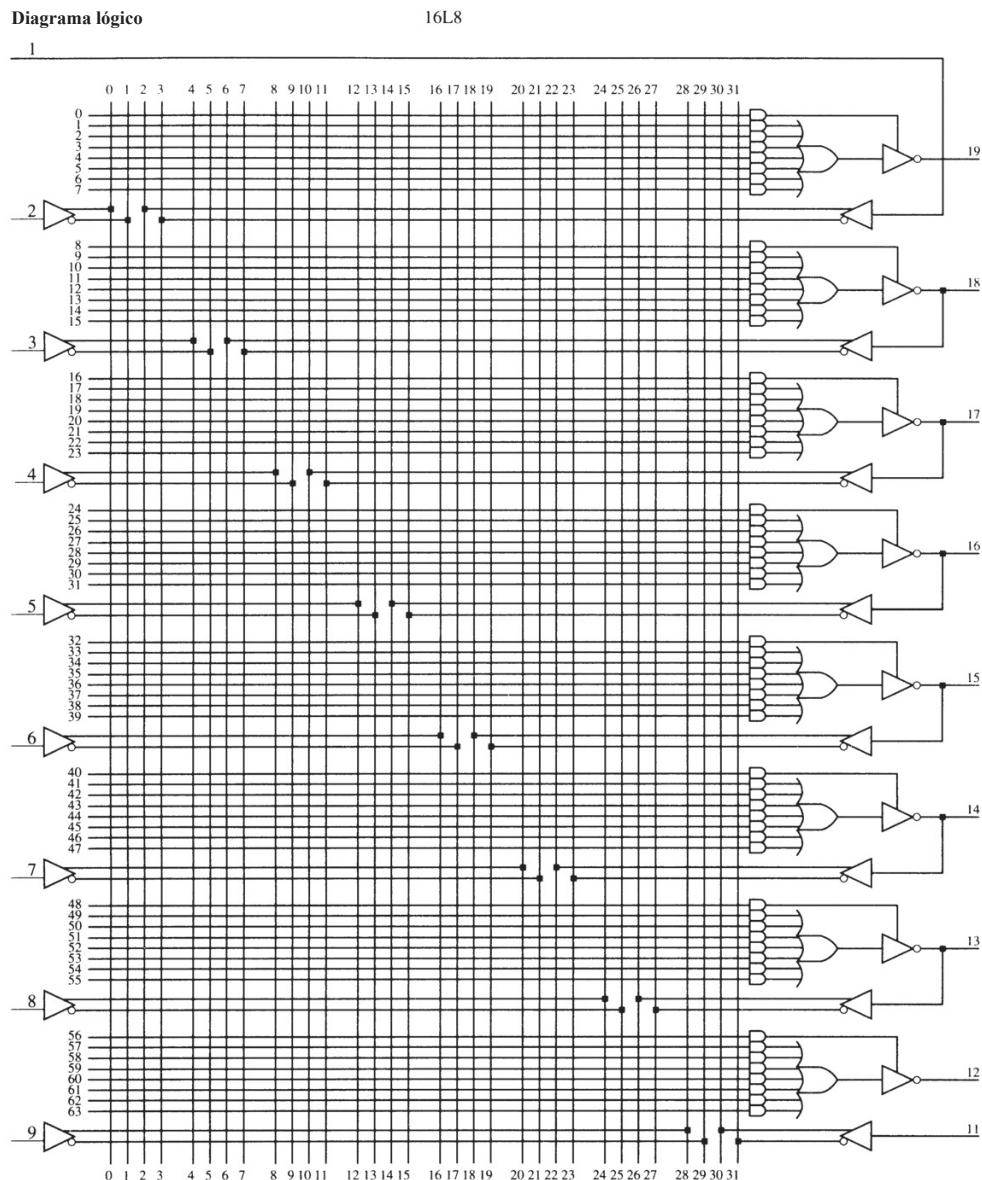


FIGURA 10-18 El PAL 16L8. (Copyright Advanced Micro Devices, Inc., 1998. Reimpreso con permiso del propietario de los derechos de autor. Todos los derechos reservados.)

```

entity DECODIFICADOR_10_17 is
port (
    A19, A18, A17, MIO: in STD_LOGIC;
    ROM, RAM, AX19: out STD_LOGIC;
);
end;

architecture V1 if DECODIFICADOR_10_17 is
begin
    ROM  <= A19 or A18 or A17 or MIO;
    RAM  <= A18 and A17 and (not MIO);
    AX19 <= not A19;
end V1;

```

Los comentarios en la programación con VHDL empiezan con un par de signos negativos, como se muestra en la primera línea del código VHDL en el ejemplo 10-5. Las instrucciones library y use especifican la biblioteca estándar IEEE que utiliza lógica estándar. La instrucción entity nombra el módulo de VHDL; en este caso se llama DECODIFICADOR_10_17. Las instrucciones port definen las terminales de entrada (in), de salida (out) y de entrada-salida (in-out) que se utilizan en las ecuaciones para la expresión lógica, la cual aparece en el bloque begin. A₁₉, A₁₈, A₁₇ y MIO (esta señal no puede definirse como IO/M, por lo que se le asignó el nombre MIO) se definen como terminales de entrada; ROM y RAM son las terminales de salida para la conexión a las terminales CS en los dispositivos de memoria. La instrucción architecture sólo hace referencia a la versión (V₁) de este diseño. Por último, las ecuaciones para el diseño se colocan en el bloque begin. Cada terminal de entrada tiene su propia ecuación. La palabra clave not se utiliza para la inversión lógica y la palabra clave and se utiliza para la operación and lógica. En este caso, la ecuación ROM hace que la terminal ROM sea un cero lógico sólo cuando A₁₉, A₁₈, A₁₇ y MIO sean todos ceros lógicos (00000H-1FFFFH). La ecuación RAM hace que la terminal RAM se convierta en un cero lógico cuando A₁₈ y A₁₇ sean uno al mismo tiempo que MIO sea un cero lógico. A₁₉ se conecta a la terminal CE2 activa en alta, después de invertirse mediante el PLD. La RAM se selecciona para las direcciones 60000H-7FFFFH. En la figura 10-19 podrá ver la realización del PLD del ejemplo 10-5.

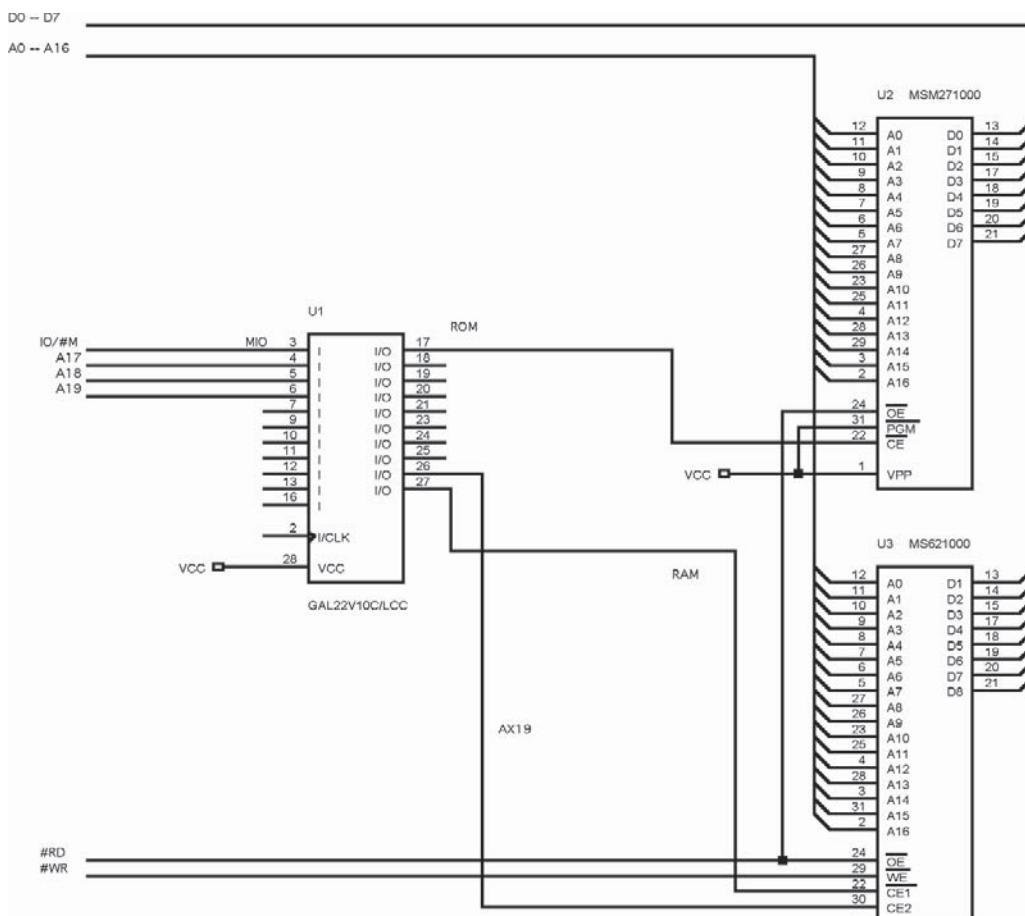


FIGURA 10-19 Una interfaz RAM y ROM que utiliza un dispositivo lógico programable.

10-3**INTERFAZ DE MEMORIA DEL 8088 Y 80188 (8 BITS)**

Este libro contiene secciones separadas sobre el uso de interfaces de memoria para los microprocesadores 8088 y 80188 con sus buses de datos de 8 bits; para los microprocesadores 8086, 80186, 80286 y 80386SX con sus buses de datos de 16 bits; para los microprocesadores 80386DX y 80486 con sus buses de datos de 32 bits; y para los procesadores Pentium-Pentium 4 con sus buses de datos de 64 bits. Se proporcionan secciones separadas debido a que los métodos que se utilizan para direccionar la memoria son un poco distintos en los microprocesadores que contienen buses de datos con distintas anchuras. Es conveniente que los ingenieros o técnicos de hardware que deseen ampliar su experiencia en el uso de interfaces de memoria de 16 bits, 32 bits y 64 bits lean todas las secciones. Esta sección es mucho más completa que las secciones sobre la interfaz de memoria de 16, 32 y 64 bits, en las que se cubre material que no se explica en la sección para los microprocesadores 8088/80188.

En esta sección examinaremos la interfaz de memoria para la RAM y la ROM; también explicaremos el código de corrección de errores (ECC), que aún está disponible en la actualidad para los diseñadores de sistemas de memoria. Muchos sistemas de computadoras personales no utilizan el ECC debido al costo, pero los equipos empresariales sí lo hacen.

Interfaz básica de memoria del 8088/80188

Los microprocesadores 8088 y 80188 tienen un bus de datos de 8 bits, lo que los hace ideales para conectar los dispositivos comunes de memoria de 8 bits disponibles en la actualidad. El tamaño de la memoria de 8 bits hace que el 8088 (y especialmente el 80188) sea ideal como un controlador simple. No obstante, para que el 8088/80188 funcione de manera apropiada con la memoria, el sistema de memoria debe decodificar la dirección para seleccionar un componente de la memoria. También debe utilizar las señales \overline{RD} , \overline{WR} e IO/\overline{M} que tiene el 8088/80188 para controlar el sistema de memoria.

En esta sección se utiliza la configuración en modo mínimo; en esencia es igual que el sistema en modo máximo para la interfaz de memoria. La principal diferencia es que, en modo máximo la señal IO/\overline{M} se combina con RD para generar la señal $MRDC$, y la señal IO/\overline{M} se combina con \overline{WR} para generar la señal $MWTC$. Las señales de control en modo máximo se desarrollan dentro del controlador de bus 8288. En modo mínimo, la memoria ve al 8088 o al 80188 como un dispositivo con 20 conexiones de dirección ($A_{19}-A_0$), ocho conexiones de bus de datos (AD_7-AD_0) y las señales de control IO/\overline{M} , \overline{RD} y \overline{WR} .

Interfaz entre la EPROM y el 8088. Esta sección le parecerá muy similar a la sección 10-2 sobre los decodificadores. La única diferencia es que en esta sección hablaremos sobre los estados de espera y el uso de la señal IO/\overline{M} para habilitar el decodificador.

La figura 10-20 muestra un microprocesador 8088/80188 conectado a tres EPROMs 27256, que son dispositivos de memoria de $32\text{ K} \times 8$. La EPROM 27256 tiene una entrada de dirección más (A_{15}) que la 27128, además del doble de la memoria. El decodificador 74HCT138 en esta ilustración decodifica tres bloques de memoria de $32\text{ K} \times 8$ para obtener un total de $96\text{ K} \times 8$ bits del espacio físico de dirección para el 8088/80188.

El decodificador (74HCT138) se conecta de manera un poco distinta de lo que podría esperarse, ya que la versión más lenta de este tipo de EPROM tiene un tiempo de acceso a memoria de 450 ns. En el capítulo 9 vimos que cuando el 8088 se opera con un reloj de 5 MHz, permite hasta 460 ns a la memoria para acceder a los datos. Debido al retraso de tiempo adicional del decodificador (8 ns), es imposible que esta memoria funcione dentro del rango de 460 ns. Para poder corregir este problema, puede usarse la salida de la compuerta NAND para generar una señal que habilite el decodificador y una señal para el generador de estados de espera, que vimos en el capítulo 9. (El 80188 puede insertar en forma interna de 0 a 15 estados de espera sin necesidad de hardware externo adicional, por lo que no requiere esta compuerta NAND.) Si se inserta un estado de espera cada vez que se accede a esta sección de la memoria, el 8088 permitirá 660 ns para que la EPROM acceda a los datos. Recuerde que un estado de espera adicional agrega 200 ns (1 ciclo de reloj) al tiempo de acceso. Los 660 ns son el tiempo suficiente como para que un componente de memoria de 450 ns acceda a los datos, incluso con los retrasos introducidos por el decodificador y por cualquier búfer que se agregue al bus de datos. En este sistema, los estados de espera se insertan en las posiciones de memoria C0000H-FFFFFH. Si esto produce un problema, puede agregarse una compuerta OR de tres entradas a las tres salidas del decodificador para generar una señal de espera sólo para las direcciones actuales de este sistema (E8000H-FFFFFH).

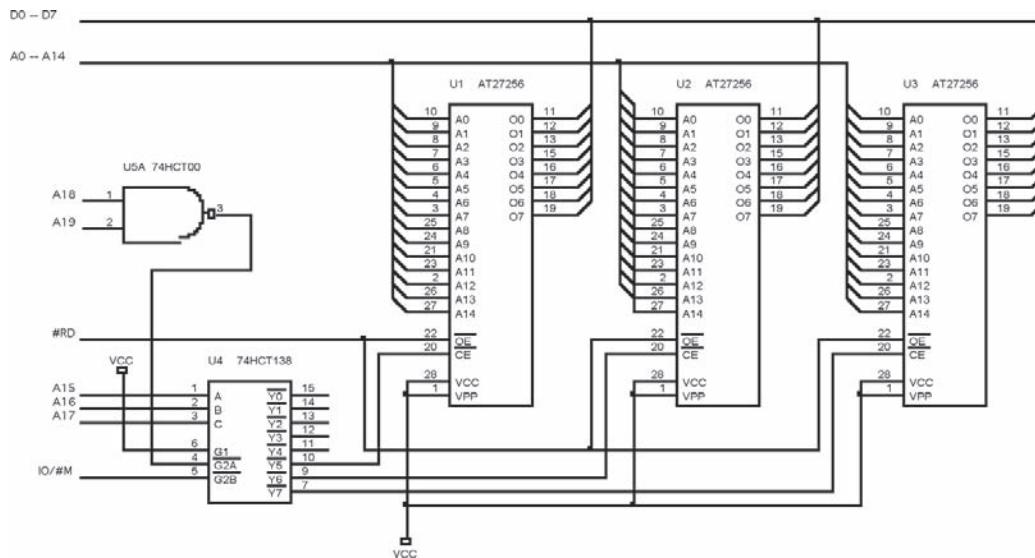


FIGURA 10-20 Interfaz entre tres EPROMs 27256 y el microprocesador 8088.

Observe que el decodificador se selecciona para un rango de direcciones de memoria que empieza en la posición E8000H y continúa hasta la posición FFFFFH; los 96 Kbytes superiores de memoria. Esta sección de memoria es una EPROM ya que FFFF0H es donde el 8088 empieza a ejecutar instrucciones después de un reinicio de hardware. Por lo general, a la posición FFFF0H se le llama la **posición de arranque en frío**. El software almacenado en esta sección de memoria contiene una instrucción JMP en la posición FFFF0H que provoca un salto a la posición E8000H, de manera que pueda ejecutarse el resto del programa. En este circuito, U₁ se decodifica en las direcciones F8000H-FFFFFH, U₂ se decodifica en F0000H-F7FFFF y U₃ se decodifica en E8000H-EFFFFH.

Interfaz entre la RAM y el 8088. Es un poco más fácil crear una interfaz con la RAM que con la EPROM, ya que la mayoría de los componentes de la memoria RAM no requieren estados de espera. Una sección ideal de la memoria para la RAM es la parte final, que contiene los vectores para las interrupciones. Por lo general, los vectores de interrupción (que describiremos con más detalle en el capítulo 12) se modifican mediante paquetes de software, por lo que es muy importante codificar esta sección de memoria con RAM.

La figura 10-21 muestra 16 RAMs estáticas 62256 de 32 K × 8 conectadas al 8088, con la posición de memoria inicial 00000H. Este tablero de circuitos utiliza dos decodificadores para seleccionar los 16 componentes de memoria RAM y un tercer decodificador para seleccionar los otros dos decodificadores, de manera que se obtengan las secciones apropiadas de memoria. Diecisésis RAMs de 32 K llenan la memoria, desde la posición 00000H hasta la posición 7FFFFFFH, para un total de 512 Kbytes de memoria.

El primer decodificador (U₄) en este circuito selecciona a los otros dos decodificadores. El decodificador U₃ se selecciona con una dirección que empieza con 00, y el decodificador U₉ se selecciona con una dirección que empieza con 01. Observe que hay terminales adicionales en la salida del decodificador U₄ para una expansión a futuro. Estas terminales permiten más bloques de RAM de 256 K × 8, para un total de 1 M × 8, con sólo agregar la RAM y los decodificadores secundarios adicionales.

En el circuito de la figura 10-21 también podemos observar que todas las entradas de dirección para esta sección de memoria utilizan búfer, al igual que las conexiones del bus de datos y las señales de control RD y WR. El uso de búferes es importante cuando aparecen muchos dispositivos en un solo tablero o en un sistema individual. Suponga que se conectan otros tres tableros como éste en un sistema. Sin los búferes en cada tablero, la carga en los buses de direcciones, de datos y de control del sistema sería suficiente como para evitar la operación apropiada. (Una carga excesiva provoca que la salida de 0 lógico se eleve por encima del máximo de 0.8 V que se permite en un sistema.) Es común utilizar búferes si se tiene pensado agregar componentes a la memoria en un futuro cercano. Si no se tiene planeado que la memoria crezca en el futuro, entonces tal vez no sean necesarios los búferes.

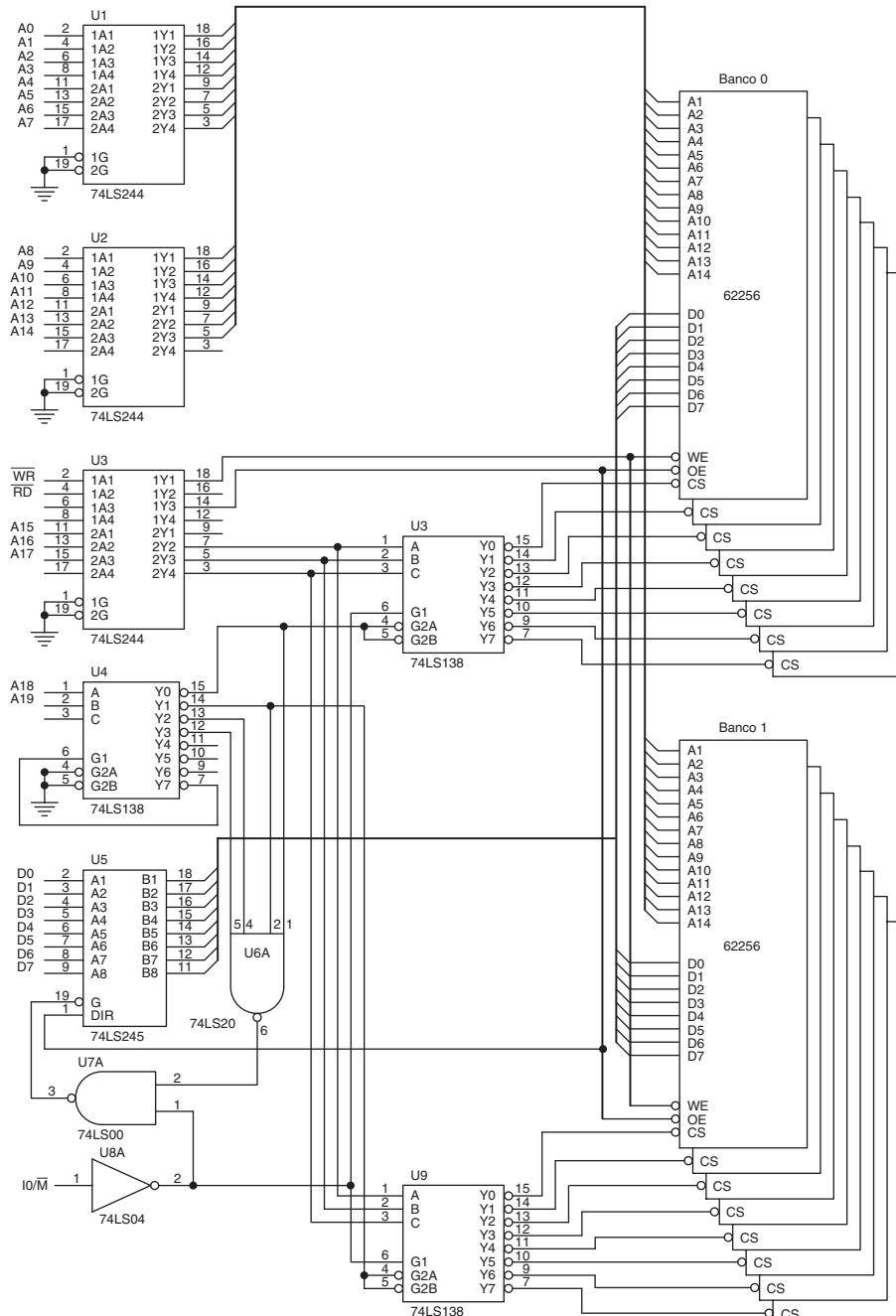


FIGURA 10-21 Un sistema de memoria estática de 512 Kbytes que utiliza 16 SRAMs 62255.

Interfaz para la memoria Flash

La memoria Flash (EEPROM) se está convirtiendo en el lugar común para almacenar información en las tarjetas de vídeo, así como para almacenar el BIOS del sistema en la computadora personal. Incluso se utiliza en reproductores de audio MP3 y en unidades portátiles de USB. También podemos encontrar la memoria Flash en muchas otras aplicaciones, para almacenar información que cambia con poca frecuencia.

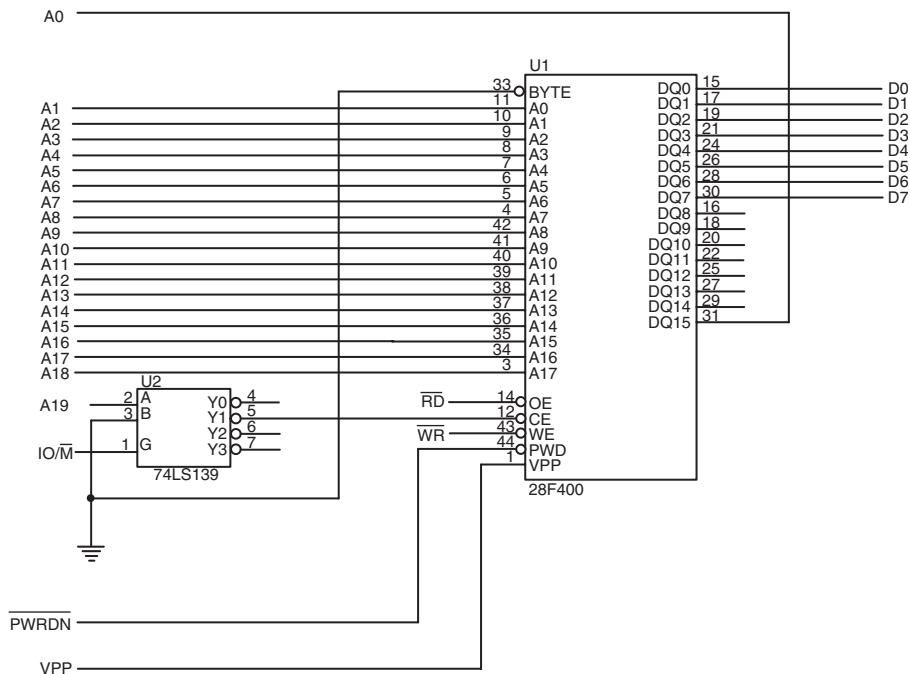


FIGURA 10-22 El dispositivo de memoria Flash 28F400 conectado al microprocesador 8088.

La única diferencia entre un dispositivo de memoria Flash y la SRAM es que el dispositivo de memoria Flash requiere un voltaje de programación de 12 V para borrarla y escribir nuevos datos. Los 12 V pueden obtenerse de la fuente de poder o mediante un convertidor de 5 a 12 V, diseñado para usarse con memoria Flash. Las versiones más recientes de memoria Flash se borran con una señal de 5.0 V (o incluso con una de 3.3 V), por lo que no se necesita un convertidor.

La figura 10-22 muestra el dispositivo de memoria Flash 28F400 de Intel, conectado con el microprocesador 8088. El 28F400 puede usarse como un dispositivo de memoria de 512 K × 8, o de 256 K × 16. Como está conectado con el 8088, su configuración es de 512 K × 8. Observe que las conexiones de control en este dispositivo son idénticas a las de una SRAM: \overline{CE} , \overline{OE} y \overline{WE} . Las únicas terminales nuevas son V_{PP} , que se conecta a 12 V para borrar y para programar; PWD , que selecciona el modo de corte de energía cuando es un 0 lógico y también se utiliza para la programación; y $BYTE$, que selecciona la operación en modo byte (0) o palabra (1). Observe que la terminal DQ_{15} funciona como la entrada de dirección menos significativa cuando se opera en modo byte. Otra diferencia es la cantidad de tiempo que se requiere para realizar una operación de escritura. La SRAM puede efectuar una operación de escritura en tan sólo 10 ns, pero la memoria Flash requiere aproximadamente 0.4 segundos para borrar un byte. En el capítulo 11 hablaremos sobre cómo programar el dispositivo de memoria Flash, junto con los dispositivos de E/S. Este dispositivo de memoria tiene registros internos que se programan mediante el uso de técnicas de E/S que no hemos explicado todavía. En este capítulo nos concentraremos en cómo conectarlo al microprocesador.

Observe en la figura 10-22 que el decodificador seleccionado es el 74LS139, ya que sólo se necesita un decodificador simple para un dispositivo de memoria Flash de este tamaño. El decodificador utiliza la conexión de dirección A_{19} y la señal IO/\bar{M} como entradas. La señal A_{15} selecciona la memoria Flash para las posiciones 80000H hasta FFFFFH, y la señal IO/\bar{M} habilita el decodificador.

Corrección de errores

Los esquemas de corrección de errores se han utilizado durante mucho tiempo, pero los fabricantes de circuitos integrados apenas han empezado a producir circuitos de corrección de errores. Uno de esos circuitos es el 74LS636, un circuito de corrección y detección de errores de 8 bits que corrige cualquier error de lectura de memoria de un bit y que marca cualquier error de dos bits, llamado SECDED

(corrección de error sencillo/corrección de error doble). Este dispositivo se encuentra en sistemas computacionales de alta tecnología debido al costo de implementar un sistema que utilice corrección de errores.

Los sistemas computacionales más recientes utilizan memoria DDR con ECC (código de corrección de errores). El esquema para corregir los errores que podrían ocurrir en estos dispositivos de memoria es idéntico al esquema que describimos en este libro.

El 74LS636 corrige los errores mediante el almacenamiento de cinco bits de paridad con cada byte de datos de memoria. Esto incrementa la cantidad de memoria requerida, pero también proporciona una corrección automática de errores de un solo bit. Si hay error en más de dos bits, tal vez este circuito no lo detecte. Por fortuna esto es raro; además el esfuerzo adicional que se requiere para corregir más de un error de un solo bit es muy costoso y no vale la pena el esfuerzo. Cuando un componente de memoria falla por completo, todos sus bits están en nivel alto, o todos están en nivel bajo. En este caso, el circuito avisa al procesador con una indicación de error en varios bits.

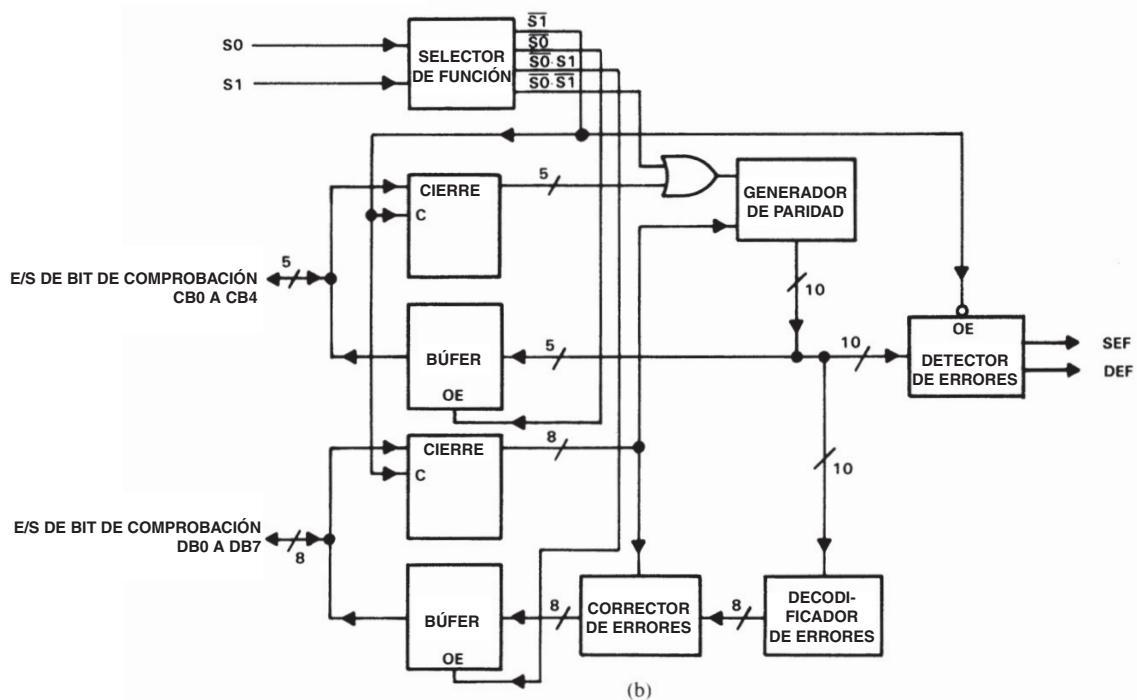
La figura 10-23 representa el diagrama de terminales del 74LS636. Observe que tiene ocho terminales de E/S de datos, cinco terminales de E/S de bit de comprobación, dos entradas de control (SO) y

asignaciones de terminales

PAQUETES J. N.	
1	DEF
2	DB0
3	DB1
4	DB2
5	DB3
6	DB4
7	DB5
8	DB6
9	DB7
10	GND
11	CB4
12	nc
13	CB3
14	CB2
15	CB1
16	CB0
17	SO
18	S1
19	SEF
20	VCC

(a)

diagrama de bloques funcional



(b)

FIGURA 10-23 (a) Las conexiones de las terminales del 74LS636. (b) El diagrama de bloques del 74LS636. (Cortesía de Texas Instruments Incorporated.)

TABLA 10-1 Los bits de control S_0 y S_1 para el 74LS636.

S_1	S_0	Función	SEF	DEF
0	0	Escribe palabra de comprobación	0	0
0	1	Corrige palabra de datos	*	*
1	0	Lee datos	0	0
1	1	Cierra datos	*	*

*Estos niveles se determinan sobre la base del tipo de error.

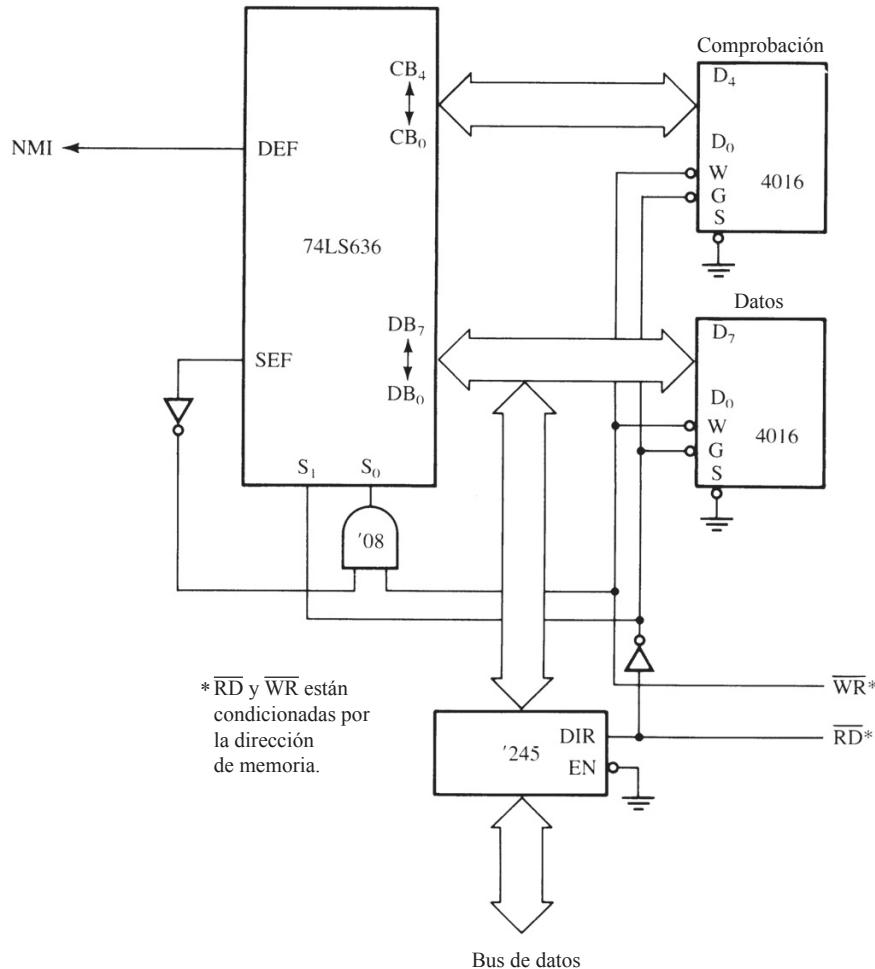
SI) y dos salidas de error: bandera de un solo error (SEF) y bandera de error doble (DEF). Las entradas de control seleccionan el tipo de operación a realizar y se listan en la tabla de verdad de la tabla 10-1.

Cuando se detecta un solo error, el 74LS636 pasa por un ciclo de corrección de errores. Coloca 01 en S_0 y S_1 , con lo que produce un estado de espera y después una lectura después de la corrección del error.

La figura 10-24 muestra un circuito que se utiliza para corregir errores de un solo bit con el 74LS636 y para interrumpir el procesador mediante la terminal NMI, para los errores de bit dobles. Para simplificar la ilustración sólo incluimos una RAM de $2\text{ K} \times 8$ y una segunda RAM de $2\text{ K} \times 8$ para almacenar el código de comprobación de cinco bits.

La conexión de este componente de memoria es distinta a la del ejemplo anterior. Observe que la terminal S o \bar{CS} está aterrizada, y que los búferes del bus de datos controlan el flujo hacia el bus del sistema. Esto es necesario si los datos se van a acceder de la memoria antes de que el estrobo \overline{RD} se vaya a nivel bajo.

FIGURA 10-24 Un circuito de detección y corrección de errores que utiliza el 74LS636.



En el siguiente borde negativo del reloj después de una señal \overline{RD} , el 74LS636 comprueba la bandera de un solo error (SEF) para determinar si ha ocurrido un error. En caso de ser así, un ciclo de corrección se encarga de corregir el defecto de un solo error. Si ocurre un error doble, se genera una petición de interrupción mediante la salida de la bandera de doble error (DEF), la cual está conectada a la terminal NMI del microprocesador.

La memoria DDR con corrección de errores (ECC) moderna en realidad no tiene circuitos lógicos en el tablero que detecten y corrijan errores. Desde el Pentium, el microprocesador incorpora los circuitos lógicos para detectar/corregir errores siempre y cuando la memoria pueda almacenar los ocho bits adicionales que se requieren para almacenar el código ECC. La memoria ECC es de 72 bits si utiliza los ocho bits adicionales para almacenar el código ECC. Si ocurre un error, el microprocesador ejecuta el ciclo de corrección para corregirlo. Algunos dispositivos de memoria, como la memoria Samsung, también realizan una comprobación de errores interna. La ECC de Samsung utiliza tres bytes para comprobar cada 256 bytes de memoria, lo cual es mucho más eficiente. En el sitio Web de Samsung podrá encontrar información adicional sobre el algoritmo ECC de Samsung.

10-4

INTERFAZ DE MEMORIA DEL 8086, 80186, 80286 Y 80386SX (16 BITS)

Los microprocesadores 8086, 80186, 80286 y 80386SX difieren del 8088/80188 en tres aspectos: (1) El bus de datos es de 16 bits en vez de ocho, como en el 8088; (2) la terminal IO/\overline{M} del 8088 se sustituye con la terminal M/\overline{IO} ; y (3) hay una nueva señal de control llamada habilitación de bus superior (BHE). El bit de dirección A_0 o BLE también se utiliza de manera distinta. (Como esta sección se basa en la información que se proporciona en la sección 10-3, es importante que lea primero la sección anterior.) Hay más diferencias entre el 8086/80186 y el 80286/80386SX. Éste contiene un bus de direcciones de 24 bits ($A_{23}-A_0$) en vez del bus de direcciones de 20 bits ($A_{19}-A_0$) del 8086/80186. Este microprocesador contiene una señal M/\overline{IO} , mientras que los microprocesadores 80286 y 80386SX contienen las señales $MRDC$ y $MWTC$ en vez de RD y WR .

Control del bus de 16 bits

El bus de datos de los microprocesadores 8086, 80186, 80286 y 80386SX es del doble de anchura que el bus del 8088/80188. Estos buses de datos más anchos nos presentan un conjunto único de problemas. Los microprocesadores 8086, 80186, 80286 y 80386SX deben ser capaces de escribir datos en cualquier ubicación de 16 bits (o en cualquier ubicación de 8 bits). Esto significa que el bus de datos de 16 bits debe dividirse en dos secciones separadas (**bancos**) que sean de ocho bits de ancho, de manera que el microprocesador pueda escribir en cualquiera de las mitades (8 bits) o en ambas (16 bits). La figura 10-25 muestra los dos bancos de la memoria. Un banco (**banco inferior**) contiene todas las posiciones de memoria con numeración par, y el otro banco (**banco superior**) contiene todas las posiciones de memoria con numeración impar.

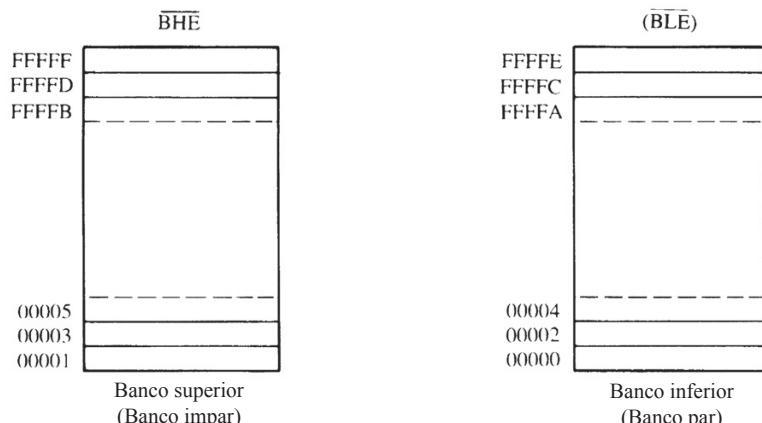
Los microprocesadores 8086, 80186, 80286 y 80386SX utilizan la señal \overline{BHE} (banco superior) y el bit de dirección A_0 o BLE (habilitación de bus inferior) para seleccionar uno o ambos de los bancos de memoria que se van a utilizar para la transferencia de datos. La tabla 10-2 contiene los niveles lógicos en estas dos terminales y el banco o bancos seleccionados.

La selección de los bancos se realiza de dos maneras: (1) se desarrolla una señal de escritura separada para seleccionar una escritura a cada uno de los bancos de la memoria, o (2) se utilizan decodificadores separados para cada banco. Una cuidadosa comparación nos revela que la primera técnica es el método menos costoso para la interfaz de memoria para los microprocesadores 8086, 80186, 80286 y 80386SX. La segunda sólo se utiliza en un sistema que deba obtener el uso más eficiente de la fuente de poder.

Decodificadores de bancos separados. El uso de éstos es a menudo la manera menos efectiva de decodificar direcciones de memoria para los microprocesadores 8086, 80186, 80286 y 80386SX. Este método se utiliza algunas veces, aunque en muchos casos es difícil de entender por qué. Tal vez una razón sea para conservar energía, ya que sólo se habilitan el banco o los bancos seleccionados. Este no

FIGURA 10-25

Los bancos de memoria superior (impar) e inferior (par) de 8 bits para los microprocesadores 8086/80286/80386SX.



Nota: A_0 se designa como \overline{BLE}
(Habilitación de bus en nivel bajo)
en el 80386SX.

es siempre el caso, al igual que con las señales de lectura y escritura de bancos por separado que describiremos más adelante.

La figura 10-26 muestra dos decodificadores 74LS138 que se utilizan para seleccionar componentes de memoria RAM de 64K para el microprocesador 80386SX (dirección de 24 bits). Aquí, el decodificador U_2 tiene la terminal \overline{BLE} (A_0) conectada a $\overline{G2A}$, y el decodificador U_3 tiene la señal BHE conectada a su entrada $G2A$. Como el decodificador no se activa hasta que se activen todas sus entradas de habilitación, el decodificador U_2 se activa sólo para una operación de 16 bits o una operación de 8 bits desde el banco inferior. El decodificador U_3 se activa para una operación de 16 bits o una operación de 8 bits para el banco superior. Estos dos decodificadores y las 16 RAMs de 64 Kbytes que controlan representan un rango de 1 M del sistema de memoria del 80386SX. El decodificador U_1 habilita a U_2 y a U_3 para el rango de direcciones de memoria 000000H-0FFFFFH.

En la figura 10-26 puede observar que la terminal de dirección A_0 no se conecta a la memoria debido a que no existe en el microprocesador 80386SX. Además puede ver que la posición de bit A_1 del bus de direcciones está conectada a la entrada de dirección A_0 de la memoria. Si A_0 o \overline{BLE} estuvieran conectadas a la terminal de dirección A_0 de la memoria, se utilizaría cualquier otra posición de memoria en cada banco. Esto significa que se desperdiciaría la mitad de la memoria si A_0 o \overline{BLE} estuvieran conectadas a A_0 .

Estrobo de escritura en bancos separados. La manera más efectiva de manejar la selección de bancos es mediante el desarrollo de un estrobo de escritura separado para cada banco de memoria. Esta técnica sólo requiere un decodificador para seleccionar una memoria de 16 bits, lo que a menudo ahorra dinero y reduce el número de componentes en un sistema.

¿Por qué no generar también estrobo de lectura separados para cada banco de memoria? Por lo general, esto es innecesario ya que los microprocesadores 8086, 80186, 80286 y 80386SX sólo leen el byte de datos que necesitan en un momento dado de la mitad del bus de datos. Si se presentan siempre secciones de datos de 16 bits en el bus de datos durante una operación de lectura, el microprocesador ignora la sección de 8 bits que no necesita, sin conflictos ni problemas especiales.

TABLA 10-2 Selección de los bancos de memoria mediante el uso de \overline{BHE} y \overline{BLE} (A_0).

\overline{BHE}	\overline{BLE}	Función
0	0	Se habilitan ambos bancos para una transferencia de 16 bits
0	1	Se habilita el banco superior para una transferencia de 8 bits
1	0	Se habilita el banco inferior para una transferencia de 8 bits
1	1	No se habilita ningún banco

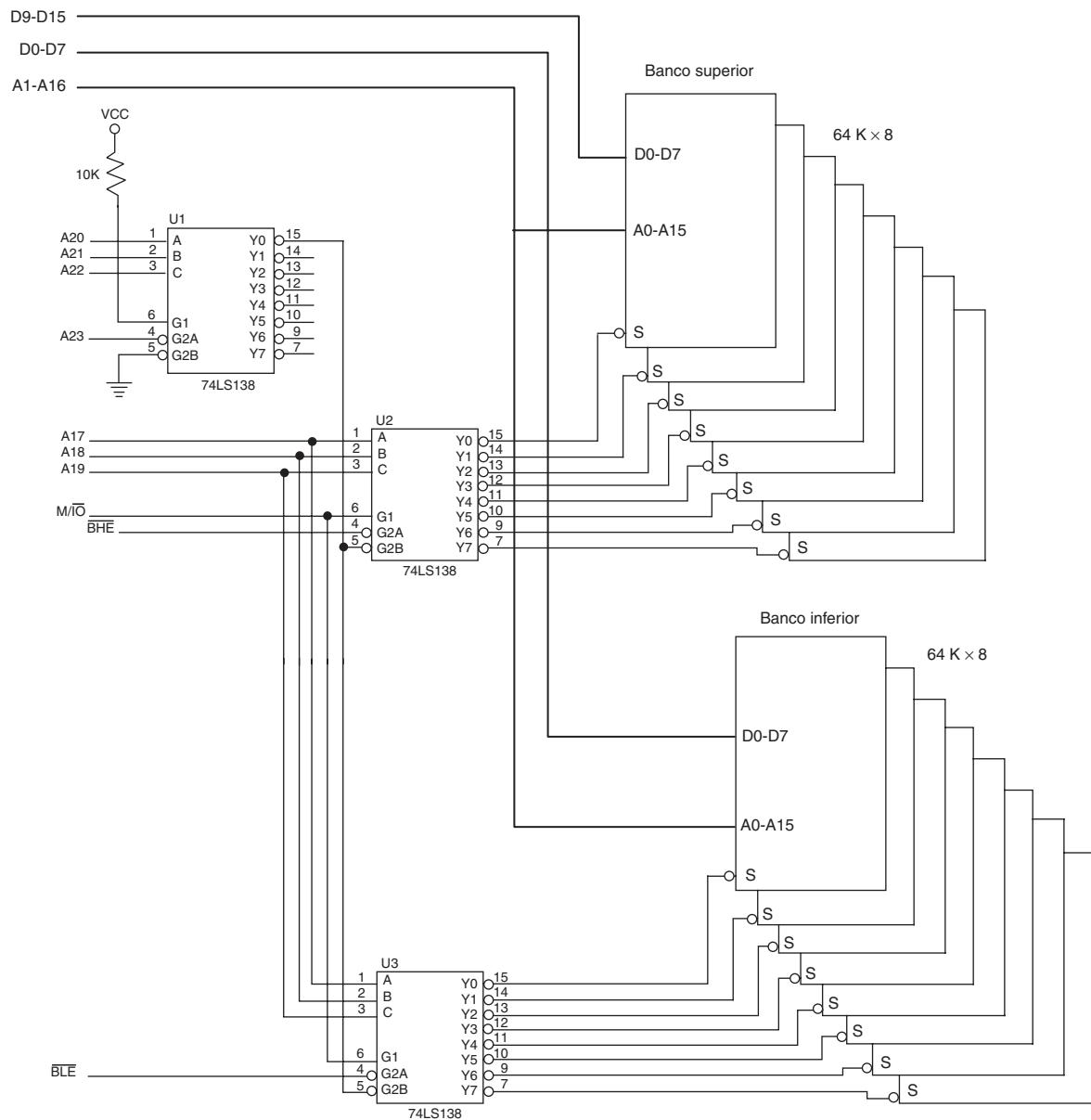
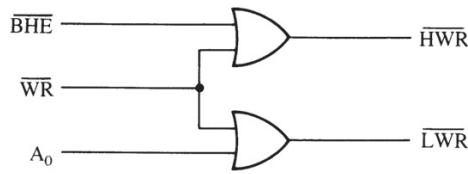


FIGURA 10-26 Decodificadores de bancos separados.

La figura 10-27 muestra la generación de estrobos de escritura separados del 8086 para la memoria. Aquí, una compuerta OR 74LS32 combina A_0 con \overline{WR} para la señal de selección del banco inferior (\overline{LWR}), y \overline{BIE} se combina con \overline{WR} para la señal de selección del banco superior (\overline{HWR}). Los estrobos de escritura para el 80286/80386SX se generan mediante el uso de la señal \overline{MWTC} en vez de WR .

Un sistema de memoria que utiliza estrobos de escritura separados se construye de manera distinta al sistema de 8 bits (8088) o al sistema que utiliza bancos de memoria separados. La memoria en un sistema que utiliza estrobos de escritura se decodifica como una memoria de 16 bits. Por ejemplo, suponga que un sistema de memoria debe contener 64 K bytes de memoria SRAM. Esta memoria requiere dos dispositivos de memoria de 32 Kbytes (62256) para poder construir una memoria de 16 bits. Debido a que la memoria es de 16 bits de ancho y que otro circuito es el que genera las señales de escritura en los bancos, el bit de dirección A_0 toma el valor “no importa”. De hecho, A_0 ni siquiera es una terminal en el microprocesador 80386SX.

FIGURA 10-27 Las señales de entrada de selección de escritura en banco de memoria: $\overline{\text{HWR}}$ (escritura en banco superior) y $\overline{\text{LWR}}$ (escritura en banco inferior).



El ejemplo 10-6 muestra cómo se decodifica una memoria de 16 bits que se almacena en las posiciones 060000H-06FFFFH para el microprocesador 80286 u 80386. La memoria está decodificada en este ejemplo, por lo que el bit A_0 tiene el valor “no importa” para el decodificador. Las posiciones de bit A_1-A_{15} están conectadas a las terminales de dirección A_0-A_{14} del componente de memoria. El decodificador (GAL22V10) habilita ambos dispositivos de memoria mediante el uso de la conexión de dirección $A_{23}-A_{15}$ para seleccionar la memoria siempre que aparezca el valor 06XXXXH en el bus de direcciones.

EJEMPLO 10-6

0000 0110 0000 0000 0000 0000 = 060000H
a
0000 0110 1111 1111 1111 1111 = 06FFFFH
0000 0110 XXXX XXXX xxxx XXXX = 06XXXXH

La figura 10-28 muestra este circuito simple mediante el uso de un GAL22V10 para decodificar la memoria y generar el estrobo de escritura separado. El programa para el decodificador GAL22V10

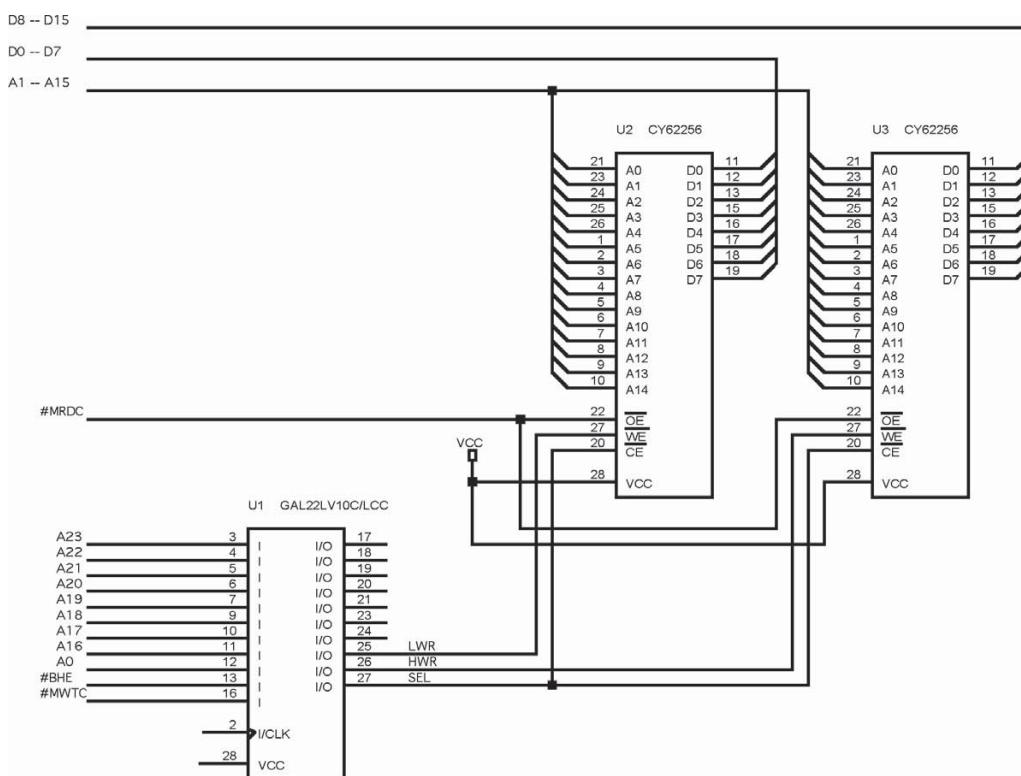


FIGURA 10-28 Una memoria de 16 bits con interfaz en las posiciones de memoria 060000H-06FFFFH.

se muestra en el ejemplo 10-7. Observe que no sólo se selecciona la memoria, sino que el PLD también genera los estrobo de escritura inferior y superior.

EJEMPLO 10-7

```
-- código de VHDL para el decodificador de la figura 10-28

library ieee;
use ieee.std_logic_1164.all;

entity DECODIFICADOR_10_28 is

port (
    A23, A22, A21, A20, A19, A18, A17, A16, A0, BHE, MWTC: in STD_LOGIC;
    SEL, LWR, HWR: out STD_LOGIC
);

end;

architecture V1 of DECODIFICADOR_10_28 is

begin

    SEL <= A23 or A22 or A21 or A20 or A19 or (not A18) or (not A17) or A16;
    LWR <= A0 or MWTC;
    HWR <= BHE or MWTC;

end V1;
```

La figura 10-29 muestra un pequeño sistema de memoria para el microprocesador 8086, que contiene una sección de EPROM y una sección de RAM. Aquí hay cuatro EPROMs 27128 (16 K × 8) que conforman una memoria de 32 K × 16 bits en las posiciones F0000-FFFFFH y cuatro RAMs 62256 (32 K × 8) que conforman una memoria de 64 K × 16 bits en las posiciones 00000H-1FFFFH. (Recuerde que aún y cuando la memoria es de 16 bits de ancho, aún se enumera en bytes.)

Este circuito utiliza un decodificador 74LS139 dual de 2 a 4 líneas que selecciona EPROM con una mitad y RAM con la otra mitad. Decodifica memoria de 16 bits de ancho, no de 8 bits como antes. Observe que el estrobo \overline{RD} está conectado a todas las entradas \overline{OE} de la EPROM y a todas las terminales de entrada \overline{G} de la RAM. Esto se hace porque, aún si el 8086 lee sólo ocho bits de datos, la aplicación de los ocho bits restantes en el bus de datos no tiene efecto sobre la operación del 8086.

Los estrobo \overline{LWR} y \overline{HWR} están conectados a bancos distintos de la memoria RAM. Aquí sí importa si el microprocesador está realizando una operación de escritura de 16 bits o de 8 bits. Si el 8086 escribe un número de 16 bits en la memoria, tanto \overline{LWR} como \overline{HWR} se van al nivel bajo y habilitan las terminales \overline{W} en ambos bancos de memoria. Pero si el 8086 realiza una operación de escritura de 8 bits, sólo uno de los estrobo de escritura se va al nivel bajo y escribe sólo en un banco de memoria. De nuevo, el único momento en el que los bancos hacen una diferencia es para una operación de escritura en la memoria.

Observe que se envía una señal del decodificador de EPROM al generador de estados de espera del 8086, ya que la memoria EPROM, por lo general, requiere un estado de espera. La señal proviene de la compuerta NAND que se utiliza para seleccionar la sección del decodificador de EPROM, por lo que si se selecciona la EPROM se solicita un estado de espera.

La figura 10-30 muestra un sistema de memoria conectado al microprocesador 80386SX mediante el uso de un GAL22V10 como decodificador. Esta interfaz contiene 256 Kbytes de EPROM en la forma de cuatro EPROMs 27512 (64 K × 8) y 128 Kbytes de memoria SRAM en cuatro SRAMs 62256 (32 K × 8).

Observe en la figura 10-30 que el PLD también genera las señales de escritura en banco de memoria \overline{LWR} y \overline{HWR} . Como podemos deducir de este circuito, el número de componentes requeridos para conectar la memoria se reduce a sólo uno, en la mayoría de los casos (el PLD). El listado del programa para el PLD se muestra en el ejemplo 10-8. El PLD decodifica las direcciones de memoria de 16 bits en las posiciones 000000H-01FFFFH para la SRAM, y en las posiciones FC0000H-FFFFFH para la EPROM.

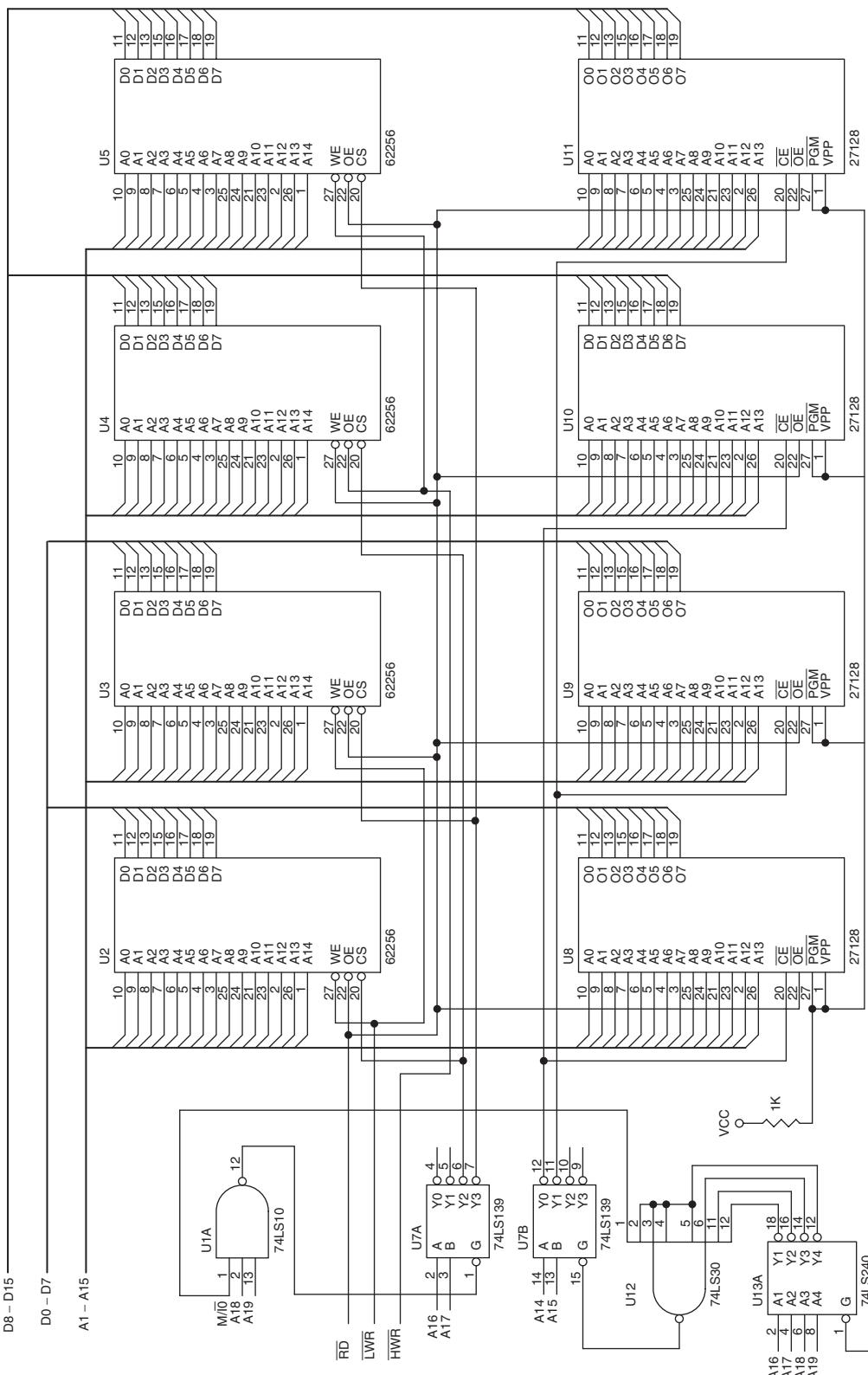


FIGURA 10-29 Un sistema de memoria para el 8086 que contiene una EPROM de 64 Kbytes y una SRAM de 128 Kbytes.

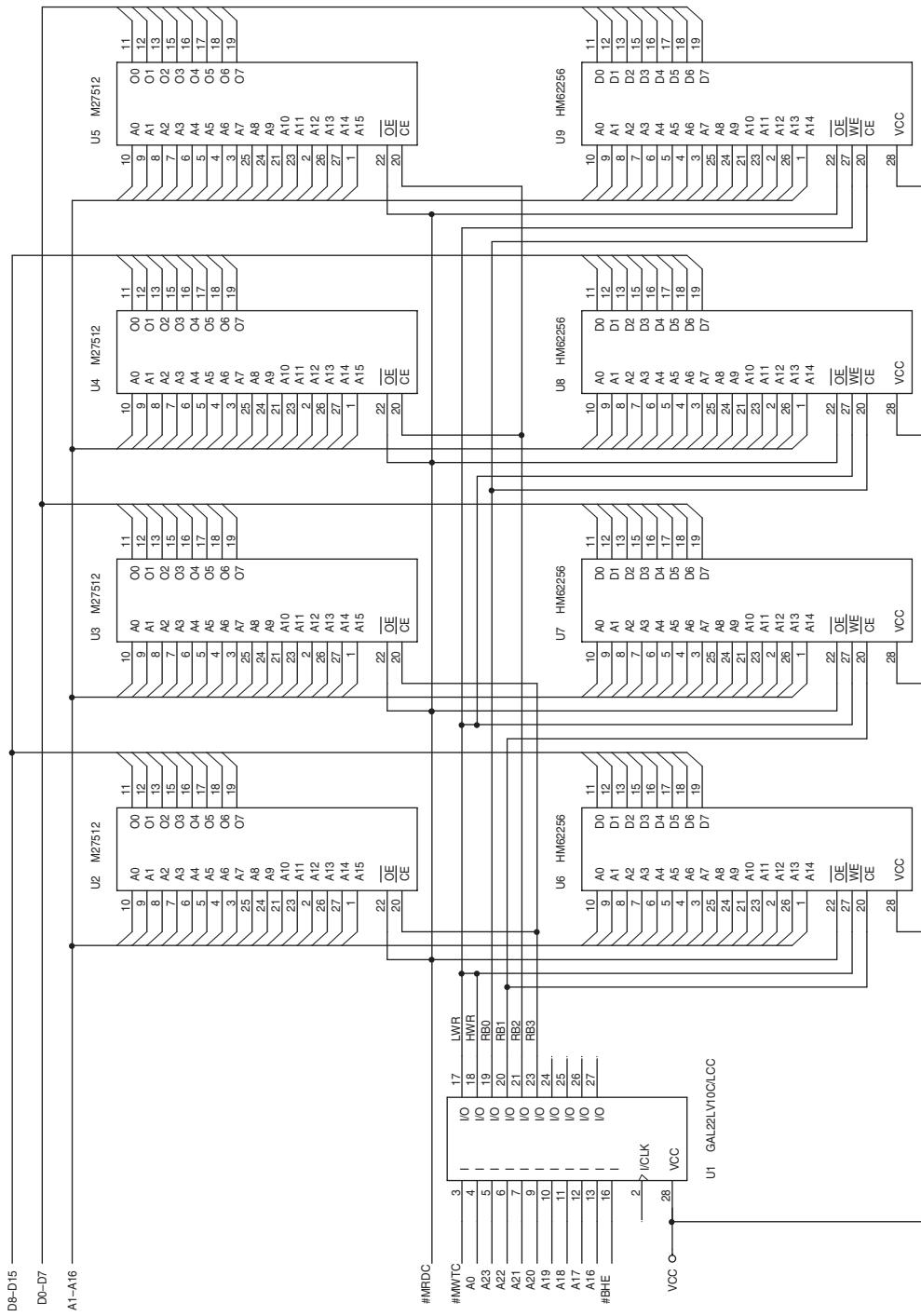


FIGURA 10-30 Un sistema de memoria para el 80386SX que contiene 256 K de EPROM y 128 K de SRAM.

EJEMPLO 10-8

```
-- código de VHDL para el decodificador de la figura 10-30

library ieee;
use ieee.std_logic_1164.all;

entity DECODIFICADOR_10_30 is

port (
    A23, A22, A21, A20, A19, A18, A17, A16, A0, BHE, MWTC: in STD_LOGIC;
    LWR, HWR, RB0, RB1, RB2, RB3: out STD_LOGIC
);
end;

architecture V1 of DECODIFICADOR_10_30 is

begin
    LWR <= A0 or MWTC;
    HWR <= BHE or MWTC;
    RB0 <= A23 or A22 or A21 or A20 or A19 or A18 or A17 or A16;
    RB1 <= A23 or A22 or A21 or A20 or A19 or A18 or A17 or not(A16);
    RB2 <= not(A23 and A22 and A21 and A20 and A19 and A18 and A17);
    RB3 <= not(A23 and A22 and A21 and A20 and A19 and A18 and not(A17));

end V1;
```

10-5**INTERFAZ DE MEMORIA DEL 80386DX Y 80486 (32 BITS)**

Al igual que con los sistemas de memoria de 8 y de 16 bits, el microprocesador se conecta a la memoria a través de su bus de datos y las señales de control que seleccionan bancos de memoria separados. Una de las diferencias en un sistema de memoria de 32 bits es que el microprocesador tiene un bus de datos de 32 bits y cuatro bancos de memoria, en vez de uno o dos. Otra diferencia es que tanto el 80386DX como el 80486 (tanto SX como DX) contienen un bus de direcciones de 32 bits que por lo general requiere decodificadores PLD en vez de decodificadores integrados, debido al número variable de bits de dirección.

Bancos de memoria

En la figura 10-31 se muestran los bancos de memoria para los microprocesadores 80386DX y 80486. Observe que estos extensos sistemas de memoria contienen cuatro bancos de 8 bits y cada uno de ellos contiene hasta 1 Gbyte de memoria. La selección de bancos se realiza mediante las señales de selección de bancos $\overline{BE3}$, $\overline{BE2}$, $\overline{BE1}$ y $\overline{BE0}$. Si se transfiere un número de 32 bits se seleccionan todos los bancos; si se transfiere un número de 16 bits se seleccionan dos bancos (por lo general $\overline{BE3}$ y $\overline{BE2}$, o $\overline{BE1}$ y $\overline{BE0}$); y si se transfieren 8 bits, se selecciona un solo banco.

Al igual que los microprocesadores 8086/80286/80386SX, el 80386DX y el 80486 requieren señales de estrobo de escritura separadas para cada banco de memoria. Estos estrobos de escritura separados se desarrollan (como se muestra en la figura 10-32) mediante el uso de una compuerta OR simple o de otro componente lógico.

Interfaz de memoria de 32 bits

Basándose en la discusión anterior, podemos deducir que una interfaz de memoria para el 80386DX o el 80486 requiere que se generen cuatro estrobo de escritura en banco y que se decodifique una dirección de 32 bits. No hay decodificadores integrados (como el 74LS138) que puedan alojar con facilidad una interfaz de memoria para los microprocesadores 80386DX u 80486. Observe que los bits A_0 y A_1 tienen el valor “no importa” cuando se decodifica una memoria de 32 bits. Estos bits de dirección se utilizan

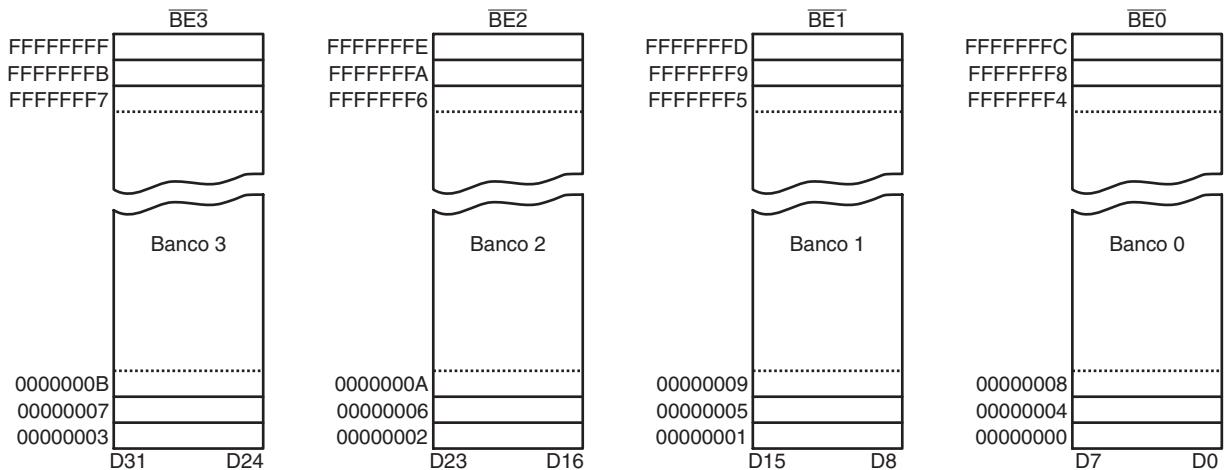
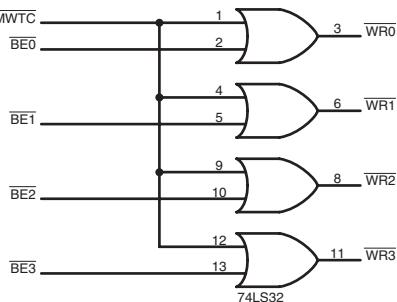


FIGURA 10-31 La organización de la memoria para los microprocesadores 80386DX y 80486.

FIGURA 10-32 Señales de escritura en banco para los microprocesadores 80386DX y 80486.



dentro del microprocesador para generar las señales de habilitación de los bancos. Observe que la conexión A₂ del bus de direcciones se conecta a la terminal de dirección A₀ de la memoria. Esto ocurre ya que no hay una terminal A₀ o A₁ en el microprocesador 80486.

La figura 10-33 muestra un sistema de memoria SRAM de 512 K × 8 para el microprocesador 80486. Esta interfaz utiliza ocho dispositivos de memoria SRAM de 64 K × 8, un PLD y una compuerta OR. Esta compuerta se requiere debido al número de conexiones de dirección que contiene el microprocesador. Este sistema coloca la memoria SRAM en las posiciones 02000000H-0203FFFFH. En el ejemplo 10-9 se muestra el programa para el dispositivo PLD.

EJEMPLO 10-9

```

library ieee;
use ieee.std_logic_1164.all;

entity DECODIFICADOR_10_30 is
port (
    A30, A29, A28, A27, A26, A25, A24, A23, A22, A21, A19, BE0, BE1, BE2,
    BE3, MWTC: in STD_LOGIC;
    RB0, RB1, WR0, WR1, WR2, WR3: out STD_LOGIC
);
end;

architecture V1 of DECODIFICADOR_10_30 is

```

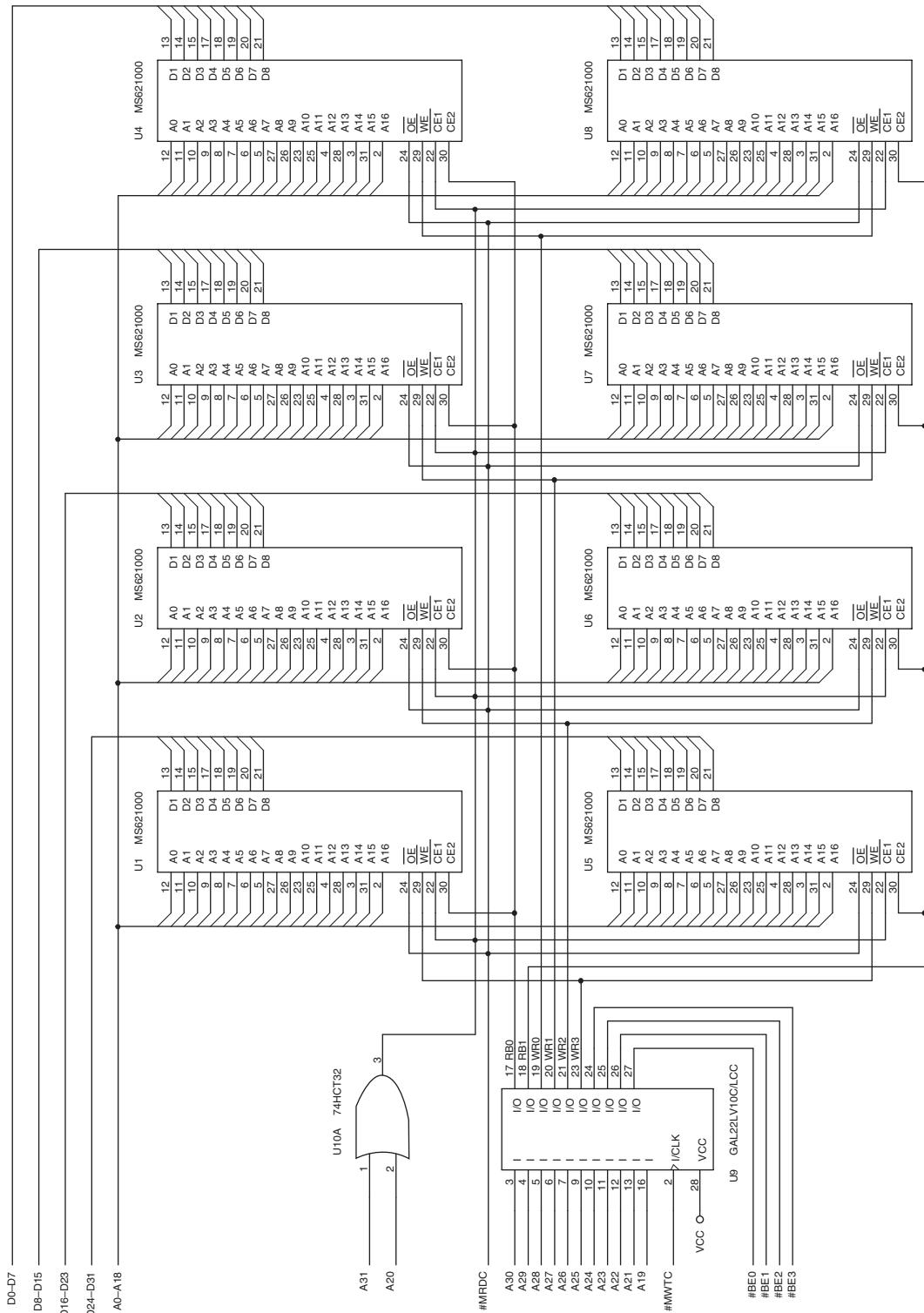


FIGURA 10-33 Un pequeño sistema de memoria SRAM de 512 Kbytes para el microprocesador 80486.

```

begin
    WR0 <= BE0 or MWTC;
    WR1 <= BE1 or MWTC;
    WR2 <= BE2 or MWTC;
    WR3 <= BE3 or MWTC;
    RB0 <= A30 or A29 or A28 or A27 or A26 or A25 or A24 or A23 or A22
        or A21 or A19;
    RB1 <= A30 or A29 or A28 or A27 or A26 or A25 or A24 or A23 or A22
        or A21 or not(A19);

end V1;

```

Aunque no lo mencionamos en esta sección, los microprocesadores 80386DX y 80486 operan con velocidades de reloj muy altas que, por lo general, requieren estados de espera para el acceso a la memoria. En los capítulos 17 y 18 hablaremos sobre los cálculos del tiempo de acceso para estos microprocesadores. La interfaz proporciona una señal que se utiliza con el generador de estados de espera, la cual no se muestra en esta sección. Otros dispositivos con estos microprocesadores de mayor velocidad son los sistemas de memoria caché y de memoria intercalada, que también presentaremos en el capítulo 17 con los microprocesadores 80386DX y 80486.

10-6

INTERFAZ DE MEMORIA DEL PENTIUM AL PENTIUM 4 (64 BITS)

Los microprocesadores del Pentium al Pentium 4 (excepto la versión P24T del Pentium) contienen un bus de datos de 64 bits, el cual requiere ocho decodificadores (uno por banco) u ocho señales de escritura separadas. En la mayoría de los sistemas se utilizan señales de escritura separadas con este microprocesador, cuando se conecta a la memoria. La figura 10-34 muestra la organización de la memoria del Pentium y sus ocho bancos de memoria. Observe que esta organización es casi idéntica a la del 80486, sólo que contiene ocho bancos en vez de cuatro.

Al igual que con versiones anteriores del microprocesador Intel, se requiere esta organización para la compatibilidad ascendente de la memoria. Las señales de estrobo de escritura separadas se obtienen mediante la combinación de las señales de habilitación de banco con la señal MWTC, la cual se genera mediante la combinación de M/IO con W/R. El circuito que se emplea para las señales de escritura en los bancos aparece en la figura 10-35. Como puede imaginar, a menudo se utiliza un PLD para la generación de las señales de escritura en los bancos.

Interfaz de memoria de 64 bits

La figura 10-36 muestra un pequeño sistema de memoria para el Pentium-Pentium 4. Este sistema utiliza un PLD para decodificar la dirección de memoria y contiene ocho dispositivos de memoria EPROM 27C4001 (512 K × 8), que se conectan con el Pentium-Pentium 4 en las posiciones FFC0000H a FFFFFFFFH. Esto representa un tamaño total de memoria de 4 Mbytes, organizados de tal forma que cada banco contenga dos componentes de memoria. Observe que los microprocesadores del Pentium Pro al Pentium 4 pueden configurarse con 36 conexiones de dirección, lo cual permite hasta 64 G de memoria.

La decodificación de la memoria (que se muestra en el ejemplo 10-10) es similar a los ejemplos anteriores, sólo que en los microprocesadores del Pentium al Pentium 4 se ignoran los tres bits de dirección de más a la derecha (A_2-A_0). En este caso, el decodificador selecciona secciones de memoria que son de 64 bits y que contienen 4 Mbytes de memoria EPROM.

La entrada de dirección A_0 de cada dispositivo de memoria se conecta a la salida de dirección A_3 del Pentium y superiores. Esta conexión de dirección asimétrica continúa hasta que se conecta la entrada de dirección A_{18} de la memoria con la salida de dirección A_{22} del Pentium. Las posiciones de dirección $A_{22}-A_{31}$ se decodifican mediante un PLD. En el ejemplo 10-10 se muestra el programa para el dispositivo PLD, para las posiciones de memoria FFC0000H-FFFFFFFFFFH.

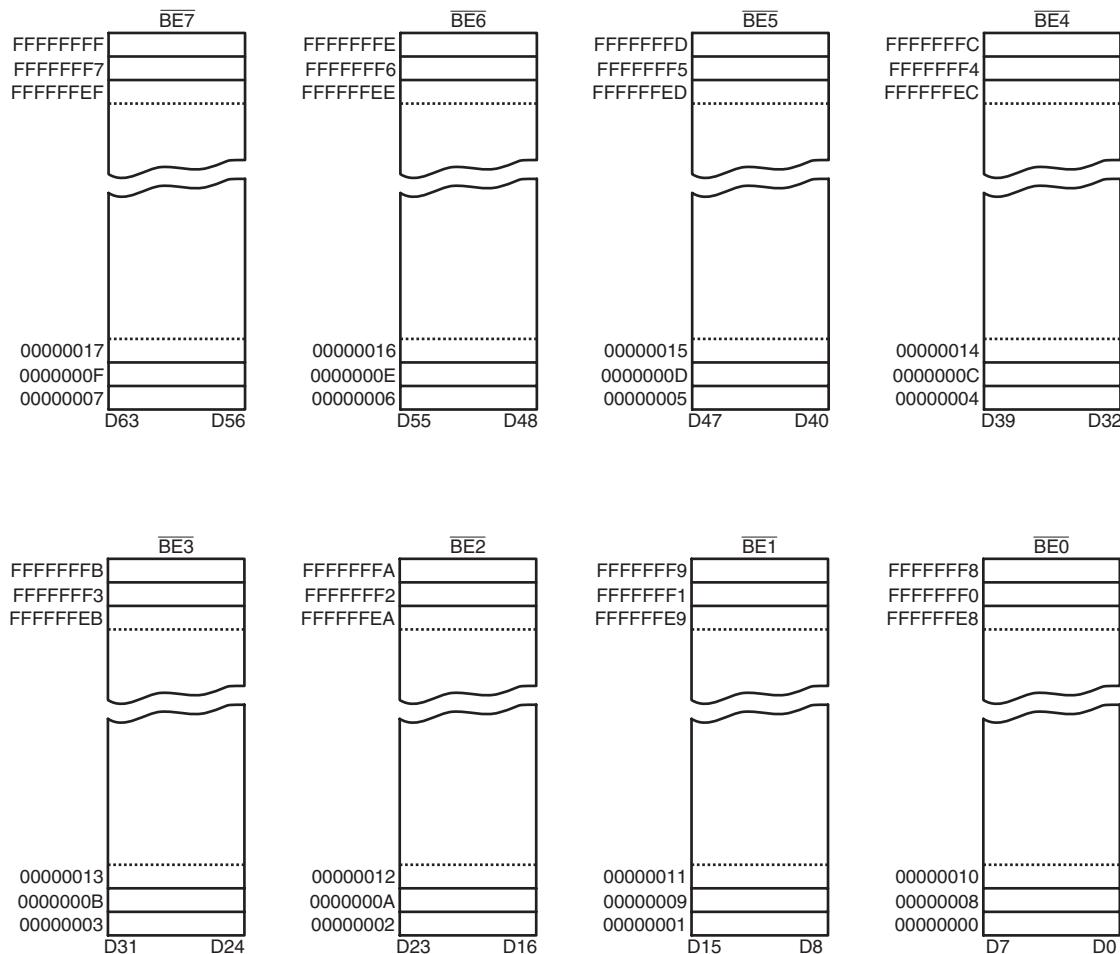


FIGURA 10-34 La organización de la memoria de los microprocesadores Pentium-Pentium 4.

EJEMPLO 10-10

```

Library ieee;
use ieee.std_logic_1164.all;

entity DECODIFICADOR_10_36 is
port (
    A31, A30, A29, A28, A27, A26, A25, A24, A23, A22: in STD_LOGIC;
    SEL: out STD_LOGIC
);
end;

architecture V1 of DECODIFICADOR_10_36 is
begin
    SEL <= not(A31 and A30 and A29 and A28 and A27 and A26 and A25 and A24
        and A23 and A22);
end V1;

```

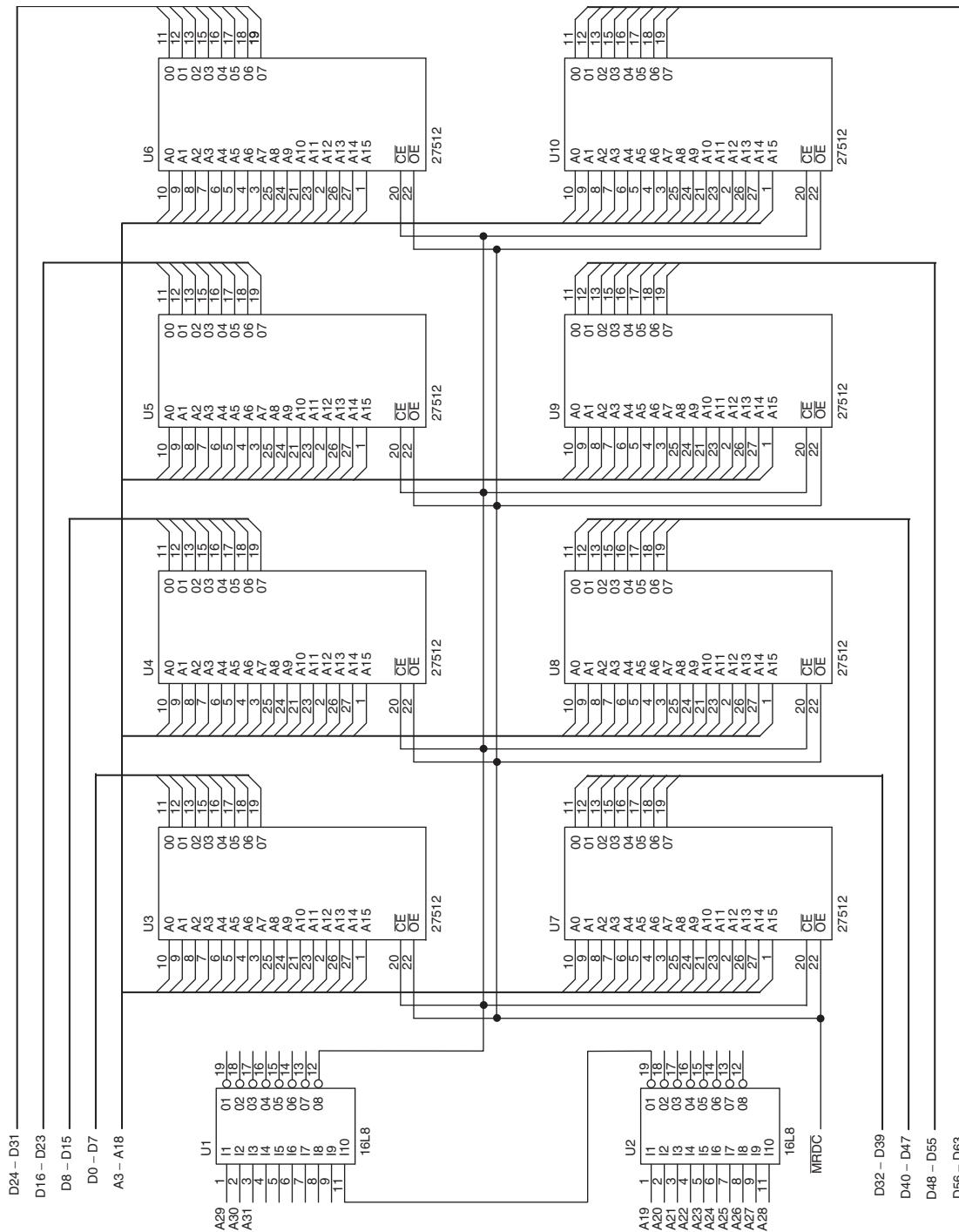


FIGURA 10-35 Una memoria EEPROM pequeña de 512 Kbytes conectada a los microprocesadores Pentium-Pentium 4.

D0-D7

D8-D15

D16-D23

D24-D31

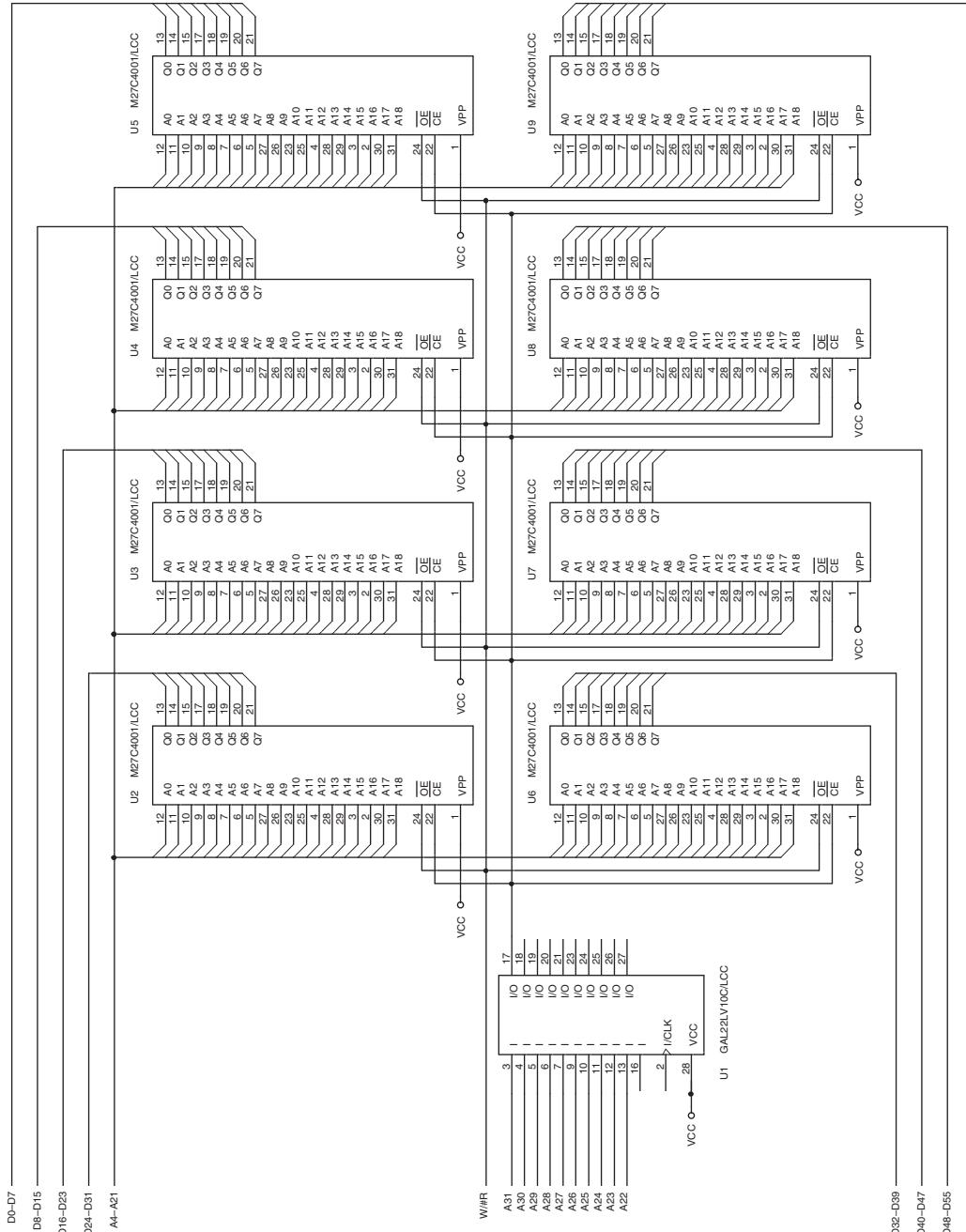


FIGURA 10-36 Un sistema de memoria EPROM pequeño de 4 Mbytes para los microprocesadores Pentium-Pentium 4.

10-7

RAM DINÁMICA

Como por lo general un sistema de memoria RAM es muy grande, requiere muchos dispositivos SRAM a un costo elevado, o sólo unos cuantos dispositivos DRAM (**RAM dinámica**) a un costo mucho menor. La memoria DRAM, de la cual se dio una breve descripción en la sección 10-1, es bastante compleja ya que requiere multiplexión de direcciones y operaciones de refresco. Por fortuna, los fabricantes de circuitos integrados cuentan con un controlador de RAM dinámica que incluye los multiplexores de dirección y todos los circuitos de sincronización necesarios para refrescar la memoria.

En esta sección hablaremos sobre el dispositivo de memoria DRAM con mucho más detalle que en la sección 10-1; además proporcionaremos información sobre el uso de un controlador dinámico en un sistema de memoria.

Más detalles sobre la DRAM

Como se mencionó en la sección 10-1, una DRAM retiene datos durante sólo 2 a 4 ms y requiere la multiplexión de las entradas de dirección. Aunque ya hablamos sobre los multiplexores de dirección en la sección 10-1, aquí explicaremos con detalle la operación de la DRAM durante el tiempo que se refresca.

Como dijimos antes, una DRAM debe refrescarse en forma periódica ya que almacena datos de manera interna mediante capacitores que pierden su carga en un periodo corto de tiempo. Para refrescar una DRAM, debe leerse o escribirse en forma periódica el contenido de una sección de la memoria. Cualquier operación de lectura o de escritura refresca automáticamente toda una sección completa de la DRAM. El número de bits que se refrescan depende del tamaño del componente de memoria y de su organización interna.

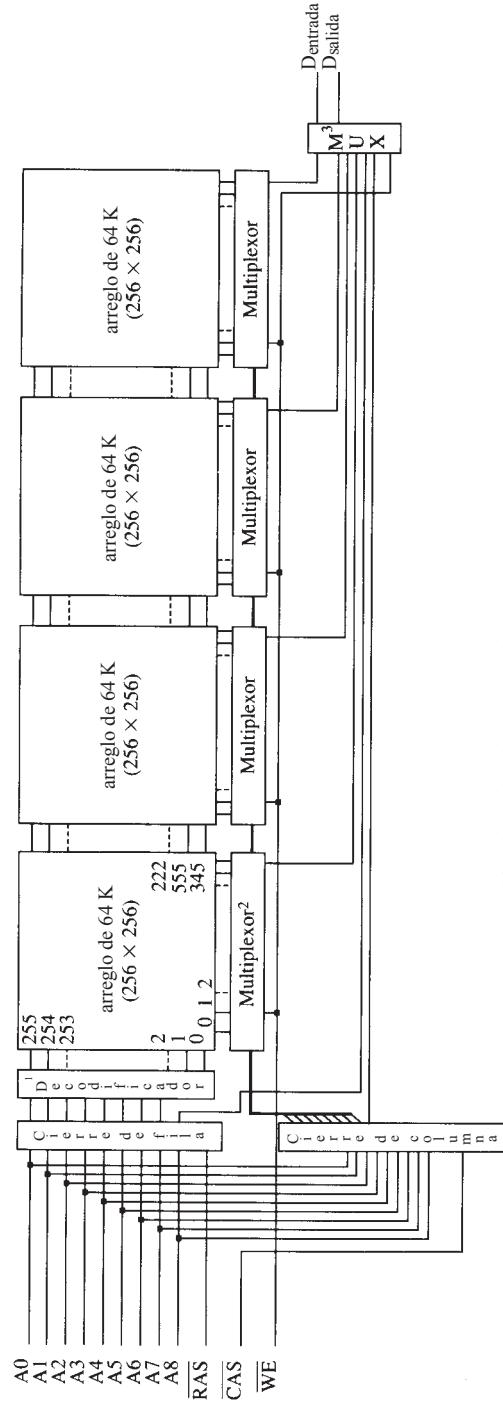
Los ciclos de refresco se realizan cuando se hace una operación de lectura, de escritura o en un ciclo de refresco especial que no requiere de leer o escribir datos. El ciclo de refresco es interno para la DRAM y, por lo general, se realiza mientras operan los demás componentes en el sistema. A este tipo de refresco de memoria se le llama *refresco oculto*, *refresco transparente* o algunas veces *robo de ciclo*.

Para poder realizar un refresco oculto mientras funcionan otros componentes del sistema, en un ciclo se envía sólo un estrobo mediante la señal RAS con una dirección de fila hacia la DRAM para seleccionar una fila de bits a refrescar. La entrada RAS también hace que la fila seleccionada se lea en forma interna y se reescriba en los bits seleccionados. Esto recarga los capacitores internos que almacenan los datos. Este tipo de refresco se oculta del sistema, ya que ocurre mientras el microprocesador lee o escribe en otras secciones de la memoria.

La organización interna de la DRAM contiene una serie de filas y columnas. Una DRAM de 256 K × 1 tiene 256 columnas, cada una de las cuales contiene 256 bits o filas organizadas en cuatro secciones de 64 Kbits cada una. Cada vez que se direcciona una posición de memoria, la dirección de columna selecciona una columna (o palabra de memoria interna) de 1024 bits (uno por cada sección de la DRAM). En la figura 10-37 podrá ver la estructura interna de una DRAM de 256 K × 1. Los dispositivos de memoria más grandes están estructurados de manera similar al dispositivo de 256 K × 1. Por lo general, la diferencia está en el tamaño de cada sección o en el número de secciones en paralelo.

La figura 10-38 muestra la sincronización para un ciclo de refresco en el que sólo hay una señal RAS. La diferencia entre la señal RAS y una lectura o escritura es que sólo se aplica una dirección de refresco, la cual por lo general se obtiene de un contador binario de 7 u 8 bits. El tamaño del contador se determina sobre la base del tipo de DRAM que se va a refrescar. El contador de refresco se incrementa al final de cada ciclo de refresco, de manera que todas las filas se refrescan en 2 o 4 ms, dependiendo del tipo de DRAM.

Si hay que refrescar 256 filas en un lapso no mayor a 4 ms, como en una DRAM de 256 K × 1, entonces el ciclo de refresco debe activarse por lo menos cada 15.6 μ s, para poder cumplir con la especificación de refresco. Por ejemplo, el 8086/8088 (si se ejecuta a una velocidad de reloj de 5 MHz) tarda 800 ns en realizar una operación de lectura o de escritura. Como la DRAM debe tener un ciclo de refresco cada 15.6 μ s, para cada 19 lecturas o escrituras en memoria el sistema de memoria debe ejecutar un ciclo de refresco, ya que de lo contrario se perderán los datos. Esto representa una pérdida



Notas: 1. El decodificador es de 8 líneas a 256 líneas.
 2. El multiplexor es de 256 a 1 línea.
 3. El multiplexor es de 4 a 1 línea.

FIGURA 10-37 La estructura interna de una DRAM de $256 \text{ K} \times 1$. Observe que cada una de las 256 palabras internas son de 1024 bits.

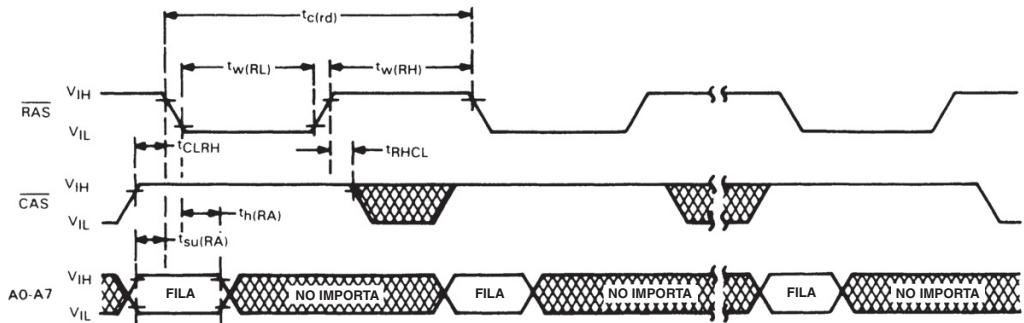


FIGURA 10-38 Diagrama de sincronización del ciclo de refresco $\overline{\text{RAS}}$ para la DRAM TMS4464. (Cortesía de Texas Instruments Corporation.)

del 5% del tiempo de la computadora, un pequeño precio a pagar por los ahorros que representa el uso de la RAM dinámica. En un sistema moderno tal como un Pentium 4 a 3.0 GHz, 15.6 μs es una cantidad considerable de tiempo. Como el Pentium 4 a 3.0 GHz ejecuta una instrucción en aproximadamente un tercio de ns (se ejecutan muchas instrucciones en un solo ciclo de reloj), puede ejecutar cerca de 46,000 instrucciones entre cada ciclo de refresco. Esto significa que en los nuevos equipos se requiere mucho menos del 1% (=0.002%) del tiempo para un ciclo de refresco.

Memoria EDO

Una ligera modificación de la estructura de la DRAM transforma al dispositivo en una DRAM EDO (**salida extendida de datos**). En la memoria EDO, un acceso a memoria (incluyendo un refresco) almacena en cierres los 256 bits seleccionados por $\overline{\text{RAS}}$. Estos cierres almacenan los siguientes 256 bits de información, por lo que en la mayoría de los programas (que se ejecutan en forma secuencial) los datos están disponibles sin necesidad de estados de espera. Esta ligera modificación a la estructura interna de la DRAM incrementa el rendimiento del sistema de un 15 a un 25%. Aunque la memoria EDO ya no está disponible en el mercado, esta técnica se emplea aún en todos los dispositivos DRAM modernos.

SDRAM

La RAM dinámica sincrónica (**SDRAM**) se utiliza con la mayoría de los sistemas más recientes en una forma u otra, debido a su velocidad. Hay versiones disponibles con tiempos de acceso de 10 ns para utilizarse con un bus de sistema de 66 MHz; de 8 ns para utilizarse con un bus de sistema de 100 MHz; y de 7 ns para el bus de 133 MHz. Al principio, el tiempo de acceso podría llevártelo a uno a pensar que estos dispositivos operan sin estados de espera, pero no es verdad. Después de todo, el tiempo de acceso de la DRAM es de 60 ns y el tiempo de acceso de la SDRAM es de 10 ns. Este tiempo de acceso de 10 ns es incorrecto, ya que sólo se aplica a la segunda, tercera y cuarta lecturas de 64 bits del dispositivo. La primera lectura requiere el mismo número de estados de espera que una DRAM estándar.

Cuando ocurre una transferencia tipo ráfaga desde el microprocesador hacia la SDRAM, se requieren tres o cuatro ciclos de reloj para que se lea el primer número de 64 bits. Cada número subsiguiente se lee sin estados de espera y en un ciclo de bus cada uno. Como las ráfagas de SDRAM leen cuatro números de 64 bits, y como las lecturas de la segunda a la cuarta no requieren estados de espera y pueden leerse en un ciclo de bus cada una, la SDRAM es superior en funcionamiento a la DRAM o incluso a la memoria EDO. Esto significa que se requieren tres ciclos de bus para el primer número y tres ciclos más para los siguientes tres números; un total de siete ciclos de reloj del bus para leer cuatro números de 64 bits. Si se compara con la DRAM, que requiere tres ciclos de reloj por cada número (que en este caso serían 12 ciclos), puede verse el aumento en velocidad. La mayoría de las estimaciones indican que la SDRAM presenta un aumento de casi un 10% más de rendimiento en comparación con la memoria EDO.

DDR

La memoria de doble velocidad de datos (DDR) es la última de varias mejoras sobre la RAM. La memoria DDR transfiere datos al doble de velocidad que la SDRAM, debido a que transfiere datos en cada borde del reloj. Aún y cuando esto parece indicar que se duplica la velocidad de la memoria, en realidad no es así. La razón principal es que aún existe el problema del tiempo de acceso, incluso con la memoria más avanzada que requiere un tiempo de acceso de 40 ns. Si consideramos un microprocesador que se ejecuta a muchos GHz, éste es un tiempo muy largo para esperar a la memoria. Por consecuencia, la velocidad no se duplica, aún y cuando así lo indique el nombre.

Controladores de DRAM

En la mayoría de los sistemas, un circuito integrado controlador de DRAM realiza la tarea de multiplexar direcciones y generar las señales de control de la DRAM. Algunos microprocesadores integrados más recientes (como el 80186/80188) incluyen el circuito de refresco como parte del microprocesador. La mayoría de las computadoras modernas contienen el controlador de DRAM en el conjunto de chips, por lo que no hay un controlador de DRAM individual disponible. El controlador de DRAM en el conjunto de chips para el microprocesador sincroniza de refresco y los inserta en la sincronización. El refresco de memoria es transparente para el microprocesador, ya que en realidad éste no controla el refresco.

10-8 RESUMEN

1. Todos los dispositivos de memoria tienen entradas de dirección; entradas y salidas de datos, o sólo salidas; una terminal para selección y una o más terminales que controlan la operación de la memoria.
2. En un componente de memoria, las conexiones de dirección se utilizan para seleccionar una de las posiciones de memoria dentro del dispositivo. Diez terminales de dirección tienen 1024 combinaciones y, por lo tanto, pueden direccionar 1024 posiciones de memoria distintas.
3. En una memoria, las conexiones de datos se utilizan para introducir la información que se va a almacenar en una posición de memoria, y también para recuperar la información que se lee de una posición de memoria. Por ejemplo, los fabricantes listan su memoria como de $4\text{ K} \times 4$, lo que significa que el dispositivo tiene 4 K posiciones de memoria (4096) y que se almacenan cuatro bits en cada posición.
4. La selección de memoria se lleva a cabo mediante una terminal de selección de chip ($\overline{\text{CS}}$) en muchas memorias RAM, o mediante una terminal de habilitación de chip ($\overline{\text{CE}}$) en muchas memorias EPROM o ROM.
5. La función de la memoria se selecciona mediante una terminal de habilitación de salida ($\overline{\text{OE}}$) para leer datos, la cual por lo general se conecta a la señal de lectura del sistema ($\overline{\text{RD}}$ o $\overline{\text{MRDC}}$). La terminal de habilitación de escritura ($\overline{\text{WE}}$), que se utiliza para escribir datos, se conecta por lo general a la señal de escritura del sistema ($\overline{\text{WR}}$ o $\overline{\text{MWTC}}$).
6. Una memoria EPROM se programa mediante un programador de EPROMs y puede borrarse si se expone a la luz ultravioleta. En la actualidad, las EPROMs están disponibles en tamaños desde $1\text{ K} \times 8$ hasta $512\text{ K} \times 8$ y mayores.
7. La memoria Flash (EEPROM) se programa en el sistema mediante el uso de un pulso de programación de 12 V o de 5.0 V.
8. La RAM estática (SRAM) retiene datos durante el tiempo que esté conectada la fuente de poder del sistema. Estos tipos de memorias están disponibles en tamaños hasta $128\text{ K} \times 8$.
9. La RAM dinámica (DRAM) retiene datos sólo durante un periodo corto de tiempo, por lo general de 2 a 4 ms. Esto crea problemas para el diseñador del sistema de memoria, ya que la DRAM debe refrescarse en forma periódica. Las DRAMs también tienen entradas de dirección multiplexadas, las cuales requieren un multiplexor externo para proporcionar cada una de las dos mitades de la dirección en el momento apropiado.

10. Los decodificadores de direcciones de memoria seleccionan una EPROM o una RAM en un área específica de la memoria. Los decodificadores de direcciones que se utilizan con más frecuencia son el decodificador 74LS138 de 3 a 8 líneas, el decodificador 74LS139 de 2 a 4 líneas y la lógica de selección programada en la forma de un PLD.
11. El decodificador de direcciones PLD para los microprocesadores del 8088 al Pentium 4 reduce el número de circuitos integrados que se requieren para completar un sistema de memoria funcional.
12. La interfaz de memoria en modo mínimo del 8088 contiene 20 líneas de dirección, ocho líneas de datos y tres líneas de control: \overline{RD} , \overline{WR} e $\overline{IO/M}$. La memoria del 8088 funciona en forma correcta sólo cuando se utilizan todas estas líneas para la interfaz de memoria.
13. La velocidad de acceso de la EPROM debe ser compatible con el microprocesador al que se va a conectar. Muchas EPROMs disponibles en la actualidad tienen un tiempo de acceso de 450 ns, que es demasiado lento para el 8088 a 5 MHz. Para solucionar este problema se inserta un estado de espera para incrementar el tiempo de acceso a la memoria hasta 660 ns.
14. Las características de corrección de errores también están disponibles en los sistemas de memoria, pero se requiere el almacenamiento de muchos bits más. Si se almacena un número de 8 bits con un circuito de corrección de errores, en realidad requiere 13 bits de memoria: cinco para un código de comprobación de errores y ocho para los datos. La mayoría de los circuitos integrados de corrección de errores sólo pueden corregir un error en un solo bit.
15. La interfaz de memoria del 8086/80286/80386SX tiene un bus de datos de 16 bits y contiene una terminal de control M/\overline{IO} , mientras que la del 8088 tiene un bus de datos de 8 bits y contiene una terminal $\overline{IO}/\overline{M}$. Además de estos cambios hay una señal de control adicional: habilitación de bus superior (\overline{BHE}).
16. La memoria del 8086/80386/80386SX está organizada en dos bancos de 8 bits: banco superior y banco inferior. El banco superior de memoria se habilita mediante la señal de control \overline{BHE} y el banco inferior se habilita mediante la señal de dirección A_0 o mediante la señal de control \overline{BLE} .
17. Hay dos esquemas comunes para seleccionar los bancos en un sistema basado en los microprocesadores 8086/80286/80386SX: (1) un decodificador separado para cada banco y (2) señales de control \overline{WR} separadas para cada banco con un decodificador común.
18. La memoria que se conecta al 80386DX y al 80486DX es de 32 bits, la cual se selecciona mediante un bus de direcciones de 32 bits. Debido a la anchura de esta memoria, se organiza en cuatro bancos de memoria de ocho bits cada uno. El microprocesador proporciona las siguientes señales de selección de banco: \overline{BE}_3 , \overline{BE}_2 , \overline{BE}_1 y \overline{BE}_0 .
19. La memoria que se conecta a los microprocesadores Pentium-Pentium 4 es de 64 bits y se selecciona mediante un bus de direcciones de 32 bits. Debido a la anchura de la memoria, se organiza en ocho bancos de ocho bits cada uno. El microprocesador proporciona las señales de selección de banco $\overline{BE}_7-\overline{BE}_0$.
20. Los controladores de RAM dinámica están diseñados para controlar componentes de memoria DRAM. Muchos de los controladores de DRAM que se utilizan en la actualidad contienen multiplexores de direcciones, contadores de refresco y los circuitos requeridos para realizar un refresco periódico de la memoria DRAM.

10-9**PREGUNTAS Y PROBLEMAS**

1. ¿Qué tipos de conexiones son comunes para todos los dispositivos de memoria?
2. Liste el número de palabras en cada dispositivo de memoria para los siguientes números de conexiones de dirección:
 - (a) 8
 - (b) 11
 - (c) 12
 - (d) 13

3. Liste el número de elementos de datos que se almacenan en cada uno de los siguientes dispositivos de memoria, y el número de bits en cada dato:
 - (a) $2\text{ K} \times 4$
 - (b) $1\text{ K} \times 1$
 - (c) $4\text{ K} \times 8$
 - (d) $16\text{ K} \times 1$
 - (e) $64\text{ K} \times 4$
4. ¿Cuál es el propósito de la terminal $\overline{\text{CS}}$ o $\overline{\text{CE}}$ en un componente de memoria?
5. ¿Cuál es el propósito de la terminal $\overline{\text{OE}}$ en un dispositivo de memoria?
6. ¿Cuál es el propósito de la terminal $\overline{\text{WE}}$ en una SRAM?
7. ¿Cuántos bytes de almacenamiento contienen los siguientes dispositivos de memoria EPROM?
 - (a) 2708
 - (b) 2716
 - (c) 2732
 - (d) 2764
 - (e) 27512
8. ¿Por qué una EPROM de 450 ns no funciona directamente con un 8088 a 5 MHz?
9. ¿Qué puede decirse acerca de la cantidad de tiempo que se necesita para borrar y escribir en una posición de un dispositivo de memoria Flash?
10. ¿Qué tipo de dispositivo utiliza el acrónimo SRAM?
11. La memoria 4016 tiene una terminal \overline{G} , una terminal \overline{S} y una terminal \overline{W} . ¿Para qué se utilizan estas terminales en esta RAM?
12. ¿Cuánto tiempo de acceso a memoria requiere la 4016 más lenta?
13. ¿Qué tipo de dispositivo utiliza el acrónimo DRAM?
14. La DIMM de 256 M tiene 28 entradas de dirección; aún así es una DRAM de 256 M. Explique cómo se fuerza una dirección de memoria de 28 bits en 14 entradas de dirección.
15. ¿Cuáles son los propósitos de las entradas $\overline{\text{CAS}}$ y $\overline{\text{RAS}}$ de una DRAM?
16. ¿Cuánto tiempo se requiere para refrescar una RAM típica?
17. ¿Por qué son importantes los decodificadores de dirección?
18. Modifique el decodificador de compuerta NAND de la figura 10-13 para que seleccione la memoria para el rango de direcciones DF800H-DFFFFH.
19. Modifique el decodificador de compuerta NAND de la figura 10-13 para que seleccione la memoria para el rango de direcciones 40000H-407FFH.
20. Cuando la entrada G1 está en nivel alto y tanto $\overline{G2A}$ como $\overline{G2B}$ están en nivel bajo, ¿qué ocurre a las salidas del decodificador 74HCT138 de 3 a 8 líneas?
21. Modifique el circuito de la figura 10-15 para que dirija la memoria 70000H-7FFFFH.
22. Modifique el circuito de la figura 10-15 para que dirija la memoria 40000H-4FFFFH.
23. Describa el decodificador 74LS139.
24. ¿Qué significa VHDL?
25. ¿Cuáles son las cinco palabras clave principales en VHDL para las cinco funciones lógicas principales (AND, OR, NAND, NOR e inversión)?
26. ¿En qué bloque principal de un programa en VHDL se colocan las ecuaciones?
27. Modifique el circuito de la figura 10-19 mediante la reescritura del programa del PLD para que dirija la memoria en las posiciones A0000H-BFFFFH para la ROM.
28. ¿Cuáles son las dos señales de control que sustituyen a las señales de control $\overline{\text{RD}}$ y $\overline{\text{WR}}$ en modo mínimo en el 8086 en modo máximo?
29. Modifique el circuito de la figura 10-20 para que seleccione la memoria en las posiciones 60000H-77FFFFH.
30. Modifique el circuito de la figura 10-20 para que seleccione ocho EPROMs 27256 de $32\text{K} \times 8$ en las posiciones de memoria 40000H-7FFFFH.
31. Agregue otro decodificador al circuito de la figura 10-21, de manera que se agreguen ocho SRAMs 62256 adicionales en las posiciones C0000H-FFFFFH.

32. El circuito 74LS636 de corrección y detección de errores almacena un código de comprobación con cada byte de datos. ¿Cuántos bits se almacenan para el código de comprobación?
33. ¿Cuál es el propósito de la terminal SEF en el 74LS636?
34. El 74LS636 corrige _____ bits que tienen error.
35. Describa la principal diferencia entre los buses de los microprocesadores 8086 y 8088.
36. ¿Cuál es el propósito de las terminales \overline{BHE} y A_0 en el microprocesador 8086?
37. ¿Cuál es la terminal \overline{BLE} y qué otra terminal sustituye?
38. ¿Cuáles son los dos métodos que se utilizan para seleccionar la memoria en el microprocesador 8086?
39. Si \overline{BHE} es un 0 lógico, entonces se selecciona el banco de memoria _____.
40. Si A_0 es un 0 lógico, entonces se selecciona el banco de memoria _____.
41. ¿Por qué no se necesitan desarrollar estrobos de lectura de banco (\overline{RD}) separados cuando se conecta la memoria al 8086?
42. Modifique el circuito de la figura 10-28 de manera que la RAM se encuentre en el rango de memoria 30000H-4FFFFH.
43. Desarrolle una interfaz de memoria de 16 bits que contenga memoria SRAM en las posiciones 200000H-21FFFFH para el microprocesador 80386SX.
44. Desarrolle una interfaz de memoria de 32 bits que contenga memoria EPROM en las posiciones FFFF0000H-FFFFFFFFFFH.
45. Desarrolle una memoria de 64 bits para los microprocesadores Pentium-Pentium 4 que contenga EPROM en las posiciones FFF00000H-FFFFFFFFH y SRAM en las posiciones 00000000H-003FFFFFFH.
46. Busque en Internet la EPROM más grande que pueda encontrar. Liste su tamaño y fabricante.
47. ¿Qué es un ciclo con sólo una señal RAS?
48. ¿Puede realizarse un refresco de DRAM mientras operan otras secciones de la memoria?
49. Si una DRAM de $1\text{ M} \times 1$ requiere 4 ms para un refresco y tiene 256 filas para refrescar, no deben pasar más de _____ de tiempo antes de que se refresque otra fila.
50. Explore en Internet para encontrar la DRAM más grande que haya disponible en la actualidad.
51. Escriba un informe sobre la memoria DDR. (Sugerencia: Samsung las fabrica.)
52. Escriba un informe que detalle el funcionamiento de la RAM tipo RAMBUS. Trate de determinar por qué esta tecnología parece haber quedado obsoleta.

CAPÍTULO 11

Interfaz básica de E/S

INTRODUCCIÓN

Un microprocesador es bueno para resolver problemas, pero si no puede comunicarse con el mundo exterior, no es muy valioso. En este capítulo describiremos algunos de los métodos básicos de comunicaciones, tanto en serie como en paralelo, entre los humanos o máquinas y el microprocesador.

En este capítulo primero presentaremos la interfaz básica de E/S y hablaremos sobre la decodificación para dispositivos de E/S. Después proporcionaremos los detalles acerca de las interfaces en serie y en paralelo, las cuales tienen una variedad de aplicaciones. Para estudiar las aplicaciones conectaremos convertidores de analógico a digital y de digital a analógico al microprocesador, así como motores de CD y de velocidad gradual.

OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

1. Explicar la operación de las interfaces básicas de entrada y salida.
2. Decodificar dispositivos de E/S de 8, 16 y 32 bits para que puedan utilizarse en cualquier dirección del puerto de E/S.
3. Definir el protocolo de intercambio (handshaking) y explicar cómo utilizarlo con los dispositivos de E/S.
4. Conectar y programar la interfaz paralela programable 82C55.
5. Conectar pantallas LCD, pantallas LED, teclados, ADC, DAC y varios dispositivos más al 82C55.
6. Conectar y programar el adaptador de interfaz de comunicaciones en serie 16550.
7. Conectar y programar el temporizador de intervalo programable 8254.
8. Conectar un convertidor de analógico a digital y un convertidor de digital a analógico al microprocesador.
9. Conectar motores de CD y de velocidad gradual al microprocesador.

En esta sección explicaremos las instrucciones de E/S (IN, INS, OUT y OUTS) y las utilizaremos en aplicaciones para ejemplificar. También explicaremos el concepto de E/S aislada (algunas veces conocida como E/S directa o E/S por asignación de E/S) y E/S por asignación de memoria, las interfaces básicas de entrada y salida, y el protocolo de intercambio (handshaking). Un conocimiento práctico de estos

temas facilita la comprensión de la conexión y la operación de los componentes de la interfaz programable y de las técnicas de E/S que presentaremos en el resto de este capítulo y de este libro.

Las instrucciones de E/S

El conjunto de instrucciones contiene una instrucción que transfiere la información a un dispositivo de E/S (OUT) y otra que lee la información de un dispositivo de E/S (IN). También se proporcionan instrucciones (INS y OUTS, disponibles en todas las versiones excepto en el 8086/8088) para transferir cadenas de datos entre la memoria y un dispositivo de E/S. La tabla 11-1 lista todas las versiones de cada instrucción existente en el conjunto de instrucciones del microprocesador.

Las instrucciones que transfieren datos entre un dispositivo de E/S y el acumulador (AL, AX o EAX) del microprocesador se llaman **IN** y **OUT**. La dirección de E/S se almacena en el registro DX como una dirección de E/S de 16 bits, o como una dirección de 8 bits en el byte (p8) que sigue inmediatamente después del código de operación. Intel llama **dirección fija** a la forma de 8 bits (p8), ya que se almacena junto con la instrucción, por lo general en una ROM. La dirección de E/S de 16 bits en DX se llama **dirección variable**, ya que se almacena en DX y después se utiliza para direccionar el dispositivo de E/S. Las instrucciones INS y OUTS también utilizan DX para direccionar la E/S. Los puertos de E/S tienen ocho bits de anchura, por lo que cuando se accede a un puerto de 16 bits en realidad se direccionan dos puertos consecutivos de 8 bits. Un puerto de E/S de 32 bits está formado en realidad por cuatro puertos de 8 bits. Por ejemplo, si se accede al puerto 100H como una palabra, en realidad se accede a los puertos 100H y 101H. El puerto 100H contiene la parte menos significativa de los datos y el puerto 101H contiene la parte más significativa.

TABLA 11-1 Instrucciones de entrada/salida.

Instrucción	Anchura de datos	Función
IN AL, p8	8	Se introduce un byte en AL desde el puerto p8.
IN AX, p8	16	Se introduce una palabra en AX desde el puerto p8.
IN EAX, p8	32	Se introduce una doble palabra en EAX desde el puerto p8.
IN AL, DX	8	Se introduce un byte en AL desde el puerto direccionado por DX.
IN AX,DX	16	Se introduce una palabra en AX desde el puerto direccionado por DX.
IN EAX, DX	32	Se introduce una doble palabra en EAX desde el puerto direccionado por DX.
INSB	8	Se introduce un byte desde el puerto direccionado por DI y se almacena en la posición de memoria del segmento extra direccionada por DI, después DI = DI ± 1.
INSW	16	Se introduce una palabra desde el puerto direccionado por DI y se almacena en la posición de memoria del segmento extra direccionada por DI, después DI = DI ± 2.
INSD	32	Se introduce una doble palabra desde el puerto direccionado por DI y se almacena en la posición de memoria del segmento extra direccionada por DI, después DI = DI ± 4.
OUT p8, AL	8	Se envía un byte desde AL hacia el puerto p8.
OUT p8, AX	16	Se envía una palabra desde AX hacia el puerto p8.
OUT p8, EAX	32	Se envía una doble palabra desde EAX hacia el puerto p8.
OUT DX, AL	8	Se envía un byte desde AL hacia el puerto direccionado por DX.
OUT DX, AX	16	Se envía una palabra desde AX hacia el puerto direccionado por DX.
OUT DX, EAX	32	Se envía una doble palabra desde EAX hacia el puerto direccionado por DX.
OUTSB	8	Se envía un byte desde la posición de memoria del segmento de datos direccionada por SI hacia el puerto direccionado por DX, después SI = SI ± 1.
OUTSW	16	Se envía una palabra desde la posición de memoria del segmento de datos direccionada por SI hacia el puerto direccionado por DX, después SI = SI ± 2.
OUTSD	32	Se envía una doble palabra desde la posición de memoria del segmento de datos direccionada por SI hacia el puerto direccionado por DX, después SI = SI ± 4.

Siempre que se transfieren datos mediante el uso de la instrucción IN u OUT, la dirección de E/S (que a menudo se le conoce como **número de puerto**, o sólo puerto) aparece en el bus de direcciones. La interfaz de E/S externa decodifica el número de puerto de la misma forma que decodifica una dirección de memoria. El número de puerto fijo de 8 bits (p8) aparece en las conexiones A₇-A₀ del bus de dirección y los bits A₁₅-A₈ son iguales a 00000000₂. Las conexiones de dirección por encima de A₁₅ no están definidas para una instrucción de E/S. El número de puerto variable de 16 bits (DX) aparece en las conexiones de dirección A₁₅-A₀. Esto significa que tanto las instrucciones de E/S fijas como variables tienen acceso a las primeras 256 direcciones de puertos (00H-FFH), pero sólo la dirección de E/S variable tiene acceso a cualquier dirección de E/S de 0100H a FFFFH. En muchos sistemas dedicados sólo se decodifican los ocho bits de más a la derecha de la dirección, con lo que se reduce la cantidad de circuitos requeridos para la decodificación. En una computadora PC se decodifican los 16 bits del bus de direcciones con las posiciones 0000H-03FFH, que son las direcciones de E/S que se utilizan para las operaciones de E/S dentro de la PC en el bus ISA (**arquitectura estándar de la industria**).

Las instrucciones INS y OUTS direccionan un dispositivo de E/S mediante el uso del registro DX, pero no transfieren datos entre el acumulador y el dispositivo de E/S, como lo hacen las instrucciones IN y OUT. En vez de ello, estas instrucciones transfieren datos entre la memoria y el dispositivo de E/S. La dirección de memoria se encuentra basándose en ES:DI para la instrucción INS y basándose en DS:SI para la instrucción OUTS. Al igual que con las demás instrucciones de cadenas, el contenido de los apuntadores se incrementa o se decrementa, según lo indique el estado de la bandera de dirección (DF). Tanto INS como OUTS pueden llevar el prefijo REP, lo cual permite transferir más de un byte, una palabra o una doble palabra entre la E/S y la memoria.

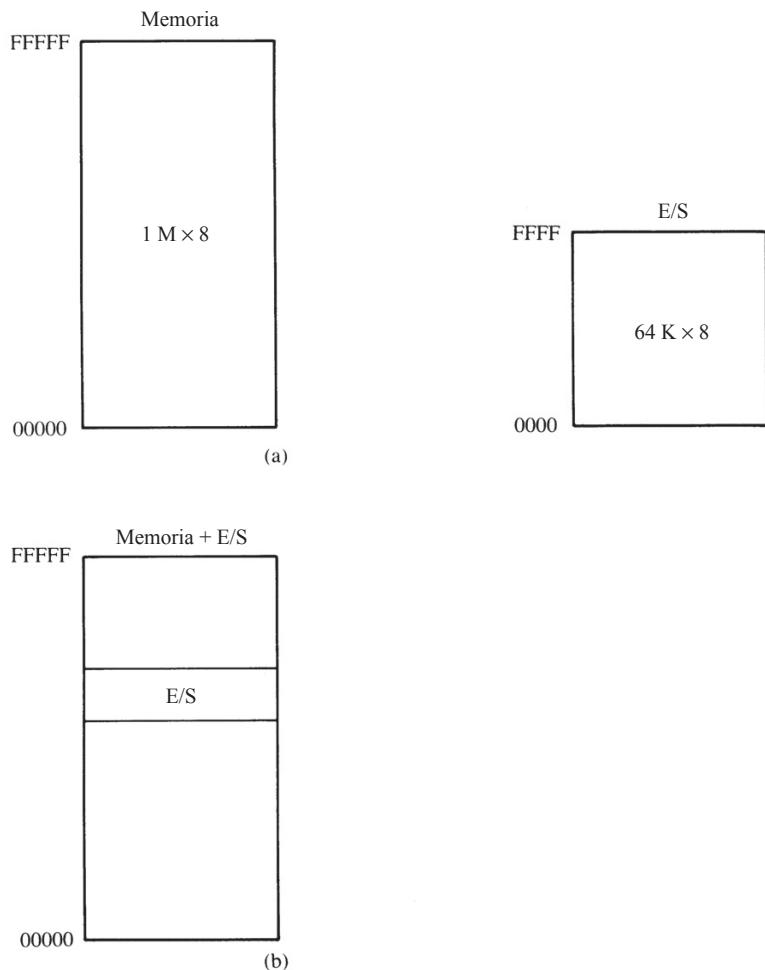
E/S aislada y E/S por asignación de memoria

Existen dos métodos para conectar la E/S con el microprocesador: **E/S aislada** y **E/S por asignación de memoria**. En el esquema de E/S aislada, las instrucciones IN, INS, OUT y OUTS transfieren datos entre el acumulador del microprocesador o la memoria y el dispositivo de E/S. En el esquema de E/S por asignación de memoria, cualquier instrucción que haga referencia a la memoria puede realizar la transferencia. Ambos métodos están en uso, por lo que hablaremos sobre ellos en este libro. La PC no utiliza la E/S por asignación de memoria.

E/S aislada. La técnica de transferencia de E/S más común que se utiliza en el sistema basado en microprocesadores Intel es la E/S aislada. El término *aislada* describe cómo se aislan las ubicaciones de E/S del sistema de memoria en un espacio de direcciones de E/S separado. (La figura 11-1 muestra los espacios de direcciones aislado y por asignación de memoria para cualquier microprocesador Intel 80X86 o Pentium-Pentium 4.) Las direcciones para los dispositivos de E/S aislados, conocidas como puertos, están separadas de la memoria. Como los puertos están separados, el usuario puede expandir la memoria a su tamaño completo sin utilizar espacio para los dispositivos de E/S. Una desventaja de la E/S aislada es que las instrucciones IN, INS, OUT y OUTS deben acceder a los datos que se transfieren entre los dispositivos de E/S y el microprocesador. Debido a esto se desarrollan señales de control separadas para el espacio de E/S (mediante el uso de M/IO y W/R), que indican una operación de lectura de E/S (IORC) o una operación de escritura de E/S (IOWC). Estas señales indican que se utiliza una dirección de puerto de E/S (que aparece en el bus de direcciones) para seleccionar el dispositivo de E/S. En la computadora personal, los puertos de E/S aislados se utilizan para controlar dispositivos periféricos. Una dirección de puerto de 8 bits se utiliza para acceder a los dispositivos ubicados en el tablero del sistema, como el temporizador y la interfaz del teclado, mientras que una dirección de puerto de 16 bits se utiliza para acceder a los puertos serial y paralelo, así como a los sistemas de vídeo y las unidades de disco.

E/S por asignación de memoria. A diferencia de la E/S aislada, la E/S por asignación de memoria no utiliza las instrucciones IN, INS, OUT u OUTS. En vez de ello utiliza cualquier instrucción que transfiera datos entre el microprocesador y la memoria. Un dispositivo de E/S por asignación de memoria se trata como una posición en el mapa de memoria. La principal ventaja de la E/S por asignación de memoria es que puede utilizarse cualquier instrucción de transferencia de memoria para acceder al dispositivo de E/S. La principal desventaja es que se utiliza una porción del sistema de memoria como mapa de E/S. Esto reduce la cantidad de memoria disponible para las aplicaciones. Otra ventaja es que las señales IORC e IOWC no tienen función en un sistema de E/S por asignación de memoria, lo cual puede reducir la cantidad de circuitos requeridos para la decodificación.

FIGURA 11-1 Los mapas de memoria y de E/S para los microprocesadores 8086/8088. (a) E/S aislada. (b) E/S por asignación de memoria.



Mapa de E/S de la computadora personal

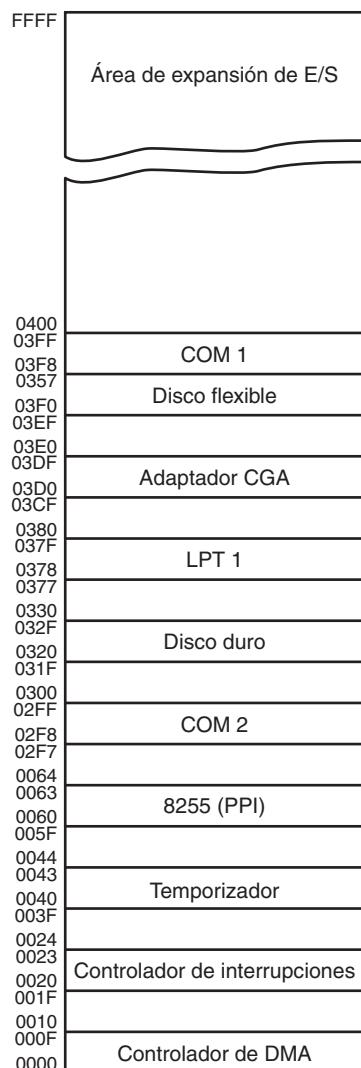
La computadora personal utiliza parte del mapa de E/S para funciones dedicadas. La figura 11-2 muestra el mapa de E/S para la PC. Observe que el espacio de E/S entre los puertos 0000H y 03FFH por lo general se reserva para el sistema computacional y el bus ISA. Los puertos de E/S que se ubican en 0400H-FFFFH por lo general están disponibles para las aplicaciones de los usuarios, las funciones del tablero principal y el bus PCI. El coprocesador aritmético 80287 utiliza las direcciones de E/S 00F8H-00FFH para las comunicaciones. Por esta razón, Intel reserva los puertos de E/S 00FOH-00FFH. Los microprocesadores del 80386 al Pentium 4 utilizan los puertos de E/S 800000F8-800000FFH para comunicarse con sus coprocesadores. Los puertos de E/S que se ubican entre 0000H y 00FFH se utilizan mediante las instrucciones de E/S de puerto fijo; los puertos que se ubican por encima de 00FFH se utilizan mediante las instrucciones de puerto de E/S variable.

Interfaces básicas de entrada y salida

El dispositivo básico de entrada es un conjunto de búferes de tres estados. El dispositivo básico de salida es un conjunto de enclavamientos de datos. El término IN (ENTRADA) se refiere a mover datos del dispositivo de E/S hacia el microprocesador, y el término OUT (SALIDA) se refiere a mover datos desde el microprocesador hacia el dispositivo de E/S.

La interfaz básica de entrada. Para construir el puerto de entrada de 8 bits que se muestra en la figura 11-3 se utilizan búferes de tres estados. Los datos TTL externos (simples interruptores de palanca en este

FIGURA 11-2 El mapa de E/S de una computadora personal, en el que se muestran muchas de las áreas de E/S fija.

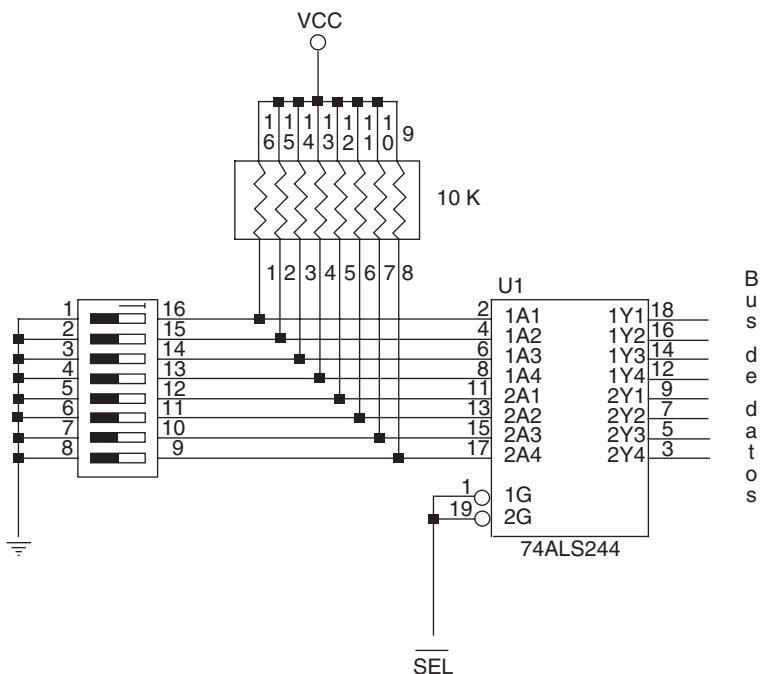


ejemplo) se conectan a las entradas de los búferes. Las salidas de los búferes se conectan al bus de datos. Las conexiones exactas del bus de datos dependen de la versión del microprocesador. Por ejemplo, el 8088 tiene las conexiones D₇-D₀ del bus de datos, los microprocesadores 80386/80486 tienen las conexiones D₃₁-D₀ y los microprocesadores Pentium-Pentium 4 tienen las conexiones D₆₃-D₀. El circuito de la figura 11-3 permite que el microprocesador lea el contenido de los ocho interruptores que se conectan con cualquier sección de 8 bits del bus de datos cuando la señal de selección SEL se vuelve un 0 lógico. Por lo tanto, siempre que se ejecuta la instrucción IN se copia el contenido de los interruptores en el registro AL.

Cuando el microprocesador ejecuta una instrucción IN, la dirección del puerto de E/S se decodifica para generar el 0 lógico en SEL. Si se coloca un 0 en las entradas de control de salida (1G y 2G) del búfer 74ALS244, las conexiones de entrada de datos (A) se conectan a las conexiones de salida de datos (Y). Si se coloca un 1 lógico en las entradas de control de salida del búfer 74ALS244, el dispositivo entra al modo de alta impedancia de tres estados que en efecto desconecta los interruptores del bus de datos.

Este circuito básico de entrada no es opcional y debe aparecer en cualquier momento en que se conectan los datos de entrada al microprocesador. Algunas veces aparece como una parte discreta del circuito, como se muestra en la figura 11-3; muchas veces está integrado a un dispositivo de E/S programable.

FIGURA 11-3 Interfaz básica de entrada que muestra la conexión de ocho interruptores. Observe que el 74ALS244 es un búfer de tres estados que controla la aplicación de los datos del interruptor al bus de datos.



También pueden conectarse datos de diecisésis o de 32 bits a varias versiones del microprocesador, pero esto no es tan común como el uso de datos de 8 bits. Para conectar 16 bits de datos, el circuito de la figura 11-3 se duplica para incluir dos búferes 74ALS244 que conecten 16 bits de datos de entrada al bus de datos de 16 bits. Para conectar 32 bits de datos, el circuito se expande sobre la base de un factor de 4.

La interfaz básica de salida. Esta interfaz recibe datos del microprocesador y por lo general debe guardarlos para cierto dispositivo externo. Es común que sus enclavamientos o flip-flops, al igual que los búferes del dispositivo de entrada, se construyan dentro del dispositivo de E/S.

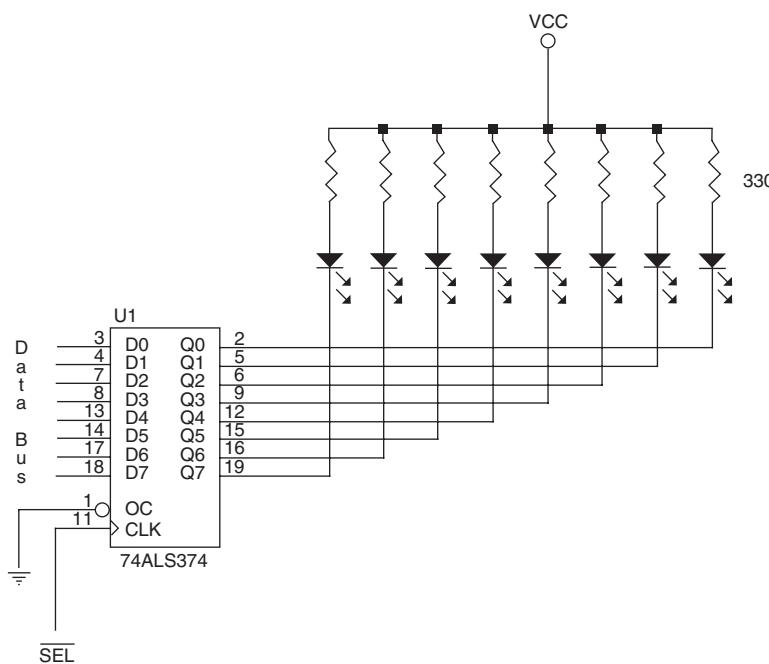
La figura 11-4 muestra cómo se conectan ocho diodos emisores de luz (LEDs) simples al microprocesador mediante un conjunto de ocho enclavamientos de datos. El enclavamiento almacena el número producido por el microprocesador desde el bus de datos, de manera que los LEDs puedan encenderse con cualquier número binario de 8 bits. Los enclavamientos se necesitan para guardar los datos, ya que cuando el microprocesador ejecuta una instrucción OUT, los datos sólo están presentes en el bus de datos durante menos de 1.0 μ s. Sin un enclavamiento, el usuario nunca vería que se iluminaran los LEDs.

Cuando se ejecuta la instrucción OUT, los datos de AL, AX o EAX se transfieren al enclavamiento mediante el bus de datos. Aquí, las entradas D de un enclavamiento octal 74ALS374 se conectan al bus de datos para capturar los datos de salida, y las salidas Q del enclavamiento se conectan a los LEDs. Cuando una salida Q se vuelve un 0 lógico, el LED se enciende. Cada vez que se ejecuta la instrucción OUT se activa la señal SEL para el enclavamiento y se captura la salida de datos que va al enclavamiento desde cualquier sección de 8 bits del bus de datos. Los datos se guardan hasta que se ejecuta la siguiente instrucción OUT. Por lo tanto, siempre que se ejecuta la instrucción de salida en este circuito aparecen los datos del registro AL en los LEDs.

Protocolo de intercambio

Muchos dispositivos de E/S aceptan o liberan información a una velocidad mucho menor que la del microprocesador. Existe otro método de control de E/S, llamado **protocolo de intercambio (handshaking)** o **sondeo (polling)**, que sincroniza el dispositivo de E/S con el microprocesador. La impresora paralela es un ejemplo de un dispositivo que requiere el protocolo de intercambio; este dispositivo imprime sólo unos cuantos cientos de caracteres por segundo (CPS), por lo que es necesario desarrollar una manera de reducir la velocidad del microprocesador para que concuerde con la de la impresora.

FIGURA 11-4 La interfaz básica de salida conectada a una serie de pantallas LED.



La figura 11-5 muestra las conexiones típicas de entrada y salida de una impresora. Aquí los datos se transfieren a través de una serie de conexiones de datos (D₇-D₀). BUSY indica que la impresora está ocupada. STB es un pulso de reloj que se utiliza para enviar datos a la impresora.

Los datos ASCII que la impresora va a imprimir se colocan en D₇-D₀ y se aplica un pulso a la conexión STB. La señal de estrobo envía los datos a la impresora para que puedan imprimirse. Tan pronto como la impresora recibe los datos coloca un 1 lógico en la terminal BUSY, indicando que está ocupada imprimiendo datos. El software del microprocesador sondea o evalúa la terminal BUSY para decidir si la impresora está ocupada o no. Si está ocupada, el microprocesador espera; si no está ocupada, el microprocesador envía el siguiente carácter ASCII a la impresora. El ejemplo 11-1 muestra un procedimiento simple que evalúa la bandera BUSY y después envía datos a la impresora si no está ocupada. Aquí, el procedimiento IMPRIME imprime el contenido en código ASCII de BL sólo si la bandera BUSY es un 0 lógico, lo que indica que la impresora no está ocupada. Este procedimiento se llama cada vez que se va a imprimir un carácter.

EJEMPLO 11-1

```
;Un procedimiento en lenguaje ensamblador que imprime el contenido ASCII de BL.

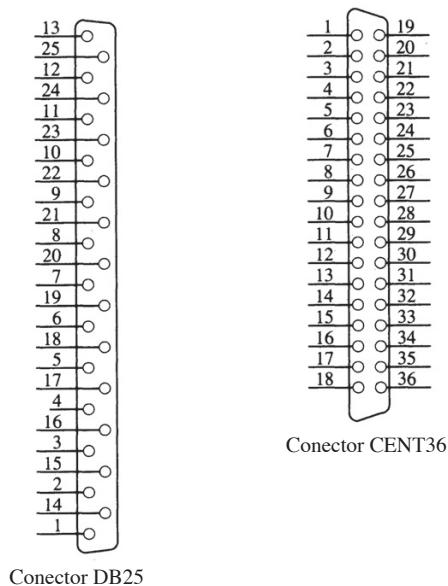
IMPRIME PROC    NEAR

    .REPEAT           ;evalúa la bandera BUSY
        IN AL,BUSY
        TEST AL,BUSY_BIT
    .UNTIL ZERO
        MOV AL,BL          ;posiciona los datos en AL
        OUT PRINTER,AL      ;imprime los datos
        RET

IMPRIME ENDP
```

Notas sobre el uso de interfaces en los circuitos

Una parte del uso de interfaces requiere cierto conocimiento de electrónica. En esta porción de la introducción al uso de interfaces examinaremos algunas de las diversas facetas de las interfaces electrónicas.



DB25 Número de terminal	CENT36 Número de terminal	Función	DB25 Número de terminal	CENT36 Número de terminal	Función
1	1	Estrobo de datos	12	12	Sin papel
2	2	Datos 0 (D0)	13	13	Selección
3	3	Datos 1 (D1)	14	14	Afd
4	4	Datos 2 (D2)	15	32	<u>Error</u>
5	5	Datos 3 (D3)	16	—	<u>RESET</u>
6	6	Datos 4 (D4)	17	31	Selección entrada
7	7	Datos 5 (D5)	18–25	19–30	Tierra
8	8	Datos 6 (D6)	—	17	Tierra de estructura
9	9	Datos 7 (D7)	—	16	Tierra
10	10	<u>Reconocido</u>	—	33	Tierra
11	11	Ocupado			

FIGURA 11-5 El conector DB25 que se encuentra en las computadoras y el conector Centronics de 36 terminales que se encuentra en las impresoras para la interfaz de impresora paralela Centronics.

Antes de que un circuito o dispositivo pueda conectarse al microprocesador, deben conocerse las características terminales del microprocesador y de sus componentes de interfaz asociados. (Este tema se presentó al principio del capítulo 9.)

Dispositivos de entrada. Los dispositivos de entrada ya son TTL y compatibles, por lo que pueden conectarse al microprocesador y a sus componentes de interfaz, o están basados en interruptores. La mayoría de los dispositivos basados en interruptores están abiertos o conectados. Éstos no son niveles TTL; los niveles TTL son un 0 lógico (0.0 V-0.8 V) o un 1 lógico (2.0 V-5.0 V).

Para que un dispositivo basado en interruptores pueda usarse como dispositivo de entrada compatible con TTL, debe aplicarse cierto acondicionamiento. La figura 11-6 muestra un interruptor de palanca simple que está conectado de forma correcta para funcionar como un dispositivo de entrada. Observe que se utiliza una resistencia elevadora para asegurar que cuando el interruptor esté abierto, la señal de salida sea un 1 lógico; cuando el interruptor esté cerrado se conecta a tierra, lo que produce un nivel 0 lógico válido. El valor de la resistencia elevadora no es crucial; simplemente asegura que

FIGURA 11-6 Un interruptor unipolar de 1 dirección conectado como dispositivo TTL.

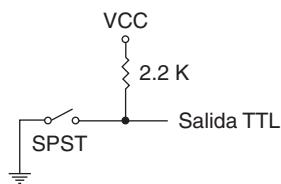
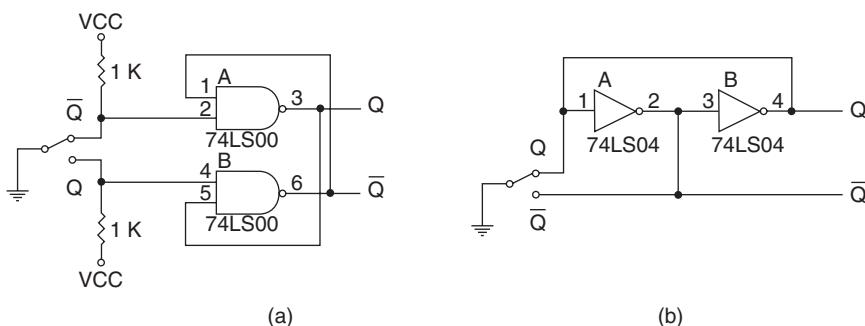


FIGURA 11-7 Contactos de interruptores para eliminar rebotes: (a) eliminación convencional de rebotes y (b) eliminación práctica de rebotes.



la señal esté en un nivel de 1 lógico. Por lo general, un rango de valores estándar para las resistencias elevadoras está entre $1\text{ K}\Omega$ y $10\text{ K}\Omega$.

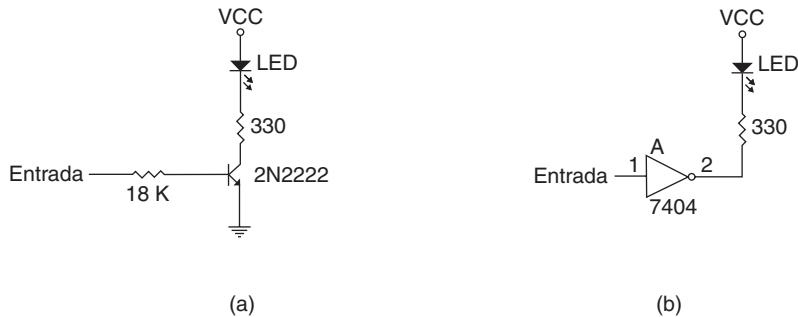
Los contactos de los interruptores mecánicos tienen un rebote físico cuando se cierran, lo que puede provocar un problema si se utiliza un interruptor como una señal de reloj para un circuito digital. Para evitar problemas con los rebotes, puede construirse uno de los dos circuitos que se muestran en la figura 11-7. El primer circuito (a) es un eliminador de rebotes clásico de un libro de texto; el segundo (b) es una versión más práctica del mismo circuito. Como la primera versión es más costosa de construir, en la práctica se utilizaría la segunda versión ya que no requiere resistencias elevadoras y sólo utiliza dos inversores en lugar de las compuertas NAND.

Tal vez haya notado que ambos circuitos de la figura 11-7 son flip-flops asíncronos. El circuito de la figura (b) funciona de la siguiente manera: suponga que el interruptor se encuentra en la posición \bar{Q} . Si se mueve hacia Q pero todavía no llega, la salida Q del circuito es un 0 lógico. Los inversores recuerdan ese estado de 0 lógico. La salida del inversor B se conecta a la entrada del inversor A. Como la salida del inversor B es un 0 lógico, la salida del inversor A es un 1 lógico. La salida de 1 lógico el inversor A mantiene la salida de 0 lógico del inversor B. El flip-flop permanece en este estado hasta que el contacto del interruptor móvil llega primero a la conexión Q . Tan pronto como la entrada Q del interruptor se convierte en un 0 lógico, cambia el estado del flip-flop. Si el contacto rebota alejándose de la entrada Q , el flip-flop recuerda y no ocurre ningún cambio, con lo que se elimina cualquier rebote.

Dispositivos de salida. Los dispositivos de salida son mucho más diversos que los de entrada, pero muchos se conectan de manera uniforme. Antes de poder conectar cualquier dispositivo de salida, debemos entender cuáles son los voltajes y corrientes del microprocesador o de un componente de interfaz TTL. Los voltajes son compatibles con TTL del microprocesador o del elemento de interfaz. (0 lógico = 0.0 V a 0.4 V; 1 lógico = 2.4 V a 5.0 V.) Las corrientes para un microprocesador y para muchos componentes de interfaz del microprocesador son menores que para los componentes TTL estándar. (0 lógico = 0.0 a 2.0 mA; 1 lógico = 0.0 a 400 μA .)

Una vez que se conocen las corrientes de salida puede conectarse un dispositivo a una de las salidas. La figura 11-8 muestra cómo conectar un simple LED a una terminal periférica del microprocesador. Observe que en la figura 11-8(a) se utiliza un transistor como controlador y en la figura 11-8(b) se utiliza un inversor TTL. Este inversor (versión estándar) proporciona hasta 16 mA de corriente a un nivel de 0 lógico, lo cual es más que suficiente para controlar un LED estándar. Este tipo de LED requiere 10 mA de corriente de polarización directa para encenderse. En ambos circuitos suponemos que la caída de voltaje a través del LED es de aproximadamente 2.0 V. La hoja técnica para un LED indica que la caída nominal es de 1.65 V, pero se sabe por experiencia que la caída está entre 1.5 V y 2.0 V.

FIGURA 11-8 Conexión de un LED: (a) mediante el uso de un transistor y (b) mediante el uso de un inversor.



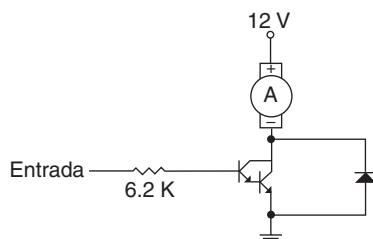
Esto significa que el valor de la resistencia compensadora es de $3.0\text{ V} \div 10\text{ mA}$, o de $300\text{ }\Omega$. Ya que $300\text{ }\Omega$ no es un valor de resistencia estándar (el menor costo), se seleccionó una resistencia de $330\text{ }\Omega$ para esta interfaz.

En el circuito de la figura 11-8(a) optamos por usar un transistor de conmutación en lugar del búfer TTL. El 2N2222 es un buen transistor de conmutación de bajo costo y de propósito general que tiene una ganancia mínima de 100. En este circuito la corriente de colector es de 10 mA , por lo que la corriente de base será $1/100$ parte de la corriente de colector, o 0.1 mA . Para determinar el valor de la resistencia compensadora de la base se utiliza la corriente de base de 0.1 mA y una caída de voltaje de 1.7 V a través de la resistencia compensadora de la base. La señal de entrada TTL tiene un valor mínimo de 2.4 V y la caída a través de la unión emisor-base es de 0.7 V . La diferencia es de 1.7 V , que es la caída de voltaje a través de la resistencia. El valor de la resistencia es $1.7\text{ V} \div 0.1\text{ mA}$, o $17\text{ K}\Omega$. Ya que $17\text{ K}\Omega$ no es un valor estándar, se seleccionó una resistencia de $18\text{ K}\Omega$.

Suponga que necesitamos conectar un motor de 12 V CD al microprocesador y que la corriente del motor es de 1 A . Es obvio que no podemos utilizar un inversor TTL por dos razones: la señal de 12 V quemaría el inversor y la cantidad de corriente es mucho mayor que la corriente máxima de 16 mA del inversor. Tampoco podemos usar un transistor 2N2222 ya que la cantidad máxima de corriente es de 250 mA a 500 mA , dependiendo del estilo de paquete seleccionado. La solución es utilizar un par Darlington.

La figura 11-9 muestra un motor conectado al par Darlington. Este par tiene una ganancia máxima de corriente de 7000 y una corriente máxima de 4 A . El valor de la resistencia de polarización se calcula de la misma forma que la que se utilizó en el controlador de LED. La corriente a través de la resistencia es de $1.0\text{ A} \div 7000$, lo que nos da un valor aproximado de 0.143 mA . La caída de voltaje a través de la resistencia es de 0.9 V , debido a las dos caídas de los dos diodos (uniones base/emisor) en vez de una. El valor de la resistencia de polarización es de $0.9\text{ V} \div 0.143\text{ mA}$, o $6.29\text{ K}\Omega$. En el circuito se utiliza el valor estándar de $6.2\text{ K}\Omega$. El par Darlington debe utilizar un dissipador de calor debido a la cantidad de corriente que pasa a través de este componente, y el diodo también debe estar presente para evitar que el par Darlington se destruya debido a la inversión de giro inductiva del motor. Este circuito también se utiliza para conectar relevadores mecánicos o casi cualquier dispositivo que requiera una gran cantidad de corriente, o un cambio en el voltaje.

FIGURA 11-9 Un motor de CD conectado a un sistema mediante el uso de un par Darlington.



11-2**DECODIFICACIÓN DE DIRECCIONES DE PUERTO DE E/S**

La decodificación de direcciones de puerto de E/S es muy similar a la decodificación de direcciones de memoria, en especial para los dispositivos de E/S por asignación de memoria. De hecho, no hablaremos sobre la decodificación de E/S por asignación de memoria ya que se trata de la misma manera que la memoria (excepto que no se utilizan \overline{IORC} e \overline{IOWC} ya que no hay instrucción IN u OUT). La decisión para usar la E/S de asignación de memoria se determina frecuentemente por el tamaño de la memoria del sistema y la localización del dispositivo de entrada del sistema.

La principal diferencia entre una memoria decodificada y una E/S aislada decodificada es el número de terminales de dirección conectadas al decodificador. Para la memoria decodificamos las terminales $A_{31}-A_0$, $A_{23}-A_0$ o $A_{19}-A_0$ y para la E/S aislada decodificamos las terminales $A_{15}-A_0$. Algunas veces, si los dispositivos de E/S sólo utilizan direccionamiento de E/S fijo, decodificamos las terminales A_7-A_0 . En el sistema de computadora personal siempre decodificamos todos los 16 bits de la dirección de puerto de E/S. Otra diferencia con la E/S aislada es que \overline{IORC} e \overline{IOWC} activan los dispositivos de E/S para una operación de lectura o de escritura. En versiones anteriores del microprocesador, $IO/\overline{M} = 1$ y \overline{RD} o \overline{WR} se utilizan para activar los dispositivos de E/S. En las versiones más recientes del microprocesador, $M/\overline{IO} = 0$ y W/\overline{R} se combinan y utilizan para activar los dispositivos de E/S.

Decodificación de direcciones de puerto de E/S de 8 bits

Como se dijo antes, la instrucción de E/S fija utiliza una dirección de puerto de E/S de 8 bits que aparece en las terminales $A_{15}-A_0$ como 0000H-00FFH. Si un sistema nunca va a tener más de 256 dispositivos de E/S, por lo general sólo decodificamos las conexiones de dirección A_7-A_0 para una dirección de puerto de E/S de 8 bits. Por lo tanto, ignoramos las conexiones de dirección $A_{15}-A_8$. A menudo los sistemas integrados utilizan direcciones de puertos de 8 bits. Tenga en cuenta que el registro DX también puede direccionar los puertos de E/S 00H-FFH. Si la dirección se decodifica como una dirección de 8 bits, no podemos incluir dispositivos de E/S que utilicen una dirección de E/S de 16 bits. La computadora personal nunca utiliza o decodifica una dirección de 8 bits.

La figura 11-10 muestra un decodificador 74ALS138 que decodifica los puertos de E/S de 8 bits del F0H al F7H. (Nos basamos en la suposición de que este sistema sólo utilizará los puertos de E/S 00H-FFH para este ejemplo de decodificador.) Este decodificador es idéntico a un decodificador de direcciones de memoria, con la excepción de que sólo conectamos los bits de dirección A_7-A_0 a las entradas del decodificador. La figura 11-11 muestra la versión PLD, en la que se utiliza un GAL22V10 como decodificador. El PLD es un mejor circuito decodificador debido a que el número de circuitos integrados se ha reducido a un dispositivo. En el ejemplo 11-2 aparece el programa en VHDL para el PLD.

EJEMPLO 11-2

```
-- Código de VHDL para el decodificador de la figura 11-11

library ieee;
use ieee.std_logic_1164.all;

entity DECODIFICADOR_11_11 is
port (
    A7, A6, A5, A4, A3, A2, A1, A0: in STD_LOGIC;
    D0, D1, D2, D3, D4, D5, D6, D7: out STD:LOGIC
);
end;

architecture V1 of DECODIFICADOR_11_11 is
begin
    D0 <= not( A7 and A6 and A5 and A4 and not A3 and not A2 and not A1 and
not A0 );
    D1 <= not( A7 and A6 and A5 and A4 and not A3 and not A2 and not A1 and
A0 );

```

FIGURA 11-10 Un decodificador de puertos que decodifica los puertos de E/S de 8 bits. Este decodificador genera salidas activas en nivel bajo para los puertos F0H-F7H.

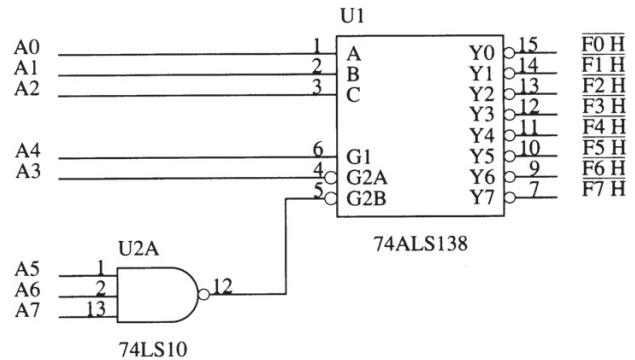
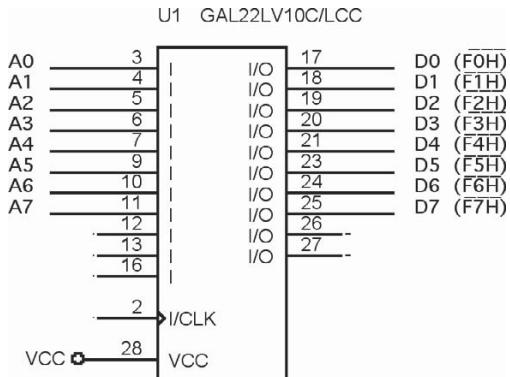


FIGURA 11-11 Un PLD que genera las señales de selección de puerto F0H-F7H.



```

D2 <= not( A7 and A6 and A5 and A4 and not A3 and not A2 and A1 and
            not A0 );
D3 <= not( A7 and A6 and A5 and A4 and not A3 and not A2 and A1 and
            A0 );
D4 <= not( A7 and A6 and A5 and A4 and not A3 and A2 and not A1 and
            not A0 );
D5 <= not( A7 and A6 and A5 and A4 and not A3 and A2 and not A1 and
            A0 );
D6 <= not( A7 and A6 and A5 and A4 and not A3 and A2 and A1 and
            not A0 );
D7 <= not( A7 and A6 and A5 and A4 and not A3 and A2 and A1 and
            A0 );
end V1;

```

Decodificación de direcciones de puerto de E/S de 16 bits

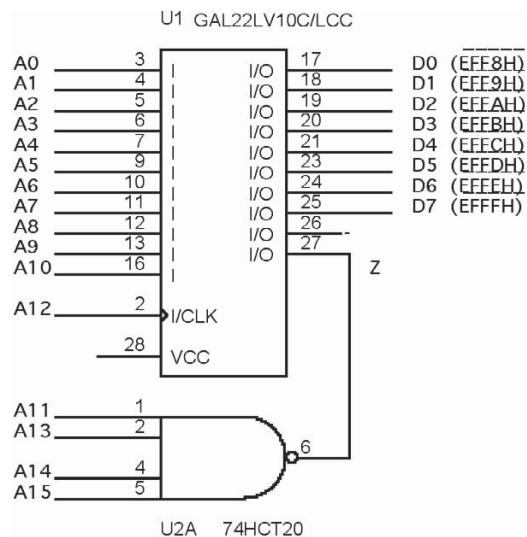
Por lo general, los sistemas de computadora personal utilizan direcciones de E/S de 16 bits. Es muy raro encontrar direcciones de puertos de 16 bits en sistemas integrados. La principal diferencia entre la decodificación de una dirección de E/S de 8 bits y una dirección de E/S de 16 bits es que deben decodificarse ocho líneas de dirección adicionales (A_{15} - A_8). La figura 11-12 muestra un circuito que contiene un PLD y una compuerta NAND de cuatro entradas para decodificar los puertos de E/S EFF8H-EFFFH.

La compuerta NAND decodifica parte de la dirección (A_{15} , A_{14} , A_{13} y A_{11}) debido a que el PLD no tiene suficientes entradas de dirección. La salida de la compuerta NAND se conecta a la entrada Z del PLD y se decodifica como parte de la dirección de puerto de E/S. El PLD genera estrobo de dirección para los puertos de E/S EFF8H-EFFFH. En el ejemplo 11-3 se muestra el programa para el PLD.

EJEMPLO 11-3

```
-- Código de VHDL para el decodificador de la figura 11-12
library ieee;
```

FIGURA 11-12 Un PLD que decodifica los puertos de E/S de 16 bits del EFF8H al EFFFH.



```

use ieee.std_logic_1164.all;

entity DECODIFICADOR_11_12 is
port (
    Z, A12, A10, A9, A8, A7, A6, A5, A4, A3, A2, A1, A0: in STD_LOGIC;
    D0, D1, D2, D3, D4, D5, D6, D7: out STD:LOGIC
);
end;

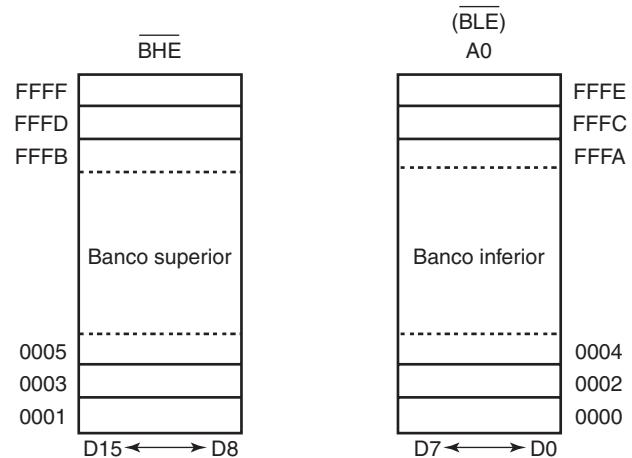
architecture V1 of DECODIFICADOR_11_12 is
begin
    D0 <= not( not Z and not A12 and A10 and A9 and A8 and A7 and A6 and A5
                and A4 and A3 and not A2 and not A1 and not A0 );
    D1 <= not( not Z and not A12 and A10 and A9 and A8 and A7 and A6 and A5
                and A4 and A3 and not A2 and not A1 and A0 );
    D2 <= not( not Z and not A12 and A10 and A9 and A8 and A7 and A6 and A5
                and A4 and A3 and not A2 and A1 and not A0 );
    D3 <= not( not Z and not A12 and A10 and A9 and A8 and A7 and A6 and A5
                and A4 and A3 and not A2 and A1 and A0 );
    D4 <= not( not Z and not A12 and A10 and A9 and A8 and A7 and A6 and A5
                and A4 and A3 and A2 and not A1 and not A0 );
    D5 <= not( not Z and not A12 and A10 and A9 and A8 and A7 and A6 and A5
                and A4 and A3 and A2 and not A1 and A0 );
    D6 <= not( not Z and not A12 and A10 and A9 and A8 and A7 and A6 and A5
                and A4 and A3 and A2 and A1 and not A0 );
    D7 <= not( not Z and not A12 and A10 and A9 and A8 and A7 and A6 and A5
                and A4 and A3 and A2 and A1 and A0 );
end V1;

```

Puertos de E/S de 8 y de 16 bits

Ahora que comprende el funcionamiento de los puertos de E/S y que aprendió que es probable que sea más simple decodificar una dirección de puerto de E/S que una dirección de memoria (debido al número de bits), puede explicar cómo conectar el microprocesador con los dispositivos de E/S de 8 y de 16 bits. En un microprocesador de 16 bits tal como el 80386SX, los datos que se transfieren a un dispositivo de E/S de 8 bits existen en uno de los bancos de E/S. Hay 64 K puertos de 8 bits distintos pero sólo 32 K puertos de 16 bits distintos, ya que un puerto de 16 bits utiliza dos puertos de 8 bits. El

FIGURA 11-13 Los bancos de E/S que se encuentran en los microprocesadores 8086, 80186, 80286 y 80386SX.



sistema de E/S en un microprocesador de este tipo contiene dos bancos de memoria de 8 bits, al igual que la memoria. Esto se muestra en la figura 11-13, la cual muestra los bancos de E/S separados para un sistema de 16 bits tal como el 80386SX.

Como existen dos bancos de E/S, cualquier operación de escritura de E/S de 8 bits requiere un estrobo de escritura separado para funcionar en forma correcta. Las operaciones de lectura de E/S no requieren estrobos de lectura separados. Al igual que con la memoria, el microprocesador lee sólo el byte que espera e ignora el otro byte. La única situación en la que una operación de lectura puede provocar problemas es cuando el dispositivo de E/S responde en forma incorrecta a una operación de lectura. En el caso de un dispositivo de E/S que responde a una lectura del banco incorrecto, tal vez haya que incluir señales de lectura separadas. Más adelante hablaremos sobre este caso.

La figura 11-4 ilustra un sistema que contiene dos dispositivos de salida de 8 bits distintos, los cuales están en las direcciones de E/S de 8 bits 40H y 41H. Como éstos son dispositivos de 8 bits y aparecen en distintos bancos de E/S, se generan señales de escritura de E/S separadas para activar un par de enclavamientos que capturan los datos de los puertos. Todos los puertos de E/S de 8 bits utilizan direcciones de 8 bits. En consecuencia, los puertos 40H y 41H pueden direccionarse como puertos de 8 bits separados, o en conjunto como un puerto de 16 bits. En el ejemplo 11-4 se ilustra el programa para el decodificador PLD que se utiliza en la figura 11-4.

EJEMPLO 11-4

```
-- Código de VHDL para el decodificador de la figura 11-14

library ieee;
use ieee.std_logic_1164.all;

entity DECODIFICADOR_11_14 is
port (
    BHE, IOWC, A7, A6, A5, A4, A3, A2, A1, A0: in STD_LOGIC;
    D0, D1: out STD:LOGIC
);
end;

architecture V1 of DECODIFICADOR_11_14 is
begin
    D0 <= BHE or IOWC or A7 or not A6 or A5 or A4 or A3 or A2 or A1 or A0;
    D1 <= BHE or IOWC or A7 or not A6 or A5 or A4 or A3 or A2 or A1 or
        not A0;
end V1;
```

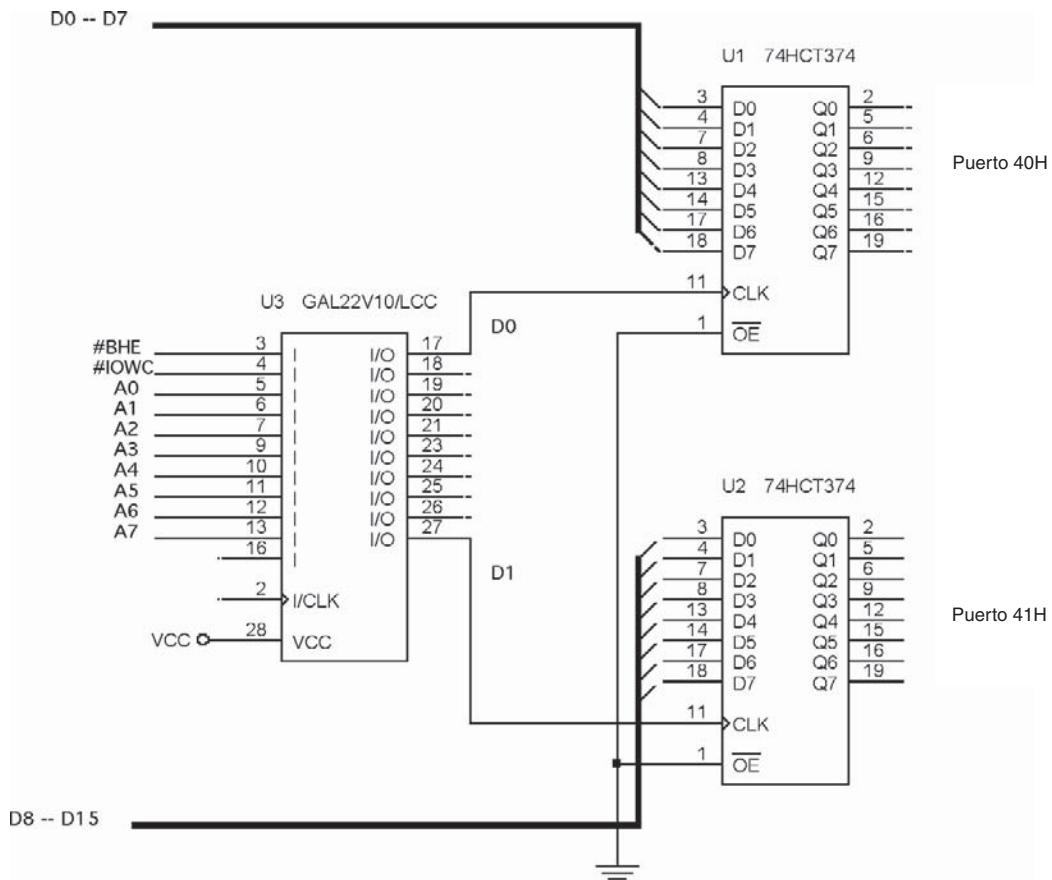


FIGURA 11-14 Un decodificador de puertos de E/S que selecciona los puertos 40H y 41H para los datos de salida.

Cuando se seleccionan dispositivos de E/S de 16 bits, las terminales \overline{BLE} (A_0) y \overline{BHE} no tienen función ya que ambos bancos de E/S se seleccionan al mismo tiempo. Aunque los dispositivos de E/S de 16 bits se utilizan raras veces, existen algunos para los convertidores de analógico a digital y de digital a analógico, así como para algunas interfaces de vídeo y de disco.

La figura 11-15 muestra un dispositivo de entrada de 16 bits conectado para funcionar en las direcciones de E/S de 8 bits 64H y 65H. Observe que el decodificador PLD no tiene una conexión para direccionar los bits \overline{BLE} (A_0) y BHE , ya que estas señales no se aplican a los dispositivos de E/S de 16 bits. El programa para el PLD que se muestra en el ejemplo 11-5 muestra cómo se generan las señales de habilitación para los búferes de tres estados (74HCT244) que se utilizan como dispositivos de entrada.

EJEMPLO 11-5

```
-- Código de VHDL para el decodificador de la figura 11-15
library ieee;
use ieee.std_logic_1164.all;

entity DECODIFICADOR_11_15 is
port (
    IORC, A7, A6, A5, A4, A3, A2, A1: in STD_LOGIC;
    D0: out STD_LOGIC
);
```

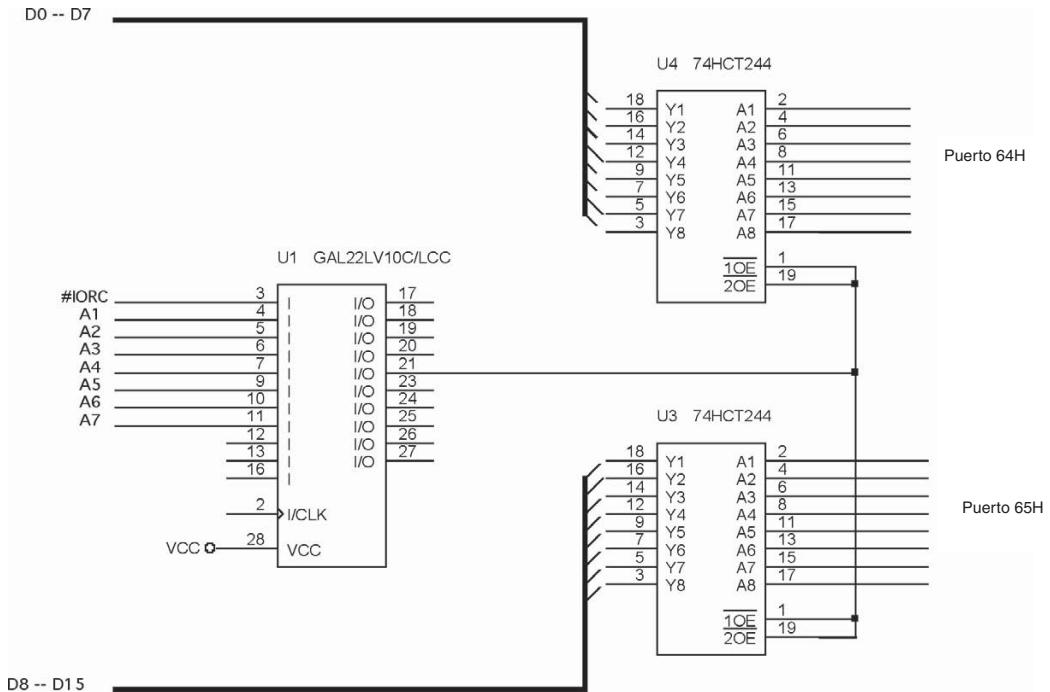


FIGURA 11-15 Un puerto de 16 bits decodificado en las direcciones de E/S 64H y 65H.

```

end;

architecture V1 of DECODIFICADOR_11_15 is
begin
    D0 <= IORC or A7 or not A6 or not A5 or A4 or A3 or not A2 or A1;
end V1;

```

Puertos de E/S de 32 bits

Aunque los puertos de E/S de 32 bits no son comunes, es probable que vayan a serlo debido a los buses más recientes que se están utilizando en los sistemas computacionales. El bus de sistema EISA (que alguna vez fue prometedor) soporta E/S de 32 bits, así como el bus local VESA y el bus PCI actual, pero no muchos dispositivos de E/S son de 32 bits.

El circuito de la figura 11-16 ilustra un puerto de entrada de 32 bits para los microprocesadores del 80386DX al 80486DX. Al igual que en las interfaces anteriores, este circuito utiliza un solo PLD para decodificar los puertos de E/S y cuatro búferes 74HCT244 para conectar los datos de E/S al bus de datos. Esta interfaz decodifica los puertos de E/S de 8 bits 70H-73H, como se indica mediante el programa de PLD en el ejemplo 11-6. De nuevo, sólo decodificamos una dirección de puerto de E/S de 8 bits. Cuando se escribe software para acceder a este puerto, es imprescindible utilizar la dirección 70H para la entrada de 32 bits, como en la instrucción IN EAX, 70H.

EJEMPLO 11-6

```

-- Código de VHDL para el decodificador de la figura 11-16

library ieee;
use ieee.std_logic_1164.all;

entity DECODIFICADOR_11_16 is

```

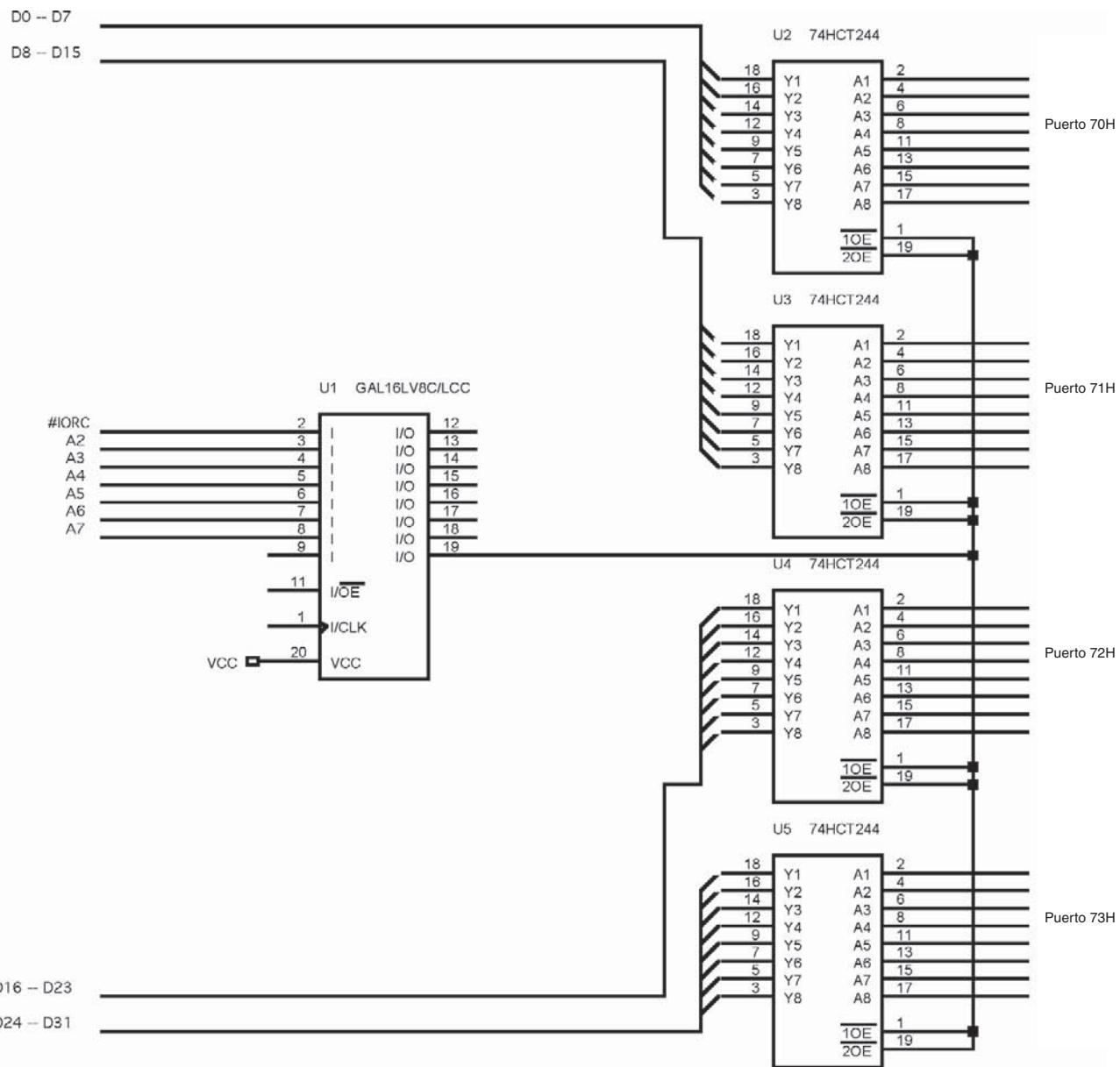


FIGURA 11-16 Un puerto de 32 bits decodificado en las direcciones 70H a 30H para el microprocesador 80486DX.

```

port (
    IORC, A7, A6, A5, A4, A3, A2: in STD_LOGIC;
    D0: out STD:LOGIC
);
begin
    architecture V1 of DECODIFICADOR_11_16 is
        begin
            D0 <= IORC or A7 or not A6 or not A5 or not A4 or A3 or A2;
    end V1;

```

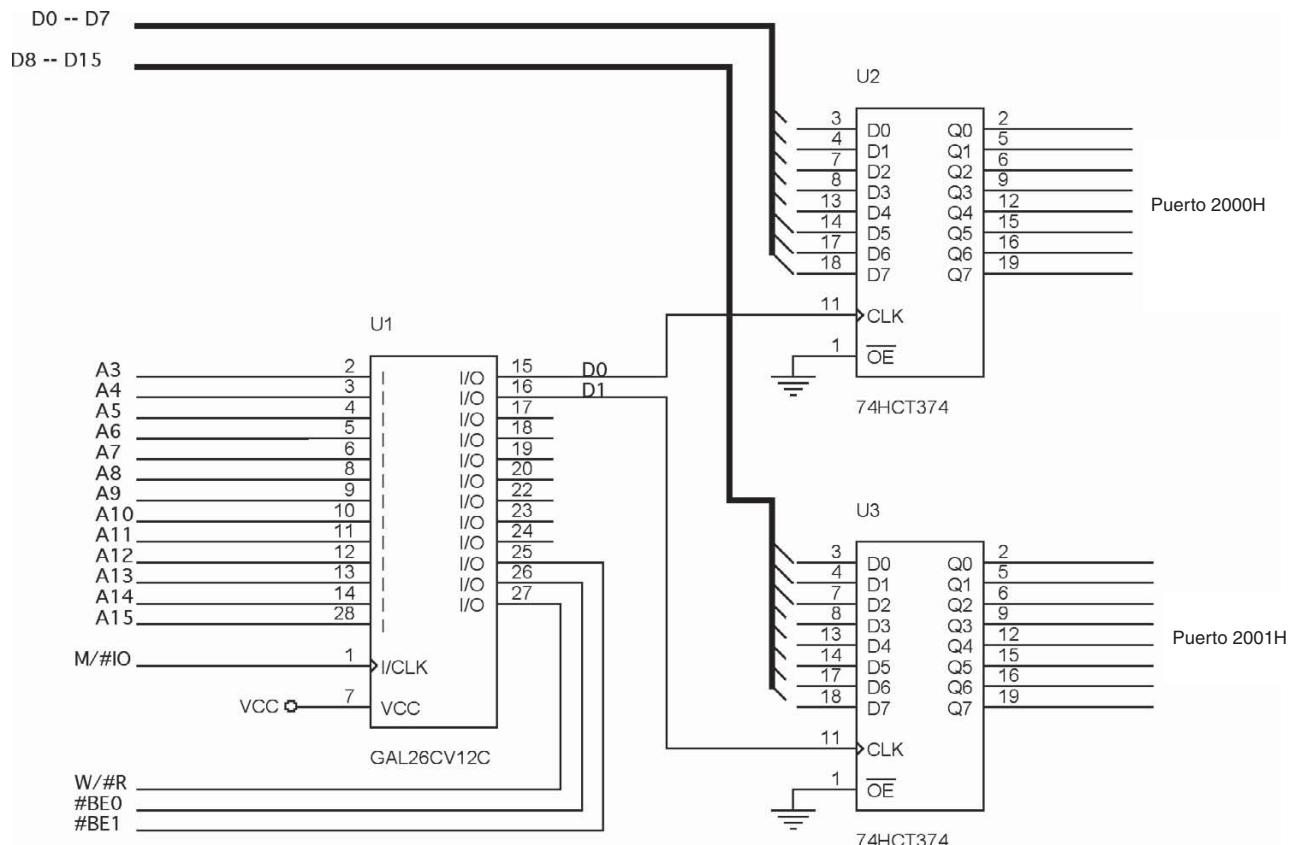


FIGURA 11-17 Un Pentium 4 conectado a un puerto de E/S de 16 bits en las direcciones de puerto 2000H y 2001H.

En los microprocesadores Pentium-Pentium 4 y sus buses de datos de 64 bits, los puertos de E/S aparecen en varios bancos, basándose en lo que determine la dirección de puerto de E/S. Por ejemplo, el puerto de E/S de 8 bits 0034H aparece en el banco 4 de E/S del Pentium, mientras que los puertos de E/S de 16 bits 0034H-0035H aparecen en los bancos 4 y 5 del Pentium. Un acceso de E/S de 32 bits en el sistema Pentium puede aparecer en 4 bancos de E/S consecutivos cualesquiera. Por ejemplo, los puertos de E/S de 32 bits 0100H-0103H aparecen en los bancos 0-3. ¿Cómo se conecta un dispositivo de E/S de 64 bits? Las transferencias de E/S más amplias son de 32 bits, además en la actualidad no hay instrucciones de E/S de 64 bits para soportar transferencias de 64 bits.

Suponga que necesita una interfaz para un puerto simple de salida de 16 bits en las direcciones de puerto de E/S 2000H y 2001H. Los tres bits de más a la derecha de la dirección de puerto inferior son 000 para el puerto 2000H. Esto significa que el puerto 2000H se encuentra en el banco de memoria 0. De igual forma, los tres bits binarios de más a la derecha del puerto de E/S 2001H son 001, lo que significa que el puerto 2001H está en el banco 1. En la figura 11-7 se muestra una interfaz y en el ejemplo 11-7 se muestra el programa de PLD.

Las señales de control M/ \overline{IO} y W/ \overline{R} deben combinarse para generar una señal de escritura de E/S para los enclavamientos, y las señales de habilitación de banco BE0 y BE1 deben utilizarse para guiar la señal de escritura hacia el ciclo de reloj del enclavamiento correcto para las direcciones 2000H (banco 0) y 2001H (banco 1). El único problema que puede surgir en la interfaz es cuando el puerto de E/S se extiende sobre un límite de 64 bits; por ejemplo, un puerto de 16 bits ubicado en las direcciones 2007H y 2008H. En este caso, el puerto 2007H utiliza el banco 7 y el puerto 2008H utiliza el banco 0, pero la dirección que se decodifica es distinta para cada ubicación: se decodifica 0010 0000 0000 0XXX para 2007H y 0010 0000 0000 1XXX para 2008H. Probablemente sea mejor evitar situaciones como ésta.

EJEMPLO 11-7

```
-- Código de VHDL para el decodificador de la figura 11-17

library ieee;
use ieee.std_logic_1164.all;

entity DECODIFICADOR_11_17 is

port (
    MIO, BE0, BE1, WR, A15, A14, A13, A12, A11, A10, A9, A8, A7, A6, A5, A4,
    A3: in STD_LOGIC;
    D0, D1: out STD:LOGIC
);

end;

architecture V1 of DECODIFICADOR_11_17 is

begin

D0 <= MIO or BE0 or not WR or A15 or A14 or not A13 or A12 or A11 or A10
or A9 or A8 or A7 or A6 or A5 or A4 or A3;
D1 <= MIO or BE1 or not WR or A15 or A14 or not A13 or A12 or A11 or A10
or A9 or A8 or A7 or A6 or A5 or A4 or not A3;

end V1;
```

11-3**LA INTERFAZ PERIFÉRICA PROGRAMABLE**

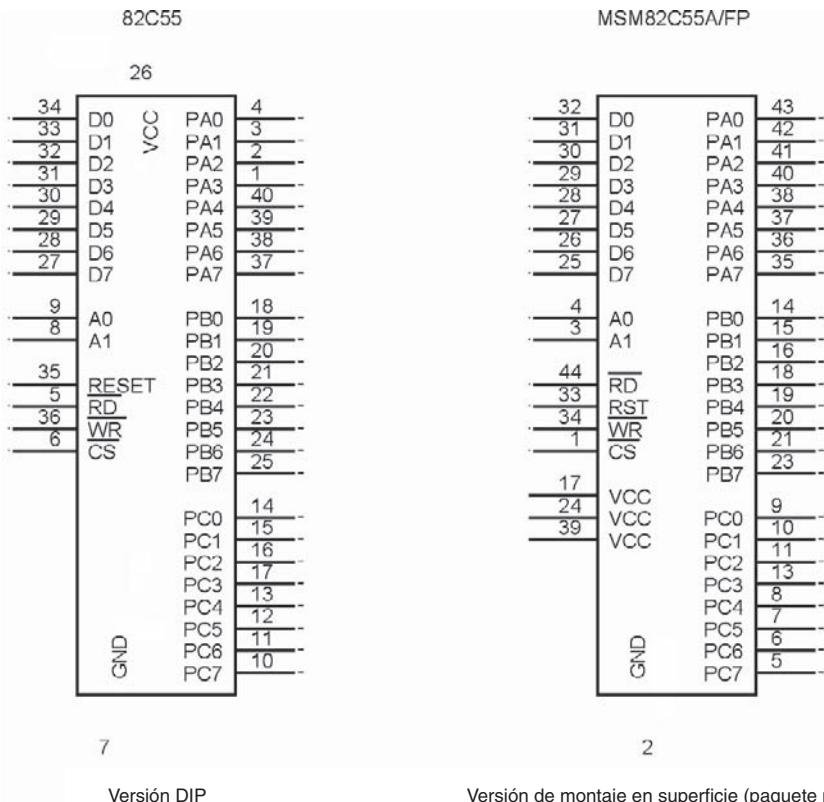
La **interfaz periférica programable** (PPI, por sus siglas en inglés de *programmable peripheral interface*) 82C55 es un componente de interfaz de bajo costo muy popular, que se encuentra en muchas aplicaciones. Esto es cierto incluso con todos los dispositivos programables disponibles para aplicaciones simples. El PPI, que tiene 24 terminales para E/S que pueden programarse en grupos de 12 terminales, tiene grupos que operan en tres modos distintos. El 82C55 puede conectar al microprocesador cualquier dispositivo de E/S compatible con TTL. El 82C55 (versión CMOS) requiere la inserción de estados de espera si se opera con un microprocesador que utilice un reloj mayor de 8 MHz. También proporciona por lo menos 2.5 mA de corriente de colector (0 lógico) en cada salida, con un máximo de 4 mA. Debido a que los dispositivos de E/S son lentos por naturaleza, los estados de espera que se utilizan durante las transferencias de E/S no afectan la velocidad del sistema en forma considerable. El 82C55 sigue teniendo aplicación (es compatible para la programación, aunque no aparezca en el sistema como un 82C55 discreto), incluso hasta en el sistema computacional más reciente basado en Pentium 4. La computadora moderna utiliza unos cuantos componentes 82C55 como interfaz para el teclado y el puerto paralelo de impresora en muchas computadoras personales, pero también se encuentra en forma de una función dentro de un conjunto de chips de interfaz. El conjunto de chips también controla el temporizador y lee datos de la interfaz del teclado.

Descripción básica del 82C55

La figura 11-18 ilustra el diagrama de terminales del 82C55, tanto en el formato DIP como en el formato de montaje en superficie (paquete plano). Sus tres puertos de E/S (etiquetados como A, B y C) se programan como grupos. Las conexiones del grupo A consisten del puerto A (PA_7-PA_0) y de la mitad superior del puerto C (PC_7-PC_4); las conexiones del grupo B consisten del puerto B (PB_7-PB_0) y de la mitad inferior del puerto C (PC_3-PC_0). El 82C55 se selecciona mediante su terminal \overline{CS} para la programación y para leer o escribir en un puerto. La selección del registro se realiza a través de las terminales de entrada A_1 y A_0 , las cuales seleccionan un registro interno para la programación u operación. La tabla 11-2 muestra las asignaciones de puerto de E/S que se utilizan para programar y acceder a los

FIGURA 11-18

El diagrama de terminales del adaptador de interfaz periférica (PPI) 82C55.

**TABLA 11-2** Asignaciones de puerto de E/S para el 82C55.

A_1	A_0	Función
0	0	Puerto A
0	1	Puerto B
1	0	Puerto C
1	1	Registro de comandos

puertos de E/S. En la computadora personal se decodifica un par de componentes 82C55 (o sus equivalentes) en los puertos 60H-63H y también en los puertos 378H-37BH.

El 82C55 es un dispositivo que se conecta al microprocesador y se programa de manera muy sencilla. Para poder leer o escribir, la entrada \overline{CS} debe ser un 0 lógico y debe aplicarse la dirección de E/S correcta a las terminales A_1 y A_0 . El resto de las terminales de dirección de puerto toman el valor “no importa” en cuanto a lo que al 82C55 respecta, y se decodifican de manera externa para seleccionar el 82C55.

La figura 11-19 muestra un 82C55 conectado al 80386SX, de manera que funcione en las direcciones de puerto de E/S de 8 bits C0H (puerto A), C2H (puerto B), C4H (puerto C) y C6H (registro de comandos). Esta interfaz utiliza el banco inferior del mapa de E/S del 80386SX. En esta interfaz todas las terminales del 82C55 son conexiones directas para el 80386SX, excepto la terminal \overline{CS} . Esta terminal se decodifica y se selecciona mediante un decodificador 74ALS138.

La entrada RESET para el 82C55 inicializa el dispositivo cada vez que se reinicia el microprocesador. Una entrada RESET para el 82C55 hace que todos los puertos se establezcan como simples puertos de entrada que operan en modo 0. Como las terminales de los puertos se programan de manera interna como terminales de entrada después de un RESET, se evitan los daños cuando se aplica energía por primera vez al sistema. Después de un RESET no se necesita ningún otro comando para programar el 82C55, siempre y cuando se utilice como dispositivo de entrada para los tres puertos. Hay que tener

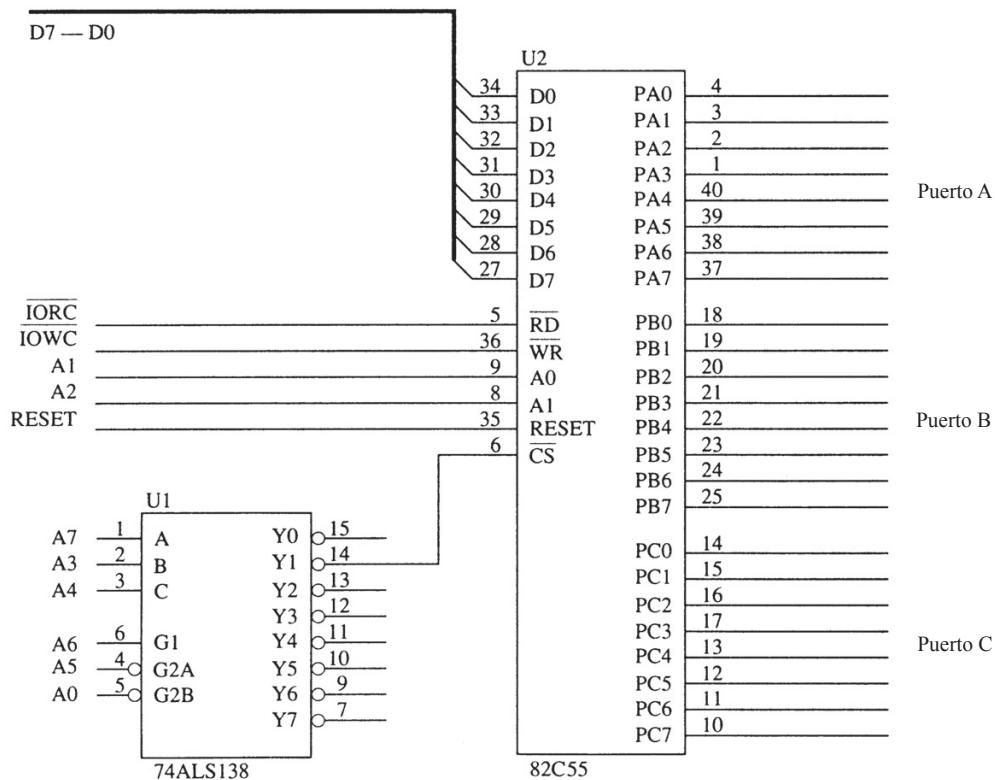


FIGURA 11-19 El 82C55 conectado al banco inferior del microprocesador 80386SX.

en cuenta que un 82C55 se conecta a la computadora personal en las direcciones de puerto 60H-63H para el control del teclado, y también para controlar la bocina, el temporizador y otros dispositivos internos tales como la expansión de memoria. También se utiliza para el puerto paralelo de impresora en los puertos de E/S 378H-37BH.

Programación del 82C55

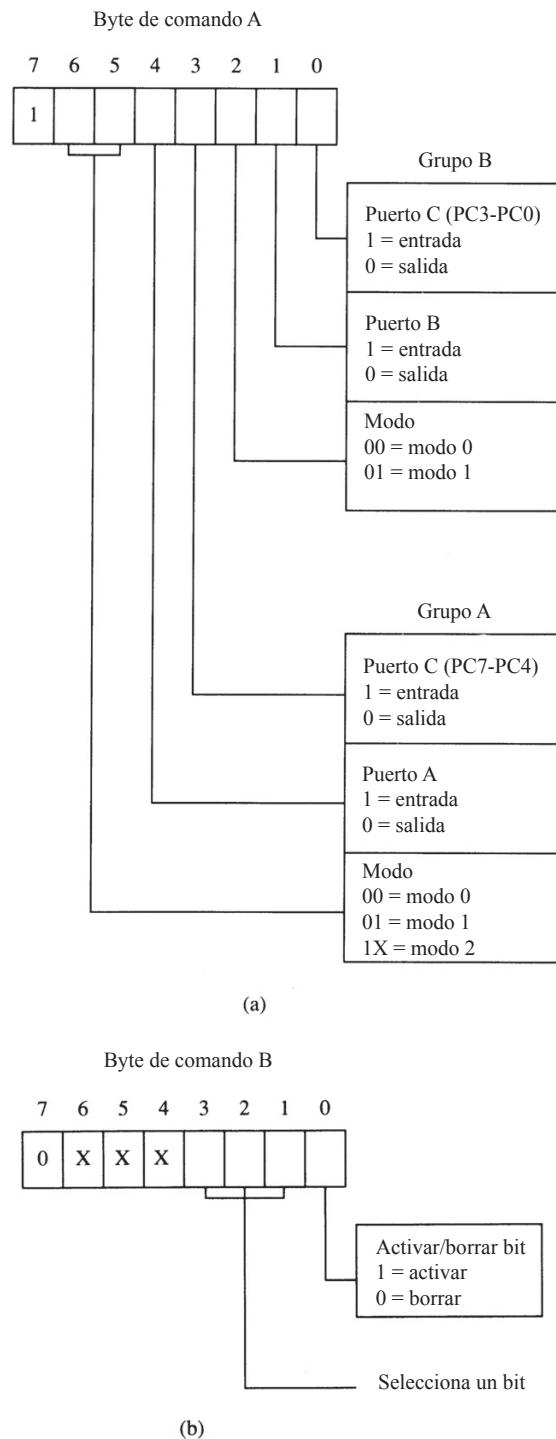
El 82C55 se programa mediante los dos registros de comandos internos que se muestran en la figura 11-20. Observe que la posición del bit 7 se selecciona el byte de comando A o el byte de comando B. El byte de comando A programa la función de los grupos A y B. El byte de comando B activa (1) o borra (0) los bits del puerto C sólo si el 82C55 está programado en modo 1 o 2.

Las terminales del grupo B (el puerto B y la parte inferior del puerto C) se programan ya sea como terminales de entrada o de salida. El grupo B opera en el modo 0 o en el modo 1. El modo 0 es el modo básico de entrada/salida que permite la programación de las terminales del grupo B como operaciones simples de entrada y de salida con enclavamiento. La operación en modo 1 es la operación con estrobo para las conexiones del grupo B en donde los datos se transfieren a través del puerto B y el puerto C proporciona señales para el protocolo de intercambio (handshaking).

Las terminales del grupo A (el puerto A y la parte superior del puerto C) se programan ya sea como terminales de entrada o de salida. La diferencia es que el grupo A puede operar en los modos 0, 1 y 2. La operación en modo 2 es un modo bidireccional de operación para el puerto A.

Si se coloca un 0 en la posición de bit 7 del byte de comando, se selecciona el byte de comando B. Este comando permite activar (1) o borrar (0) cualquier bit del puerto C, si el 82C55 se opera en el modo 1 o 2. De lo contrario, este byte de comando no se utiliza para la programación. La característica de activar/borrar bit se utiliza a menudo en un sistema de control para activar o borrar un bit de control en el puerto C. La función de activar/borrar bit no tiene deformaciones, lo que significa que las otras terminales del puerto C no cambiarán durante el comando de activar/borrar bit.

FIGURA 11-20 El byte de comando del registro de comandos en el 82C55.
 (a) Programa los puertos A, B y C. (b) Activa o borra el bit indicado en el campo para seleccionar un bit.



Operación en modo 0

Este modo de operación hace que el 82C55 funcione como un dispositivo de entrada con búfer o como un dispositivo de salida con enclavamiento. Estos circuitos son los mismos circuitos básicos de entrada y salida que vimos en la primera sección de este capítulo.

La figura 11-21 muestra el 82C55 conectado a un conjunto de ocho pantallas LED de siete segmentos. Éstos son LEDs estándar, pero la interfaz puede modificarse mediante un cambio en los valores

de las resistencias para una pantalla LED orgánica (OLED) o para LEDs de alto brillo. En este circuito los puertos A y B se programan como (modo 0) puertos de salida simples con enclavamiento. El puerto A proporciona las entradas de datos de segmento para la pantalla y el puerto B proporciona un medio para seleccionar una posición de pantalla a la vez, para multiplexar las pantallas. El 82C55 se conecta con un microprocesador 8088 a través de un PLD, para que funcione en los números de puerto de E/S 0700H-0703H. En el ejemplo 11-8 se muestra el programa para el PLD. Éste decodifica la dirección de E/S y desarrolla el estrobo de escritura para la terminal \overline{WR} del 82C55.

EJEMPLO 11-8

```
-- Código de VHDL para el decodificador de la figura 11-21

library ieee;
use ieee.std_logic_1164.all;

entity DECODIFICADOR_11_21 is

port (
    IOM, A15, A14, A13, A12, A11, A10, A9, A8, A7, A6, A5, A4, A3,
        A2: in STD_LOGIC;
    D0: out STD:LOGIC
);
end;

architecture V1 of DECODIFICADOR_11_21 is

begin

    D0 <= not IOM or A15 or A14 or not A13 or A12 or A11 or not A10
        or not A9 or not A8 or A7 or A6 or A5 or A4 or A3 or A2;

end V1;
```

Los valores de las resistencias de la figura 11-21 se seleccionan de manera que la corriente del segmento sea de 80 mA. Esta corriente se requiere para producir una corriente promedio de 10 mA por segmento, a medida que se multiplexen las pantallas. Una pantalla de seis dígitos utiliza una corriente de segmento de 60 mA para un promedio de 10 mA por segmento. En este tipo de sistema de pantallas, sólo una de las ocho posiciones de pantalla está encendida en cualquier momento dado. La corriente pico del ánodo en una pantalla de ocho dígitos es de 560 mA (siete segmentos \times 80 mA), pero la corriente promedio del ánodo es de 80 mA. En una pantalla de seis dígitos, la corriente pico sería de 420 mA (siete segmentos \times 60 mA). Cada vez que se multiplexan las pantallas, incrementamos la corriente de segmento de 10 mA (para una pantalla que utiliza 10 mA por segmento como corriente nominal) a un valor igual al número de posiciones de pantalla multiplicado por 10 mA. Esto significa que una pantalla de cuatro dígitos utiliza 40 mA por segmento, una pantalla de cinco dígitos utiliza 50 mA, y así sucesivamente.

En esta pantalla, la resistencia de carga del segmento pasa 80 mA de corriente y tiene un voltaje aproximado de 3.0 V a través de ella. Se produce una caída de voltaje equivalente al LED (1.65 V nominales) y unos cuantos décimos a través del interruptor del ánodo y del interruptor del segmento, por lo tanto, aparece un voltaje de 3.0 V a través de la resistencia de carga del segmento. El valor de la resistencia es de $3.0\text{ V} \div 180\text{ mA} = 37.5\text{ }\Omega$. En la figura 11-21 se utilizó el valor de resistencia estándar más cercano (39 Ω) para la carga del segmento.

La resistencia en serie con la base del interruptor de segmento, hace que se suponga que la ganancia mínima del transistor sea de 100. Por lo tanto, la corriente base es de $80\text{ mA} \div 100 = 0.8\text{ mA}$. El voltaje aproximado a través de la resistencia de la base es de 3.0 V (el voltaje mínimo para el nivel 1 lógico en el 82C55), menos la caída a través de la unión emisor-base (0.7 V), lo que nos da un total de 2.3 V. De aquí podemos deducir que el valor de la resistencia de la base es de $2.3\text{ V} \div 0.8\text{ mA} = 2.875\text{ K}\Omega$. El valor de resistencia estándar más cercano es de 2.7 K Ω , pero para este circuito se seleccionó una resistencia de 2.2 K Ω .

El interruptor del ánodo tiene una sola resistencia en su base. La corriente a través de esta resistencia es de $560\text{ mA} \div 100 = 5.6\text{ mA}$, ya que la ganancia mínima del transistor es de 100. Este valor

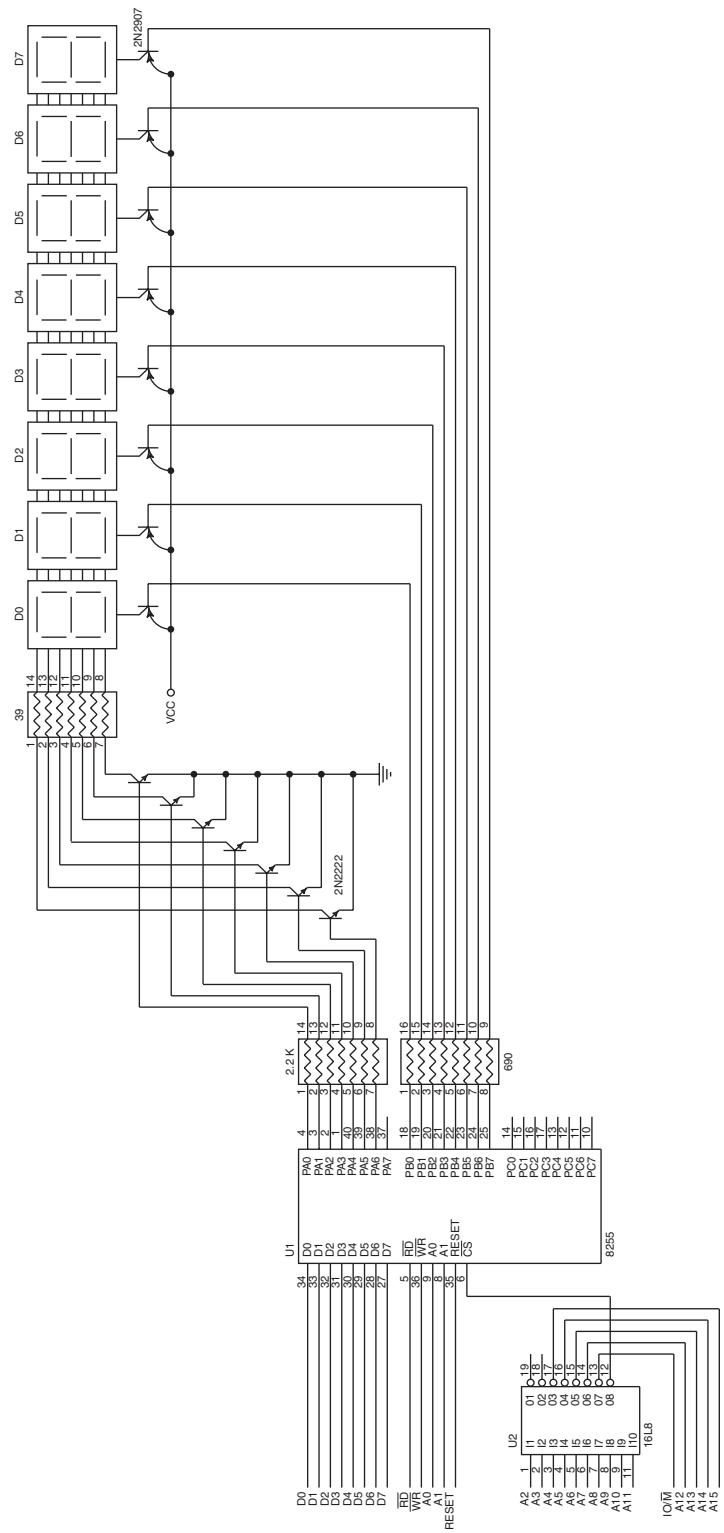


FIGURA 11-21 Una pantalla LED de 8 dígitos conectada al microprocesador 8088, a través de un PIA 82C55.

excede la corriente máxima de 4.0 mA del 82C55, pero está lo bastante cerca como para que funcione sin problemas. La corriente máxima supone que utiliza la terminal del puerto como entrada TIL para otro circuito. Si la cantidad de corriente fuera mayor que 8.0 a 10.0 mA, se requerirían circuitos apropiados (en forma de un par Darlington o de otro interruptor de transistor). Aquí, el voltaje a través de la resistencia de la base es de 5.0 V, menos la caída a través de la unión emisor-base (0.7 V), menos el voltaje en la terminal del puerto (0.4 V), para un nivel de 0 lógico. El valor de la resistencia es de $3.9\text{ V} \div 5.66\text{ mA} = 68.9\text{ }\Omega$. El valor de resistencia estándar más cercano es de $69\text{ }\Omega$, que seleccionamos en este ejemplo.

Antes de examinar el software para operar la pantalla, primero debemos programar el 82C55. Esto se logra mediante la secuencia corta de instrucciones que aparecen en el ejemplo 11-9. Aquí se programan los puertos A y B como salidas.

EJEMPLO 11-9

;programación del PIA 82C55

```
MOV    AL,10000000B      ;comando
MOV    DX,703H           ;direcciona el puerto 703H
OUT   DX,AL              ;envía el comando al puerto 703H
```

El procedimiento para multiplexar las pantallas se muestra en el ejemplo 11-10, tanto en lenguaje ensamblador puro como en lenguaje C++ combinado con lenguaje ensamblador. Para que el sistema de pantallas funcione correctamente, debemos llamar este procedimiento con frecuencia. Observe que el procedimiento llama a otro procedimiento (RETRASO) que produce un retraso de tiempo de 1.0 ms. El retraso de tiempo no se muestra en este ejemplo, pero se utiliza para dar el tiempo necesario a cada posición de pantalla para que se encienda. Los fabricantes de pantallas LED recomiendan que la pantalla parpadee entre 100 y 1500 veces por segundo. Si se utiliza un retraso de 1.0 ms, cada dígito se muestra durante 1.0 ms para una velocidad de parpadeo total de las pantallas de $1000\text{ Hz} \div 8$, o una velocidad de parpadeo de 125 Hz para los ocho dígitos.

EJEMPLO 11-10

;Un procedimiento en lenguaje ensamblador que multiplexa la pantalla de 8 dígitos.

;Este procedimiento debe llamarse con frecuencia para que la pantalla ;aparezca correctamente.

```
PANT    PROC    NEAR    USES AX BX DX SI

        PUSHF
        MOV     BX,8          ;carga el contador
        MOV     AH,7FH         ;carga el patrón de selección
        MOV     SI,OFFSET MEM-1 ;direcciona los datos de la pantalla
        MOV     DX,701H         ;direcciona el puerto B

;muestra los 8 dígitos

.REPEAT
        MOV     AL,AH          ;envía el patrón de selección al puerto B
        OUT    DX,AL
        DEC    DX
        MOV     AL,[BX+SI]       ;envía datos al puerto A
        OUT    DX,AL
        CALL   RETRASO          ;espera 1.0 ms
        ROR    AH,1              ;ajusta el patrón de selección
        INC    DX
        DEC    BX               ;decrementa el contador
.UNTIL BX == 0

        POPF
        RET

PANT    ENDP

//Una función en C++ que multiplexa la pantalla de 8 dígitos
//utiliza un arreglo tipo char llamado MEM
```

```

void Pant()
{
    unsigned int *Mem = &MEM[0];           //apunta al elemento 0 del arreglo
    for ( int a = 0; a < 8; a++ )
    {
        unsigned char b = 0xff ^ ( 1 << a );//forma el patrón de selección
        _asm
        {
            mov al,b
            mov dx,701H
            out dx,al      ;envía el patrón de selección al puerto B
            mov al,Mem[a]
            dec dx
            out dx,al      ;envía los datos al puerto A
        }
        Sleep(1);          ;espera 1.0 ms
    }
}

```

El procedimiento de pantalla (PANT) direcciona un área de memoria llamada MEM, en donde se almacenan los datos (en código de siete segmentos) para los ocho dígitos de la pantalla. El registro AH se carga con un código (7FH) que en un principio direcciona la posición de pantalla más significativa. Una vez que se selecciona esta posición, se direcciona el contenido de la posición de memoria MEM +7 y se envía al dígito más significativo. Luego se ajusta el código de selección para seleccionar el siguiente dígito de la pantalla. Este proceso se repite ocho veces para mostrar el contenido de las posiciones MEM a MEM +7 en los ocho dígitos de la pantalla.

El retraso de tiempo de 1.0 ms puede obtenerse mediante la escritura de un procedimiento que utilice la frecuencia del reloj del sistema para determinar cuánto tiempo requiere cada instrucción para ejecutarse. El procedimiento que se muestra en el ejemplo 11-11 produce un retraso de tiempo cuya duración se determina sobre la base del número de veces que se ejecuta la instrucción LOOP. En este ejemplo se utilizó XXXX, que se llenará con un valor después de que dejemos en claro ciertos puntos. La instrucción LOOP requiere un cierto número de ciclos de reloj para ejecutarse; en el apéndice B podrá descubrir cuántos. Suponga que la interfaz utiliza el microprocesador 80486 que opera con un reloj de 20 MHz. En el apéndice B puede ver que la instrucción LOOP requiere 7/6 ciclos de reloj. El primer número es la cantidad de ciclos de reloj que se requieren cuando ocurre un salto a D1 y el segundo número indica cuándo no se realiza el salto. Con un reloj de 20 MHz, un ciclo de reloj requiere $1 \div 20 \text{ MHz} = 50 \text{ ns}$. En este caso, la instrucción LOOP requiere 350 ns para ejecutarse en todas las iteraciones, excepto la última. Para determinar la cuenta (XXXX) necesaria para lograr un retraso de tiempo de 1.0 ms, dividimos 1.0 ms entre 350 ns. En este caso $XXXX = 2,857$ para lograr un retraso de tiempo de 1.0 ms. Si se produce una cuenta mayor, podrá utilizar una instrucción LOOPD con el registro ECX. Por lo general, podrá ignorar el tiempo requerido para ejecutar las instrucciones MOV CX, XXXX y RET.

Suponga que se utiliza un Pentium 4 con un reloj de 2.0 GHz para el retraso. Aquí, un ciclo de reloj es de 0.5 ns y la instrucción LOOP requiere cinco ciclos por cada iteración. Esto requiere una cuenta de 400,000, por lo que es conveniente utilizar LOOPD con ECX.

EJEMPLO 11-11

```

;ecuación para el retraso
;
;      Tiempo de retraso
;XXXX= -----
;      tiempo para LOOP
;

RETRASO PROC NEAR USES CX

        MOV CX,XXXX
D1:
        LOOP D1
        RET

RETRASO ENDP

```

Conexión de una pantalla LCD al 82C55

Las pantallas LCD (**pantallas de cristal líquido**) han sustituido a las pantallas LED en muchas aplicaciones. La única desventaja de la pantalla LCD es que es difícil de ver en situaciones con poca luz, en las que se sigue utilizando la pantalla LED con ciertas limitaciones. Si el precio de la OLED disminuye lo suficiente, las pantallas LED desaparecerán.

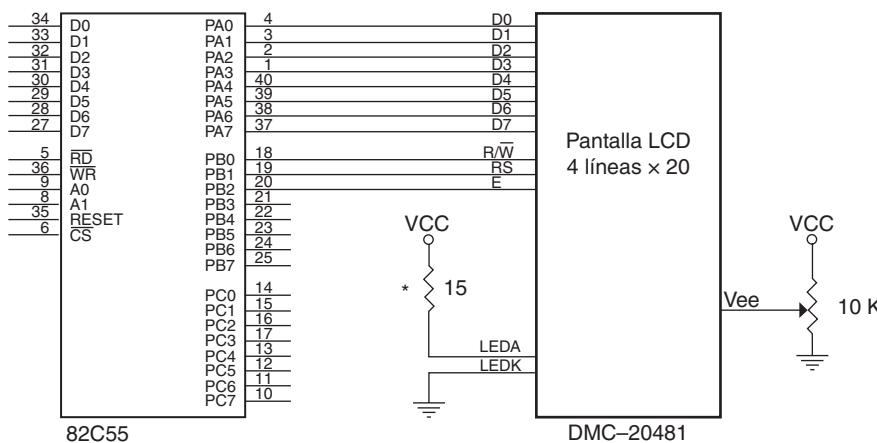
Observe en la figura 11-22 la conexión de la pantalla LCD Optrex DMC-20481 conectada a un 82C55. La DMC-20481 es una pantalla de 4 líneas con 20 caracteres por línea que acepta código ASCII como datos de entrada. También acepta comandos que la inicializan y controlan su aplicación. Como puede ver en la figura 11-22, la pantalla LCD tiene pocas conexiones. Las conexiones de datos, que están conectadas al puerto A del 82C55, se utilizan para introducir datos a la pantalla y para leer información de la misma.

Hay cuatro terminales de control en la pantalla. La conexión V_{EE} se utiliza para ajustar el contraste de la pantalla LCD y, por lo general, se conecta a un potenciómetro de $10\text{ k}\Omega$, como se muestra. La entrada RS (selección de registro) selecciona datos (RS = 1) o instrucciones (RS = 0). La entrada E (habilitación) debe ser un 1 lógico para que la DMC-20481 lea o escriba información y funcione como reloj. Por último, la terminal R/W selecciona una operación de lectura o de escritura. Por lo general, la terminal RS se coloca en 1 o en 0, la terminal R/W se activa o se borra, se colocan datos en las terminales de entrada de datos y después se aplica un pulso a la terminal E para acceder a la DMC-20481. Esta pantalla también tiene dos entradas (LEDA [ánodo] y LEDK [cátodo]) para los diodos LED de la luz de fondo, que no se muestran en la ilustración.

Para programar la DMC-20481 primero debemos inicializarla. Esto se aplica a cualquier pantalla que utilice el circuito integrado del controlador de pantalla HD44780 (Hitachi). Toda la línea completa de paneles de pantalla pequeña de Optrex y de la mayoría de los otros fabricantes se programa de la misma forma. La inicialización se logra mediante los siguientes pasos:

1. Espera por lo menos 15 ms después de que V_{CC} se eleve a 5.0 V.
2. Envía el comando de ajuste de función (30H) y espera por lo menos 4.1 ms.
3. Envía el comando de ajuste de función (30H) por segunda vez y espera por lo menos 100 μ s.
4. Envía el comando de ajuste de función (30H) por tercera vez y espera por lo menos 40 μ s.
5. Envía el comando de ajuste de función (30H) por cuarta vez y espera por lo menos 40 μ s.
6. Envía un 08H para deshabilitar la pantalla y espera por lo menos 40 μ s.
7. Envía un 01H para llevar el cursor a su posición inicial y borrar la pantalla, y espera por lo menos 1.64 ms.
8. Envía la señal habilitación de cursor de pantalla desactivado (0CH) y espera por lo menos 40 μ s.
9. Envía un 06H para seleccionar el autoincremento, desplaza el cursor y espera por lo menos 40 μ s.

FIGURA 11-22 La pantalla LCD DMC-20481 conectada al 82C55.



*Corriente máxima de 480 mA, nominal de 260 mA.

En el ejemplo 11-12 se muestra el software para realizar la inicialización de la pantalla LCD. Es largo, pero el controlador de la pantalla así lo requiere. Observe que no se incluye en el listado el software para los tres retrasos de tiempo. Si va a conectar la pantalla a una PC, puede usar la instrucción RDTSC como se indica en el capítulo sobre el retraso de tiempo en el Pentium. Si va a desarrollar la interfaz para otra aplicación debe escribir retrasos de tiempo separados, los cuales deben proporcionar los retrasos de tiempo que se indican en el diálogo de inicialización.

EJEMPLO 11-12

```

PUERTOA_DIRECCION    EQU 700H           ;establece las direcciones de los puertos
PUERTOB_DIRECCION    EQU 701H
COMANDO_DIRECCION    EQU 703H

;macro para enviar un comando o datos a la pantalla LCD
;
ENVIA    MACRO  PUERTOA_DATOS, PUERTOB_DATOS, RETRASO

        MOV     AL,PUERTOA_DATOS          ;de PUERTOA_DATOS al Puerto A
        MOV     DX,PUERTOA_DIRECCION
        OUT    DX,AL
        MOV     AL,PUERTOB_DATOS          ;de PUERTOB_DATOS al Puerto B
        MOV     DX,PUERTOB_DIRECCION
        OUT    DX,AL
        OR      AL,00000100B            ;establece el bit E
        OUT    DX,AL
        AND    AL,11111011B            ;borra el bit E
        NOP
        NOP..                         ;un pequeño retraso
        OUT    DX,AL
        MOV     BL,RETRASO             ;envía al puerto B
        ;BL = cuenta del retraso
        CALL   RETRASO_MS              ;Retraso de tiempo en ms
        ENDM

;Programa para inicializar la pantalla LCD

START:
        MOV     AL,80H                 ;Programa el 82C55
        MOV     DX,COMANDO_DIRECCION
        OUT    DX,AL
        MOV     AL,0
        MOV     DX,PUERTOB_DIRECCION
        ENVIA  30H, 2, 16              ;borra el Puerto B
        ENVIA  30H, 2, 5
        ENVIA  30H, 2, 1
        ENVIA  38H, 2, 1
        ENVIA  8, 2, 1
        ENVIA  1, 2, 2
        ENVIA  0CH, 2, 1
        ENVIA  6, 2, 1
        ;envía un 30H durante 16 ms
        ;envía un 30H durante 5 ms
        ;envía un 30H durante 1 ms
        ;envía un 38H durante 1 ms
        ;envía un 8 durante 1 ms
        ;envía un 1 durante 2 ms
        ;envía un 0CH durante 1 ms
        ;envía un 6 durante 1 ms

```

Las instrucciones NOP se agregan en la macro ENVIA para asegurar que el bit E permanezca en 1 lógico el tiempo suficiente como para que se active la pantalla LCD. Este proceso debe funcionar en la mayoría de los sistemas con casi todas las frecuencias de reloj, pero tal vez se necesiten instrucciones NOP adicionales para aumentar este tiempo en algunos casos. Observe también que se utilizan instrucciones de igualación para igualar las direcciones de los puertos con las etiquetas correspondientes. Esto se hace de manera que el software pueda modificarse con facilidad si los números de puerto son distintos a los que se utilizan en el programa.

Antes de programar la pantalla debemos explicar los comandos utilizados en el diálogo de inicialización. En la tabla 11-3 podrá ver un listado completo de los comandos o instrucciones para la pantalla LCD. Compare los comandos que se envían a la pantalla LCD en el programa de inicialización con la tabla 11-3.

Una vez que se inicializa la pantalla LCD, se necesitan unos cuantos procedimientos para mostrar la información y controlar la pantalla. Después de la inicialización ya no se necesitan retrasos de tiempo cuando se envían datos o muchos comandos a la pantalla. No obstante, el comando para borrar

TABLA 11-3 Instrucciones para la mayoría de las pantallas LCD.

<i>Instrucción</i>	<i>Código</i>	<i>Descripción</i>	<i>Tiempo</i>
Borra pantalla	0000 0001	Borra la pantalla y regresa el cursor a su posición inicial.	1.64 ms
Devuelve cursor a posición inicial	0000 0010	Devuelve el cursor a su posición inicial.	1.64 ms
Activa modo de entrada	0000 00AS	Establece la dirección de movimiento (A = 1, incremento) y desplazamiento (S = 1, desplazamiento) del cursor.	40 µs
Enciende/apaga pantalla	0000 1DCB	Enciende o apaga la pantalla (D = 1, encendida) (C = 1, cursor activo) (B = 1, parpadeo del cursor).	40 µs
Desplazamiento de cursor/pantalla	0001 SR00	Establece el movimiento del cursor y el desplazamiento de la pantalla (S = 1, desplaza pantalla) (R = 1, derecha).	40 µs
Ajuste de función	001L NF00	Programa el circuito LCD (L = 1, interfaz de 8 bits) (N = 1, 2 líneas) (F = 1, 5 × 10 caracteres) (F = 0, 5 × 7 caracteres).	40 µs
Establece dirección CGRAM	01XX XXXX	Establece la dirección de RAM del generador de caracteres.	40 µs
Establece dirección DRAM	10XX XXXX	Establece la dirección de RAM de la pantalla.	40 µs
Lee bandera BUSY	B000 0000	Lee la bandera de señal de ocupada (B = 1, ocupada).	0
Escribe datos	Datos	Escribe datos en la RAM de la pantalla o en la RAM del generador de caracteres.	40 µs
Lee datos	Datos	Lee datos de la RAM de la pantalla o de la RAM del generador de caracteres.	40 µs

la pantalla aún necesita un retraso de tiempo, ya que la bandera de ocupada (BUSY) no se utiliza con ese comando. En vez de un retraso de tiempo se evalúa la bandera de ocupada para ver si la pantalla ha completado una operación. En el ejemplo 11-13 aparece un procedimiento para evaluar la bandera de ocupada. El procedimiento OCUPADA evalúa la pantalla LCD y sólo regresa cuando ésta ha completado una instrucción anterior.

EJEMPLO 11-13

```

PUERTOA_DIRECCION EQU 700H      ;establece las direcciones de los puertos
PUERTOB_DIRECCION EQU 701H
COMANDO_DIRECCION EQU 703H

OCUPADA    PROC NEAR USES DX AX

        PUSHF
        MOV    DX, COMANDO_DIRECCION
        MOV    AL, 90H          ;programa el puerto A como IN
        OUT   DX, AL
        .REPEAT
        MOV    AL, 5           ;selecciona operación de lectura de la LCD
        MOV    DX, PUERTOB_DIRECCION
        OUT   DX, AL          ;y el pulso E
        NOP
        NOP

```

```

        MOV    AL, 1
        OUT   DX, AL
        MOV   DX, PUERTOA_DIRECCION
        MOV   AL, DX ; comando de ocupada en lectura
        SHL   AL, 1
.UNTIL !CARRY? ; hasta que no esté ocupada
        MOV   DX, COMANDO_DIRECCION
        MOV   AL, 80H
        OUT  DX, AL ; programa el puerto A como OUT
        POPF
        RET

OCUPADA    ENDP

```

Una vez disponible el procedimiento OCUPADA, pueden enviarse datos a la pantalla mediante la escritura de otro procedimiento llamado ESCRIBE. Este procedimiento utiliza el procedimiento OCUPADA para probar antes de tratar de escribir nuevos datos en la pantalla. El ejemplo 11-14 muestra el procedimiento ESCRIBE, el cual transfiere el carácter ASCII del registro BL a la posición actual del cursor de la pantalla. Observe que el diálogo de inicialización ha enviado el cursor para que se autoincremente, por lo que si ESCRIBE se llama más de una vez, los caracteres que se escriban en la pantalla aparecerán uno al lado del otro, como lo harían en una pantalla de vídeo.

EJEMPLO 11-14

```

WRITE    PROC  NEAR
        MOV   AL, BL ;BL al Puerto A
        MOV   DX, PUERTOA_DIRECCION
        OUT  DX, AL
        MOV   AL, 0 ;escribe el valor ASCII
        MOV   DX, PUERTOB_DIRECCION
        OUT  DX, AL
        OR    AL, 00000100B ;activa el bit E
        OUT  DX, AL ;envía al puerto B
        AND  AL, 11111011B ;borra el bit E
        NOP
        NOP ;un pequeño retraso
        OUT  DX, AL ;envía al puerto B
        CALL OCUPADA ;espera a que se complete
        RET
ESCRIBE  ENDP

```

El último procedimiento que se necesita para una pantalla básica es el de borrar la pantalla y devolver el cursor a su posición inicial, el cual se llama CLS, como se muestra en el ejemplo 11-15. Este procedimiento utiliza la macro ENVIA del software de inicialización para enviar el comando para borrar la pantalla. Con CLS y los procedimientos que hemos presentado hasta ahora, puede mostrar cualquier mensaje en la pantalla, borrarlo y mostrar otro mensaje; esto le permite un control básico de la pantalla. Como se mencionó antes, el comando para borrar la pantalla requiere un retraso de tiempo (por lo menos 1.64 ms) en vez de llamar a OCUPADA para la operación apropiada.

EJEMPLO 11-15

```

CLS     PROC  NEAR
        ENVIA  1, 2, 2
        RET
CLS     ENDP

```

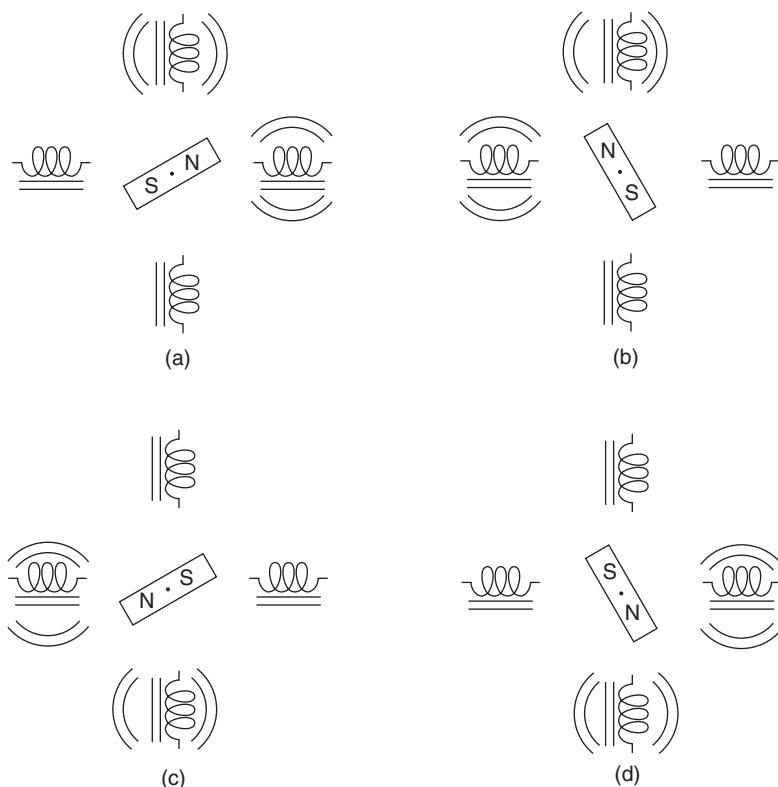
Podrían desarrollarse procedimientos adicionales para seleccionar una posición de la RAM de la pantalla. La dirección de RAM de la pantalla empieza en 0 y progresa a través de la pantalla hasta la última dirección de carácter de la primera línea en la posición 19; la posición 20 es la primera posición de la segunda línea de la pantalla, y así sucesivamente. Una vez que pueda mover la dirección de la pantalla, podrá modificar caracteres individuales e incluso leer datos. Si lo desea, puede desarrollar estos procedimientos por su cuenta.

Ahora vamos a hablar sobre la RAM que está dentro de la pantalla LCD. La LCD contiene 128 bytes de memoria, los cuales se direccionan desde 00H hasta 7FH. No siempre se utiliza toda esta memoria. Por ejemplo, la pantalla de una línea × 20 caracteres utiliza sólo los primeros 20 bytes de la memoria (00H-13H). La primera línea de estas pantallas siempre empieza en la dirección 00H. La segunda línea de cualquier pantalla que opere mediante el HD44780 siempre empieza en la dirección 40H. Por ejemplo, una pantalla de dos líneas × 40 caracteres utiliza las direcciones 00H-27H para almacenar datos en código ASCII de la primera línea. La segunda línea se almacena en las direcciones 40H-67H para esta pantalla. En las pantallas de cuatro líneas, la primera línea se encuentra en 00H, la segunda en 40H, la tercera en 14H y la última en 54H. El dispositivo de pantalla más grande que utiliza el HD44780 es una pantalla de dos líneas × 40 caracteres. La pantalla de cuatro líneas por 40 caracteres utiliza un M50530 o un par de controladores HD44780. Ya que puede encontrarse información sobre estos dispositivos mediante Internet, no los cubriremos en este texto.

Interfaz entre un motor de velocidad gradual y el 82C55. Otro dispositivo que se conecta a menudo con un sistema computacional es el *motor de velocidad gradual*. Éste es un motor digital, ya que se mueve en pasos discretos a medida que recorre 360°. Un motor de velocidad gradual común está diseñado para moverse tal vez 15° por paso para un motor económico, o 1° por paso para un motor de alta precisión y más costoso. En todos los casos, estos pasos se obtienen mediante muchos polos magnéticos y/o engranajes. En la figura 11-23 puede observar que hay dos bobinas energizadas. Si se requiere menos potencia puede energizarse una bobina a la vez, lo que ocasiona que el motor avance gradualmente a 45°, 135°, 225° y 315°.

En la figura 11-23 puede ver un motor de velocidad gradual con cuatro bobinas que utilizan una armadura con un solo polo. Observe que el motor se muestra cuatro veces, y en cada una de ellas la armadura (magnética permanente) gira cuatro posiciones distintas. Esto se logra mediante la energización de las bobinas, como se observa. Ésta es una ilustración de un avance gradual completo. El motor de velocidad gradual se controla mediante pares de amplificadores Darlington NPN para proporcionar una corriente grande a cada bobina.

FIGURA 11-23 El motor de velocidad gradual que muestra su operación en todos los pasos. (a) 45° (b) 135° (c) 225° (d) 315°.



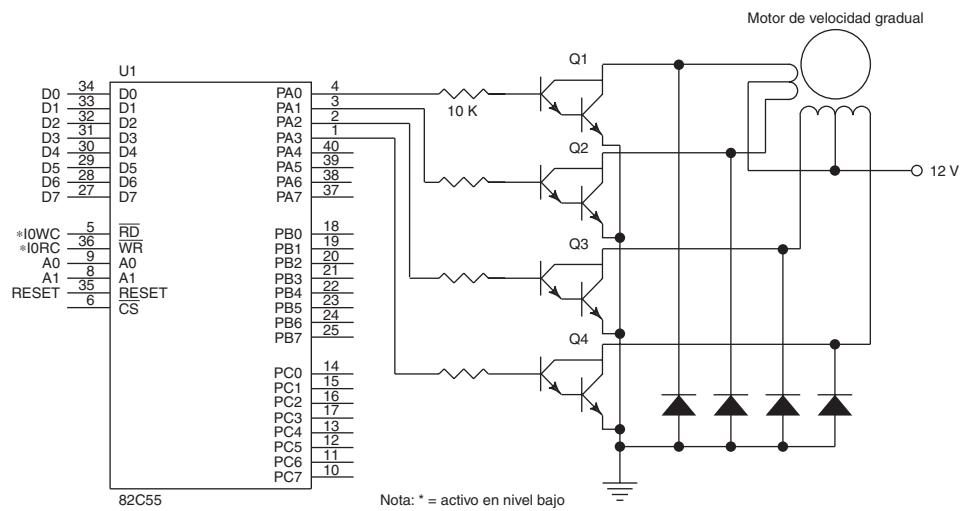


FIGURA 11-24 Un motor de velocidad gradual conectado al 82C55. En esta ilustración no se muestra el decodificador.

En la figura 11-24 se muestra un circuito que puede controlar este motor de velocidad gradual; también se muestran las cuatro bobinas. Este circuito utiliza el 82C55 para proporcionar las señales de control que se utilizan para girar la armadura del motor, ya sea en sentido de las manecillas del reloj o en sentido contrario.

En el ejemplo 11-16 se puede ver un procedimiento sencillo que controla el motor (suponiendo que el puerto A se programa en el modo 0 como dispositivo de salida), en lenguaje ensamblador y como una función en C++. En la llamada a esta subrutina, CX guarda el número de pasos y la dirección de giro. Si CX es mayor que 8000H, el motor gira en sentido de las manecillas del reloj; si CX es menor que 8000H, gira en sentido contrario a las manecillas del reloj. Por ejemplo, si el número de pasos es 0003H, avanza tres pasos en sentido de las manecillas del reloj. El bit de más a la izquierda de CX se elimina y los 15 bits restantes contienen el número de pasos. Observe que este procedimiento utiliza un retraso de tiempo (el cual no se muestra) que produce un retraso de tiempo de 1 ms. Este retraso de tiempo se requiere para dar tiempo a la armadura del motor de velocidad gradual para que avance a su siguiente posición.

EJEMPLO 11-16

```

PUERTO    EQU    40H
;  
Un procedimiento en lenguaje ensamblador que controla el motor de velocidad  
gradual

PASO      PROC    NEAR USES CX AX
        MOV AL, POS          ;obtiene la posición
        OR CX,CX             ;establece los bits de bandera
        IF !ZERO?
            .IF !SIGN?       ;si no hay signo
                .REPEAT
                    ROL AL,1      ;gira un paso a la izquierda
                    OUT PUERTO,AL
                    CALL RETRASO   ;espera 1 ms
                .UNTILCXZ
            .ELSE
                AND CX,7FFFH     ;convierte a CX en un número positivo
                .REPEAT
                    ROR AL,1      ;gira un paso a la derecha
                    OUT PUERTO,AL

```

```

        CALL RETRASO      ;espera 1 ms
        .UNTILCXZ
        .ENDIF
        .ENDIF
        MOV POS,AL
        RET

PASO      ENDP

//Una función en C++ que controla el motor de velocidad gradual

char Paso(char Pos, short Paso)
{
    char Direccion = 0;
    if ( Paso < 0 )
    {
        Direccion = 1;
        Paso =& 0x8000;
    }
    while ( Paso )
    {
        if ( Direccion )
            if ( ( Pos & 1 ) == 1 )
                Pos = ( Pos >> 1 ) | 0x80;
            else
                Pos >= 1;
        else
            if ( ( Pos & 0x80 ) == 0x80 )
                Pos = ( Pos << 1 ) | 1;
            else
                Pos <= 1;
        _asm
        {
            mov al,Pos
            out 40h,al
        }
    }
    return Pos;
}

```

La posición actual se almacena en la posición de memoria POS, la cual debe inicializarse con 33H, 66H, 0EEH o 99H. Esto permite que una simple instrucción ROR (paso a la derecha) o ROL (paso a la izquierda) gire el patrón de bits binarios para el siguiente paso.

La versión en C++ tiene dos parámetros: Pos es la posición actual del motor de velocidad gradual y Paso es el número de pasos, como se describió antes. El nuevo valor de Pos se devuelve en la versión de C++, en vez de almacenarse en una variable.

Los motores de velocidad gradual también pueden operarse en el modo de medio paso, el cual permite ocho pasos por secuencia. Esto se logra mediante el uso de la secuencia de paso completo descrita, en donde un medio paso se obtiene al energizar una bobina entremezclada en los pasos completos. El uso de medios pasos permite posicionar la armadura en 0°, 90°, 180° y 270°. Los códigos de posición de medio paso son 11H, 22H, 44H y 88H. Una secuencia completa de ocho pasos sería: 11H, 33H, 22H, 66H, 44H, 0CCH, 88H y 99H. Esta secuencia podría ser la salida de una tabla de búsqueda o podría generarse mediante software.

Interfaz de matriz de teclas. Los teclados existen en una gran variedad de tamaños, desde los teclados QWERTY estándar de 101 teclas que se conectan al microprocesador hasta los pequeños teclados especializados que pueden contener de cuatro a 16 teclas. En esta sección nos concentraremos en los teclados pequeños que pueden comprarse previamente con ensamblador o que pueden construirse a partir de interruptores de teclas individuales.

En la figura 11-25 puede observar una pequeña matriz de teclas que contiene 16 interruptores conectados a los puertos A y B de un 82C55. En este ejemplo, los interruptores se forman en una matriz de 4 × 4, pero podría utilizarse cualquier matriz (por ejemplo, de 2 × 8). Observe cómo las teclas se organizan en cuatro filas (FILA₀-FILA₃) y cuatro columnas (COL₀-COL₃). Cada fila se conecta a 5.0 V

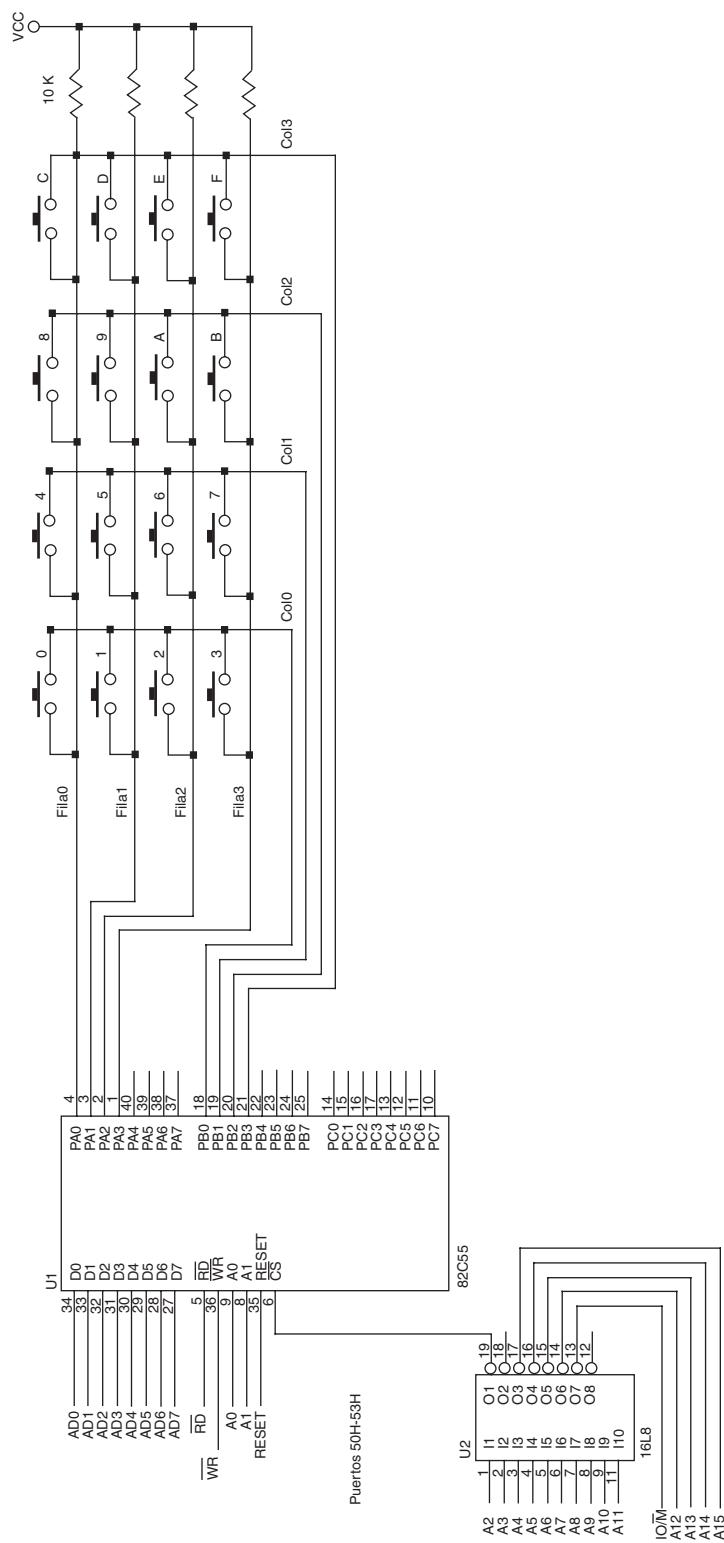


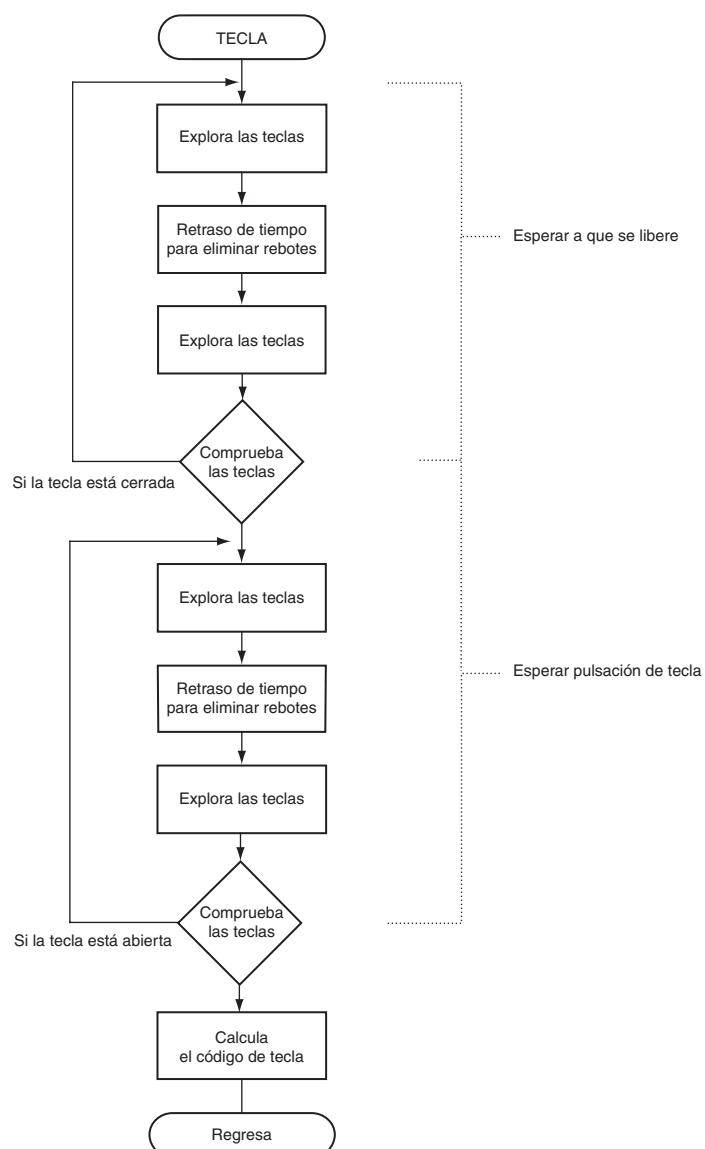
FIGURA 11-25 Una matriz de teclado de 4×4 conectada a un microprocesador 8088 a través del PIA 82C55.

a través de una resistencia elevadora de $10\text{ K}\Omega$ para asegurar que se vaya al nivel alto cuando no esté cerrado un interruptor de botón.

En un microprocesador 8088, el 82C55 se decodifica (el programa del PLD no se muestra) en los puertos de E/S 50H-53H. El puerto A se programa como puerto de entrada para leer las filas y el puerto B se programa como puerto de salida para seleccionar una columna. Por ejemplo, si se envía el número 1110 a las terminales PB₃-PB₀ del puerto B, la columna 0 tiene un 1 lógico, por lo que se seleccionan las cuatro teclas de la columna 0. Cuando hay un 0 lógico en PB, los únicos interruptores que pueden colocar un 0 lógico en el puerto A son los interruptores 0-3. Si los interruptores 4-F están cerrados, las terminales correspondientes del puerto A permanecen en 1 lógico. De igual forma, si se envía el número 1101 al puerto B se seleccionan los interruptores 4-7, y así sucesivamente.

La figura 11-26 muestra un diagrama de flujo del software requerido para leer una tecla de la matriz de teclado y eliminar el rebote. Para eliminar los rebotes de las teclas se utiliza un retraso de tiempo corto de 10 a 20 ms. El diagrama de flujo contiene tres secciones principales. La primera sección espera a que se libere una tecla. Esto parece confuso, pero el software se ejecuta con mucha velocidad en el microprocesador y existe la posibilidad de que el programa regrese al principio antes de que se libere

FIGURA 11-26 Diagrama de flujo de un procedimiento de exploración de teclado.



la tecla, por lo que primero debemos esperar a que suceda esto. A continuación, el diagrama de flujo muestra que debemos esperar una pulsación de tecla. Una vez que se detecte la pulsación, se calcula la posición de la tecla en la parte final del diagrama de flujo.

El software utiliza un procedimiento llamado EXPLORA para explorar las teclas y otro llamado RETRASO10 (el cual no se muestra en este ejemplo) que desperdicia 10 ms de tiempo para eliminar rebotes. El procedimiento principal se llama TECLA y aparece con los demás en el ejemplo 11-17. En este ejemplo también se lista una función en C++ para realizar una operación de lectura de tecla. Observe que el procedimiento TECLA es genérico, por lo que puede manejar cualquier configuración de teclado, desde una matriz de 1×1 hasta una de 8×8 . Si se cambian las dos igualaciones al inicio del programa (FILAS y COLS), se cambia la configuración de software para un teclado de cualquier tamaño. Observe también que no se muestran los pasos requeridos para inicializar el 82C55 de manera que el puerto A sea un puerto de entrada y que el puerto B sea un puerto de salida.

Con ciertos teclados que no siguen el procedimiento mediante el cual se exploran las teclas, puede ser necesaria una tabla de búsqueda para convertir los códigos de tecla puros que devuelve el procedimiento TECLA en códigos de tecla que concuerden con las teclas en el teclado. El software de búsqueda se coloca justo antes de regresar de TECLA. Consiste tan sólo en una instrucción BX, OFFSET TABLA seguida de la instrucción XLAT.

EJEMPLO 11-17(A)

```
;versión en lenguaje ensamblador;

;TECLA explora el teclado y devuelve el código de tecla en AL.

COLS      EQU    4
FILAS     EQU    4
PUERTOA   EQU    50H
PUERTOB   EQU    51H

TECLA    PROC    NEAR USES CX BX
          MOV     BL,FFH           ;calcula la máscara de fila
          SHL     BL,FILAS
          MOV     AL,0
          OUT    PUERTOB,AL        ;coloca ceros en el puerto B
          .REPEAT
              .REPEAT
                  CALL    EXPLORA
                  .UNTIL ZERO?
                  CALL    RETRASO10
                  CALL    EXPLORA
                  .UNTIL ZERO?
              .REPEAT
                  .REPEAT
                      CALL    EXPLORA
                      .UNTIL !ZERO?
                      CALL    RETRASO10
                      CALL    EXPLORA
                      .UNTIL !ZERO?
          MOV CX,00FEH
          .WHILE 1
              MOV     AL,CL
              OUT    PUERTOB,AL
              CALL    RETRASOCORTO
              CALL    EXPLORA
              .BREAK !ZERO?
              ADD    CH,COLS
              ROL    CL,1
          .ENDW
          .WHILE 1
              SHR    AL,1
              .BREAK .IF !CARRY?
              INC    CH
          ;busca la fila
          ;vea el texto
          ;busca la columna
```

```

        .ENDW
        MOV     AL,CH           ;obtiene el código de la tecla
        RET

TECLA    ENDP

EXPLORA  PROC    NEAR
        IN      AL,PUERTOA       ;lee las filas
        OR      AL,BL
        CMP     AL,0FFH          ;evalúa si no hay teclas
        RET

EXPLORA  ENDP

```

EJEMPLO 11-17(B)

```

//Versión del software para exploración de teclado en C++

#define FILAS 4
#define COLS 4
#define PUERTOA 50h
#define PUERTOB 51h

char Tecla()
{
    char mascara = 0xff << FILAS;
    _asm {
        mov al,0                 ;selecciona todas las columnas
        out PUERTOB,al
    }
    do {                                //espera a que se libere la tecla
        while ( Explora(mascara) );
        Retraso();
    } while ( Explora( mascara ) );
    do {                                //espera a que se oprima una tecla
        while ( !Explora( mascara ) );
        Retraso();
    } while ( !Explora( mascara ) );
    unsigned char seleccion = 0xfe;
    char tecla = 0;
    _asm {
        mov al,seleccion
        out PUERTOB,al
    }
    RetrasoCorto();
    while ( !Explora ( mascara ) ){      //calcula el código de la tecla
        _asm
        {
            mov al,seleccion
            rol al,1
            mov seleccion,al
            out PUERTOB,al
        }
        RetrasoCorto();
        tecla += COLS;
    }
    _asm {
        in al,PUERTOA
        mov seleccion,al
    }
    while ( ( seleccion & 1 ) != 0 ) {
        seleccion << 1;
        tecla++;
    }
    return tecla;
}
bool Explora(mascara)
{

```

```

    bool bandera;
    _asm
    {
        in al,PUERTOA
        mov bandera,al
    }
    return ( bandera | mascara );
}

```

El procedimiento RETRASOCORTO es necesario ya que la computadora cambia el puerto B a una velocidad muy alta. El retraso de tiempo corto concede tiempo para que los datos que se envían al puerto B se estabilicen en su estado final. En la mayoría de los casos esto no es necesario si la velocidad de exploración (el tiempo transcurrido entre cada instrucción de salida) de esta parte del software no excede los 30 KHz. Si lo hace, la Comisión federal de comunicaciones (FCC) no aprobará su aplicación en ninguno de los sistemas aceptados. Sin la certificación de la FCC tipo A o B, el sistema no puede venderse.

Entrada estroboscópica en modo 1

La operación en modo 1 hace que el puerto A y/o el puerto B funcionen como dispositivos de entrada con enclavamiento. Esto permite almacenar los datos externos en el puerto, hasta que el microprocesador esté listo para recuperarlos. El puerto C también se utiliza en la operación en modo 1; no para datos, sino para señales de control o del protocolo de intercambio (handshaking) que ayudan a operar al puerto A o al puerto B (o ambos) como puertos de entrada estroboscópica. En la figura 11-27 puede ver cómo se estructuran ambos puertos para la operación de entrada estroboscópica en modo 1 y el diagrama de sincronización.

El puerto de entrada estroboscópica captura datos de las terminales del puerto cuando se activa el estrobo (STB). Observe que el estrobo captura los datos del puerto en la transición de 0 a 1. La señal STB hace que se capturen los datos en el puerto; además activa las señales IBF (**búfer de entrada lleno**) e INTR (**petición de interrupción**). Una vez que el microprocesador detecta, ya sea mediante software (IBF) o hardware (INTR), que los datos se encuentran en el puerto, ejecuta una instrucción IN para leer el puerto RD. La acción de leer el puerto restaura las señales IBF e INTR a sus estados inactivos, hasta que el siguiente dato se introduzca en el puerto mediante un estrobo.

Definiciones de señales para la entrada estroboscópica en modo 1

STB	La entrada estrobo carga los datos en el enclavamiento del puerto, el cual guarda la información hasta que se envía al microprocesador mediante la instrucción IN.
IBF	Búfer de entrada lleno es una salida que indica que el enclavamiento de entrada contiene información.
INTR	Petición de interrupción es una salida que <u>solicita</u> una interrupción. La terminal INTR se vuelve 1 lógico cuando la entrada STB se regresa a un 1 lógico, y se borra cuando el microprocesador introduce los datos desde el puerto.
INTE	La señal habilitación de interrupción no es una entrada ni una salida; es un bit interno que se programa a través de la posición de bit PC ₄ (puerto A) o PC ₂ (puerto B) del puerto.
PC₇, PC₆	Las terminales 7 y 6 del puerto C son terminales de E/S de propósito general que están disponibles para cualquier fin.

Ejemplo de entrada estroboscópica. El teclado es un excelente ejemplo de un dispositivo de entrada estroboscópica. El codificador del teclado elimina los rebotes de los interruptores de las teclas y proporciona una señal estroboscópica cada vez que se oprime una tecla; además la salida de datos contiene el código de la tecla en código ASCII. La figura 11-28 muestra un teclado conectado al puerto A de entrada estroboscópica. Aquí se activa DAV (datos disponibles) durante 1.0 μ s, cada vez que se escribe una tecla mediante el teclado. Esto hace que los datos se envíen mediante estrobo al puerto A, ya que DAV se conecta a la entrada STB del puerto A. Por lo tanto, cada vez que se escribe una tecla se almacena en

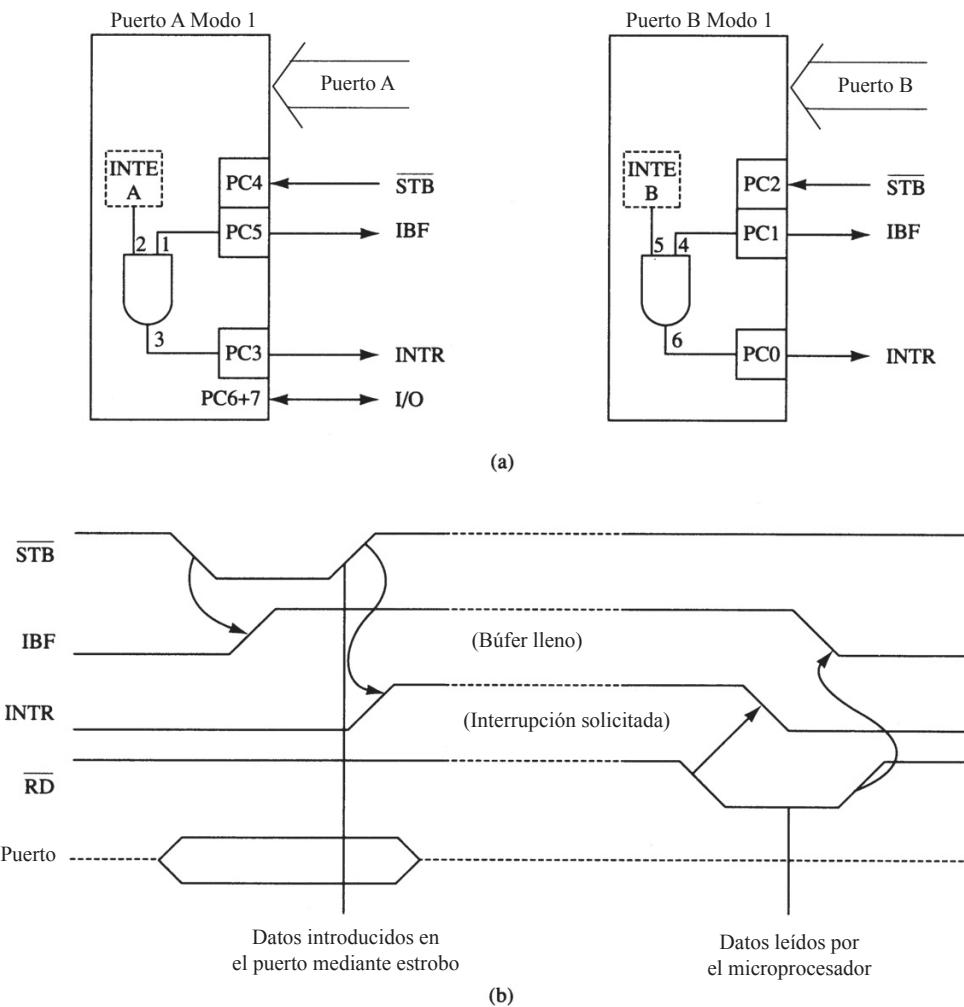
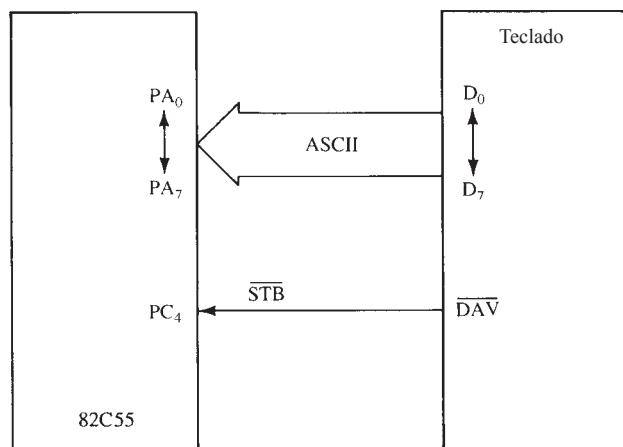


FIGURA 11-27 Operación de entrada estroboscópica (modo 1) del 82C55, (a) estructura interna y (b) diagrama de sincronización.

FIGURA 11-28 Uso del 82C55 para la operación de entrada estroboscópica de un teclado.



el puerto A del 82C55. La entrada STB también activa la señal IBF, la cual indica que los datos están en el puerto A.

El ejemplo 11-18 muestra un procedimiento que lee datos del teclado cada vez que se escribe una tecla. Este procedimiento lee la tecla del puerto A y regresa el código ASCII en AL. Para detectar una tecla se lee el puerto C y se evalúa el bit IBF (posición de bit PC₅) para ver si el búfer está lleno. Si está vacío (IBF = 0) entonces el procedimiento sigue evaluando este bit y espera a que se escriba un carácter en el teclado.

EJEMPLO 11-18

;Un procedimiento que lee el codificador del teclado y
;devuelve el código de tecla ASCII en AL

```

BIT5      EQU      20H
PUERTOC   EQU      22H
PUERTOA   EQU      20H

LEE       PROC      NEAR
          .REPEAT
          IN   AL,PUERTOC      ;sondea el bit IBF
          TEST AL,BIT5
          .UNTIL !ZERO?
          IN   AL,PUERTOA      ;obtiene los datos ASCII
          RET

LEE       ENDP

```

Salida estroboscópica en modo 1

En la figura 11-29 puede ver la configuración interna y el diagrama de sincronización del 82C55 cuando se opera como un dispositivo de salida estroboscópica en el modo 1. Esta operación de salida estroboscópica es similar a la operación de salida en modo 0, sólo que se incluyen las señales de control para proporcionar el protocolo de intercambio (handshaking).

Cada vez que se escriben datos en un puerto programado como puerto de salida estroboscópica, la señal OBF (**búfer de salida lleno**) se vuelve un 0 lógico para indicar que hay datos en el enclavamiento del puerto. Esta señal indica que hay datos disponibles para un dispositivo de E/S externo que quite los datos mediante un estrobo en la entrada ACK (reconocimiento) del puerto. La señal ACK regresa la señal OBF a un 1 lógico, con lo cual indica que el búfer no está lleno.

Definiciones de señales para la salida estroboscópica en modo 1

OBF	Búfer de salida lleno es una salida que se va al nivel bajo siempre que se envían datos (OUT) al enclavamiento del puerto A o del puerto B. Esta señal se vuelve un 1 lógico cada vez que el pulso ACK regresa del dispositivo externo.
ACK	La señal de reconocimiento hace que la terminal OBF regrese al nivel de 1 lógico. La señal ACK es una respuesta de un dispositivo externo para indicar que ha recibido los datos del puerto del 82C55.
INTR	Petición de interrupción es una señal que interrumpe con frecuencia al microprocesador cuando el dispositivo externo recibe los datos mediante la señal ACK. Esta terminal se califica basándose en el bit INTE (habilitación de interrupción) interno.
INTE	Habilitación de interrupción no es una entrada ni una salida; es un bit interno que se programa para habilitar o deshabilitar la terminal INTR. El bit INTE A se programa mediante el uso del bit PC ₆ y el bit INTE B se programa mediante el uso del bit PC ₂ .
PC₄, PC₅	Las terminales PC ₄ y PC ₅ del puerto C son terminales de E/S de propósito general. Para activar o borrar estas dos terminales se utiliza el comando para activar y borrar bits.

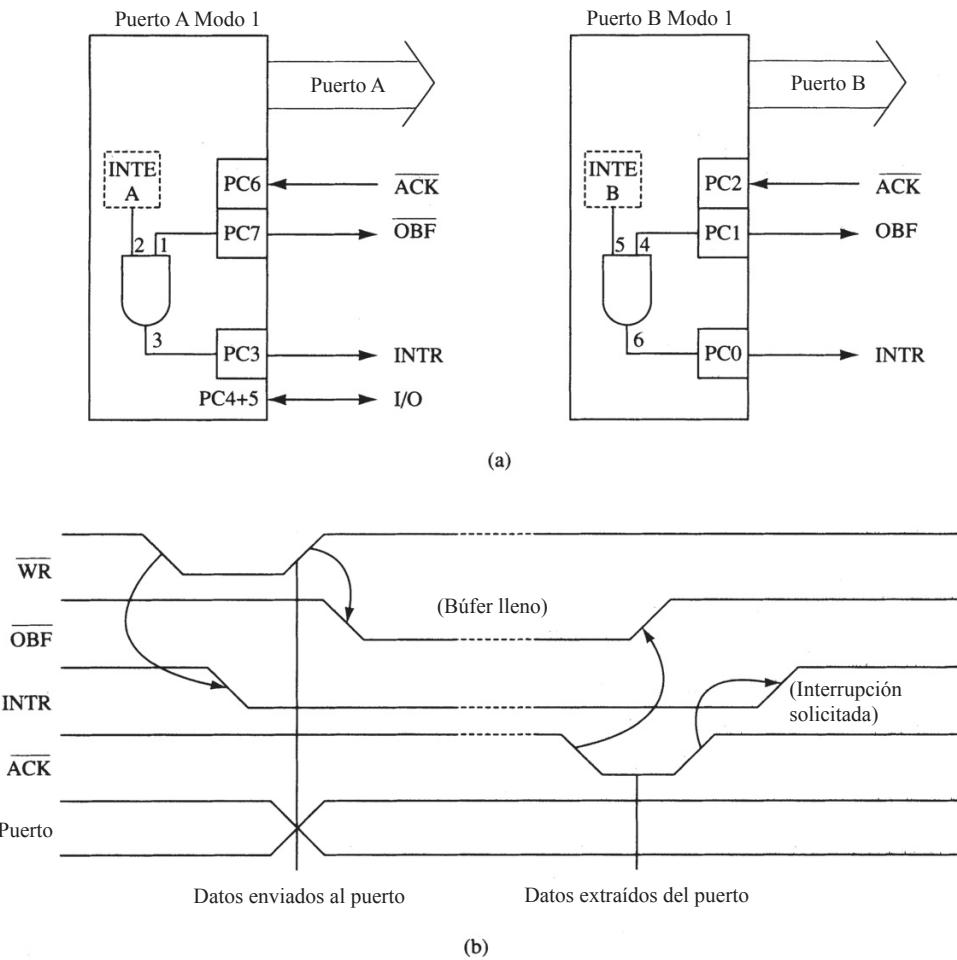


FIGURA 11-29 Operación de salida estroboscópica (modo 1) del 82C55. (a) Estructura interna y (b) diagrama de sincronización.

Ejemplo de salida estroboscópica. Aquí utilizaremos la interfaz de impresora que vimos en la sección 11-1 para demostrar cómo se obtiene la sincronización de la salida estroboscópica entre la impresora y el 82C55. Observe en la figura 11-30 el puerto B conectado a una impresora paralela, con ocho entradas de datos para recibir datos en código ASCII, una entrada \overline{DS} (estrobo de datos) para enviar datos a la impresora mediante un estrobo y una salida \overline{ACK} para aceptar la recepción del carácter ASCII.

En este circuito no hay señal para generar la señal \overline{DS} que va a la impresora, por lo que se utiliza PC_4 con software para generar la señal \overline{DS} . La señal \overline{ACK} que se devuelve de la impresora reconoce la recepción de los datos y se conecta a la entrada \overline{ACK} del 82C55.

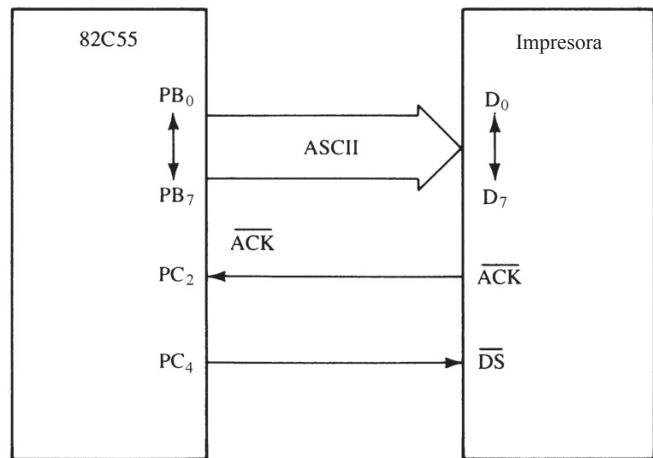
El ejemplo 11-19 muestra el software que envía a la impresora el carácter en código ASCII que está en AH. El procedimiento primero evalúa \overline{OBF} para decidir si la impresora ha extraído o no los datos del puerto B. Si no los ha extraído, el procedimiento espera a que la señal \overline{ACK} regrese de la impresora. Si $\overline{OBF} = 1$, el procedimiento envía el contenido de AH a la impresora a través del puerto B y también envía la señal \overline{DS} .

EJEMPLO 11-19

```
;Un procedimiento que transfiere un carácter ASCII de AH a la impresora
;conectada al puerto B
```

```
BIT1      EQU 2
PUERTOC  EQU 63H
```

FIGURA 11-30 El 82C55 conectado a una interfaz de impresora paralela que muestra el modo de operación de salida estroboscópica para el 82C55.



```

PUERTOB EQU 61H
CMD EQU 63H

IMPRIME PROC NEAR

    .REPEAT           ;espera a que la impresora esté lista
        INT AL,PUERTOC
        TEST AL,BIT1
    .UNTIL !ZERO?
        MOV AL,AH          ;envía ASCII
        OUT PUERTOB,AL
        MOV AL,8            ;pulso estroboscópico de datos
        OUT CMD,AL
        MOV AL,9
        OUT CMD,AL
    RET

RETURN ENDP

```

Operación bidireccional en modo 2

En el modo 2 (que se permite sólo con el grupo A), el puerto A se vuelve bidireccional, con lo cual se permite la transmisión y recepción de datos a través de los mismos ocho cables. El uso de un bus bidireccional para enviar datos es de utilidad cuando se conectan dos computadoras. También se utiliza para el estándar IEEE-488 de interfaz GPIB paralelo de alta velocidad (**bus de instrumentación de propósito general**). La figura 11-31 muestra la estructura interna y el diagrama de sincronización para la operación bidireccional en modo 2.

Definiciones de señales para el modo 2 bidireccional

- | | |
|-------------|---|
| INTR | Petición de interrupción es una salida que se utiliza para interrumpir al microprocesador basándose en las condiciones de entrada y de salida. |
| OBF | Búfer de salida lleno es una salida que indica que el búfer de salida contiene datos para el bus bidireccional. |
| ACK | Reconocimiento es una entrada que habilita los búferes de tres estados, de manera que puedan aparecer los datos en el puerto A. Si \overline{ACK} es un 1 lógico, los búferes de salida del puerto A se encuentran en su estado de alta impedancia. |
| STB | La entrada estrobo carga el enclavamiento de entrada del puerto A con datos externos que provienen del bus bidireccional del puerto A. |

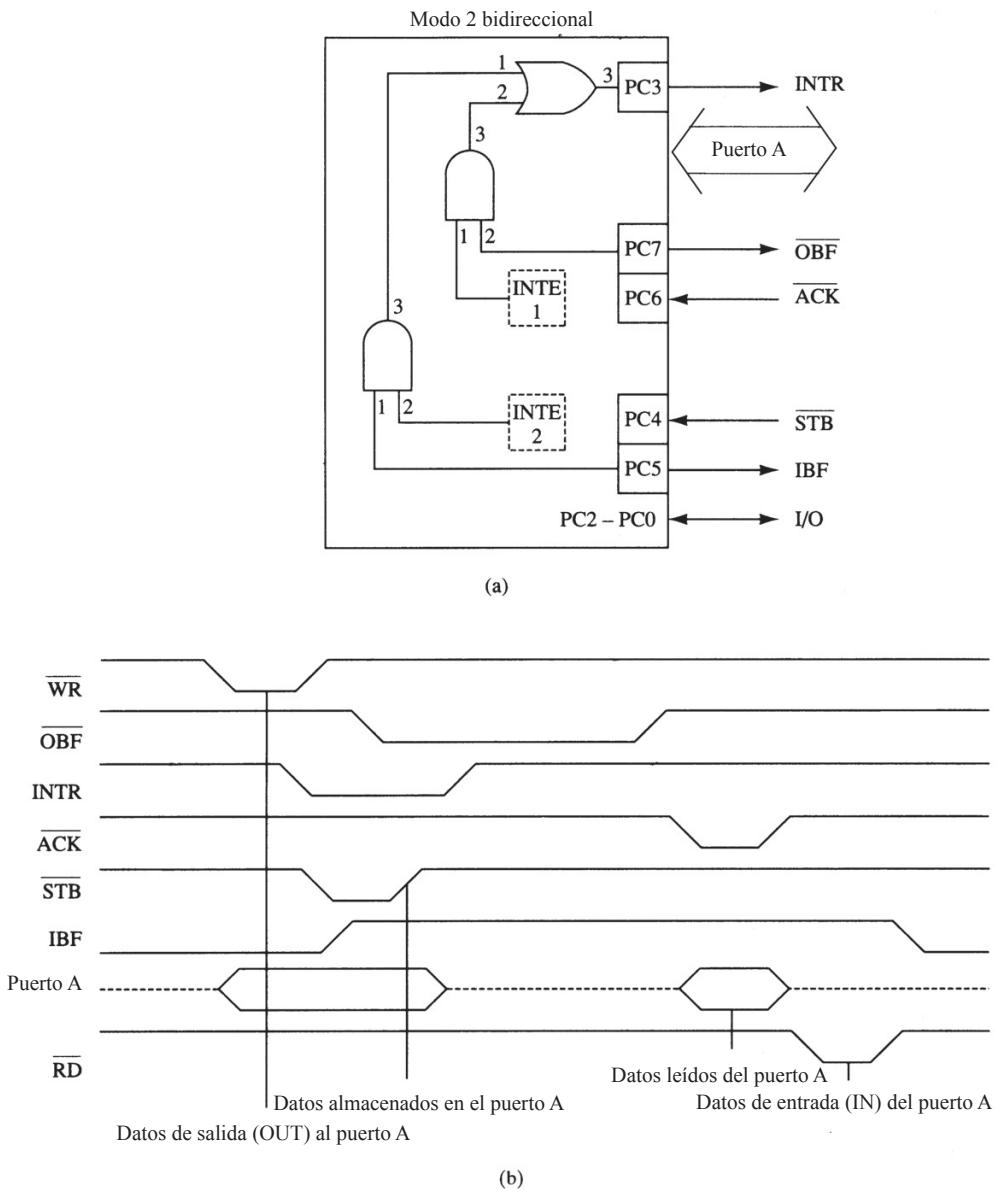


FIGURA 11-31 Operación del 82C55 en modo 2. (a) Estructura interna y (b) diagrama de sincronización.

- IBF** **Búfer de entrada lleno** es una salida que se utiliza para indicar que el búfer de entrada contiene datos para el bus bidireccional externo.
- INTE** **Habilitación de interrupción** está compuesta por los bits internos (INTE1 e INTE2) que habilitan la terminal INTR. El estado de esta terminal se controla a través de los bits PC₆ (INTE1) y PC₄ (INTE2) del puerto C.
- PC₀,PC₁ y PC₂** Estas terminales son de E/S de propósito general en el modo 2, y se controlan mediante el comando para activar y borrar bits.

El bus bidireccional. Este bus se utiliza mediante una referencia al puerto A con las instrucciones IN y OUT. Para transmitir datos a través del bus bidireccional, el programa primero evalúa la señal \overline{OBF} para

determinar si el búfer de salida está vacío. Si lo está, los datos se envían al búfer de salida mediante la instrucción OUT. El circuito interno también monitorea la señal OBF para decidir si el microprocesador ha enviado datos al bus. En el instante en que el circuito de salida ve un 0 lógico en OBF, envía de regreso la señal ACK para extraer los datos del búfer de salida. La señal ACK activa el bit OBF y habilita los búferes de salida de tres estados, de manera que puedan leerse datos. El ejemplo 11-20 muestra un procedimiento que transmite el contenido del registro AH a través del puerto A bidireccional.

EJEMPLO 11-20

;Un procedimiento que transmite AH a través del bus bidireccional

```

BIT7      EQU 80H
PUERTOC   EQU 62H
PUERTOA   EQU 60H

TRANS     PROC NEAR

    .REPEAT           ;evalúa OBF
        IN   AL,PUERTOC
        TEST AL,BIT7
    .UNTIL !ZERO?
        MOV  AL,AH          ;envía los datos
        OUT  PUERTOA,AL
        RET

TRANS     ENDP

```

Para recibir datos a través del bus bidireccional del puerto A, se evalúa el bit IBF mediante software para decidir si los datos se han enviado al puerto mediante estrobo. Si IBF = 1, los datos se envían mediante la instrucción IN. La interfaz externa envía datos al puerto mediante el uso de la señal STB. Cuando STB se activa, la señal IBF se vuelve un 1 lógico y los datos en el puerto A se guardan en un enclavamiento dentro del puerto. Cuando se ejecuta la instrucción IN, el bit IBF se borra y los datos en el puerto se pasan a AL. El ejemplo 11-21 muestra un procedimiento que lee datos del puerto.

EJEMPLO 11-21

;Un procedimiento que lee datos del bus bidireccional hacia AL

```

BIT5      EQU 20H
PUERTOC   EQU 62H
PUERTOA   EQU 60H

LEE      PROC NEAR

    .REPEAT           ;evalúa IBF
        IN   AL,PUERTOC
        TEST AL,BITS5
    .UNTIL !ZERO?
        IN   AL,PUERTOA
        RET

LEE      ENDP

```

La terminal INTR (**petición de interrupción**) puede activarse desde ambas direcciones del flujo de datos a través del bus. Si INTR se habilita mediante ambos bits INTE, entonces tanto el búfer de salida como el de entrada producirán peticiones de interrupción. Esto ocurre cuando los datos se envían en forma de estrobo hacia el búfer, mediante el uso de STB o cuando se escriben datos mediante OUT.

FIGURA 11-32

Un resumen de las conexiones de puertos para el PIA 82C55.

		Modo 0		Modo 1		Modo 2
Puerto A		ENTRADA	SALIDA	ENTRADA	SALIDA	E/S
Puerto B		ENTRADA	SALIDA	ENTRADA	SALIDA	No utilizado
Puerto C	0			INTR _B	INTR _B	E/S
	1			IBF _B	\overline{OBF}_B	E/S
	2			\overline{STB}_B	ACK _B	E/S
	3	ENTRADA	SALIDA	INTR _A	INTR _A	INTR
	4			\overline{STB}_A	E/S	STB
	5			IBF _A	E/S	IBF
	6			E/S	\overline{ACK}_A	\overline{ACK}
	7			E/S	\overline{OBF}_A	\overline{OBF}

Resumen de los modos del 82C55

Observe en la figura 11-32 un resumen gráfico de los tres modos de operación para el 82C55. El modo 0 proporciona E/S simple, el modo 1 proporciona E/S estroboscópica y el modo 2 proporciona E/S bidireccional. Como se mencionó antes, estos modos se seleccionan a través del registro de comandos del 82C55.

11-4

EL TEMPORIZADOR 8254 DE INTERVALOS PROGRAMABLES

El temporizador 8254 de intervalos programables consiste de tres contadores (**temporizadores**) programables de 16 bits. Cada contador es capaz de contar en binario o en decimal codificado en binario (BCD). La máxima frecuencia de entrada permitida para cualquier contador es de 10 MHz. Este dispositivo es útil cada vez que el microprocesador debe controlar eventos en tiempo real. Algunos ejemplos de uso son: un reloj de tiempo real, un contador de eventos y para el control de velocidad y dirección de un motor.

Este temporizador aparece también decodificado en la computadora personal en los puertos 40H-43H, para las siguientes funciones:

1. Genera una interrupción de temporizador básica que ocurre a un valor aproximado de 18.2 Hz.
2. Hace que el sistema de memoria DRAM se refresque.
3. Proporciona una fuente de sincronización para la bocina interna y otros dispositivos. El temporizador en la computadora personal es un 8253, en vez de 8254.

Descripción funcional del 8254

La figura 11-33 muestra el diagrama de terminales del 8254, que es una versión del 8253 de mayor velocidad, y un diagrama de uno de los tres contadores. Cada temporizador contiene una entrada CLK, una entrada de compuerta y una conexión de salida (OUT). La entrada CLK proporciona la frecuencia básica de operación para el temporizador, la terminal de compuerta controla el temporizador en ciertos modos y la terminal OUT es donde obtenemos la salida del temporizador.

Las señales que se conectan al microprocesador son las terminales del bus de datos (D₇-D₀), \overline{RD} , \overline{WR} , CS y las entradas de dirección A₁ y A₀. Las entradas de dirección están presentes para seleccionar cualquiera de los cuatro registros internos que se utilizan para programar, leer o escribir en un contador. La computadora personal contiene un temporizador 8253 o su equivalente, decodificado en los puertos de E/S 40H-43H. El temporizador cero se programa para generar una señal de 18.2 Hz que interrumpe el microprocesador en el vector de interrupción 8 para un pulso del reloj. Este pulso se utiliza a menudo para sincronizar los programas y eventos en el DOS. El temporizador 1 se programa para 15 μ s y se utiliza en la computadora personal para solicitar una acción DMA que se utiliza para refresh la RAM dinámica. El temporizador 2 se programa para generar un tono en la bocina de la computadora personal.

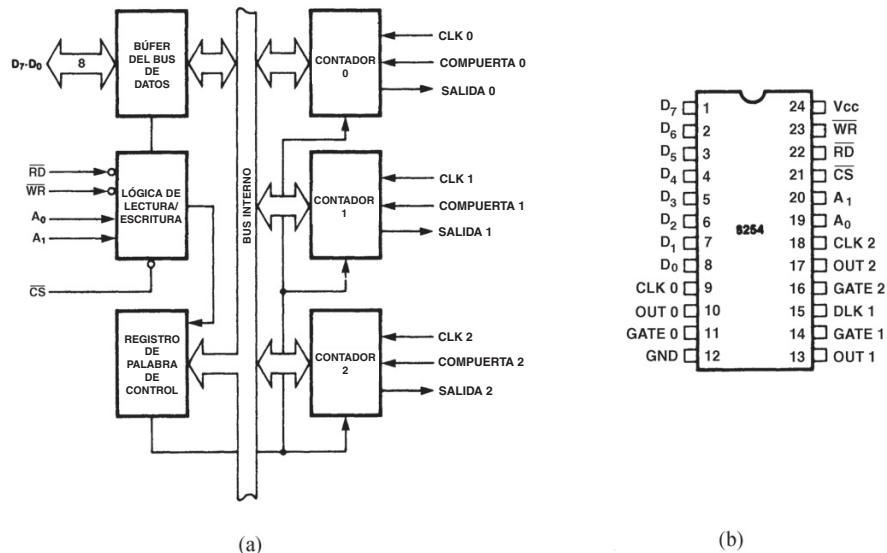


FIGURA 11-33 El temporizador 8254 de intervalos programables. (a) Estructura interna y (b) diagrama de terminales. (Cortesía de Intel Corporation.)

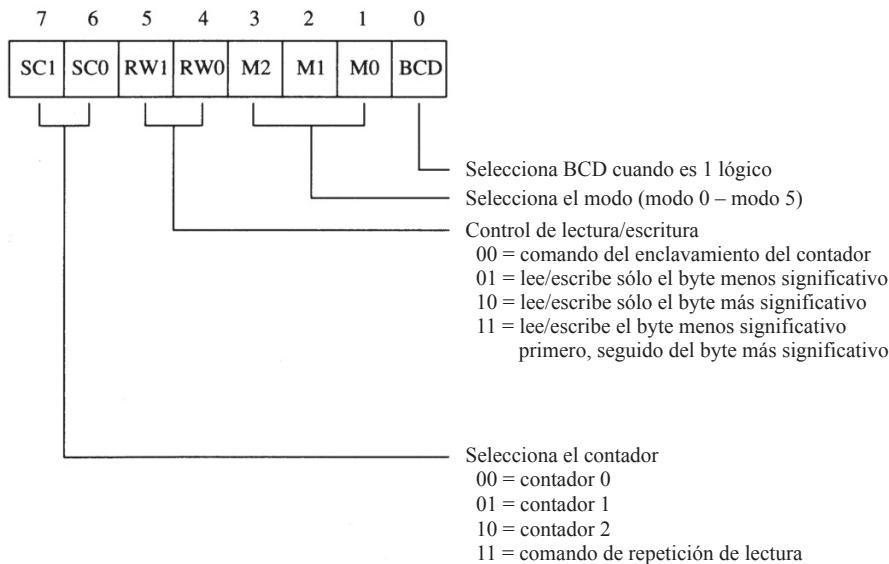
Definiciones de las terminales

A_0-A_1	Las entradas de dirección seleccionan uno de cuatro registros internos dentro del 8254. En la tabla 11-4 podrá consultar la función de los bits de dirección A_1 y A_0 .
CLK	La entrada reloj es la fuente de sincronización para cada uno de los contadores internos. Por lo general, esta entrada se conecta a la señal PCLK del controlador del bus de sistema del microprocesador.
CS	Selección de chip habilita el 8254 para la programación y para leer o escribir en un contador.
G	La entrada de compuerta controla la operación del contador en ciertos modos de operación.
GND	Tierra se conecta a tierra del bus del sistema.
OUT	Una salida de contador es en donde está disponible la forma de onda generada por el temporizador.
RD	Lectura hace que se lean datos del 8254 y a menudo se conecta a la señal \overline{IORC} .
VCC	La entrada Energía se conecta a la fuente de energía de +5.0 V.
WR	Escritura hace que se escriban datos en el 8254 y a menudo se conecta al estrobo de escritura (\overline{IOWC}).

TABLA 11-4 Entradas de selección de dirección para el 8254.

A_1	A_0	Función
0	0	Contador 0
0	1	Contador 1
1	0	Contador 2
1	1	Palabra de control

FIGURA 11-34 La palabra de control para el temporizador 8254-2.



Programación del 8254

Cada contador se programa en forma individual mediante la escritura de una palabra de control seguida de la cuenta inicial. En la figura 11-34 podrá ver la estructura de palabra de control de programa del 8254. La **palabra de control** permite que el programador seleccione el contador, el modo y el tipo de operación (lectura/escritura). La palabra de control también selecciona una cuenta binaria o BCD. Cada contador puede programarse con una cuenta de 1 a FFFFH. Una cuenta de 0 es igual a FFFFH+1 (65,536) o 10,000 en BCD. La cuenta mínima de 1 se aplica a todos los modos de operación excepto los modos 2 y 3, que tienen una cuenta mínima de 2. El temporizador 0 se utiliza en la computadora personal con una cuenta de división de 64 K (FFFFH) para generar el pulso del reloj de interrupción de 18.2 Hz (18.196 Hz). El temporizador 0 tiene una frecuencia de entrada de reloj de 4.77 MHz + 4 o 1.1925 MHz.

La palabra de control utiliza el bit BCD para seleccionar una cuenta BCD (BCD = 1) o una cuenta binaria (BCD = 0). Los bits M₂, M₁ y M₀ seleccionan uno de los seis modos distintos de operación (000-101) para el contador. Los bits RW₁ y RW₀ determinan cómo se leen o se escriben los datos en el contador. Los bits SC₁ y SC₀ seleccionan un contador o el modo especial de operación de repetición de lectura, el cual veremos más adelante en esta sección.

Cada contador tiene una palabra de control de programa que se utiliza para seleccionar la manera en que opera. Si se programan dos bytes en un contador, entonces el primer byte (LSB) lo detendrá y el segundo byte (MSB) iniciará el contador con la nueva cuenta. El orden de programación es importante para cada contador, pero la programación de distintos contadores puede interpaginarse para un mejor control. Por ejemplo, puede enviarse la palabra de control a cada contador antes de las cuentas para su programación individual. El ejemplo 11-22 muestra unas cuantas formas de programar los contadores 1 y 2. El primer método programa ambas palabras de control, después el LSB de la cuenta para cada contador, el cual hace que los contadores dejen de contar. Por último se programa la porción correspondiente al MSB del contador, lo cual inicia ambos contadores con la nueva cuenta. El segundo ejemplo muestra la programación de un contador después de otro.

EJEMPLO 11-22

```
PROGRAM CONTROL WORD 1 PROGRAM CONTROL WORD 2 PROGRAM LSB 1
PROGRAM LSB 2
PROGRAM MSB 1
PROGRAM MSB 2
;establece el contador 1
;establece el contador 2
```

```

;detiene el contador 1 y programa el LSB
;detiene el contador 2 y programa el LSB; programa el MSB del contador 1 y lo inicia
;programa el MSB del contador 2 y lo inicia

o

PROGRAM CONTROL WORD 1 PROGRAM LSB 1
PROGRAM MSB 1
PROGRAM CONTROL WORD 2 PROGRAM LSB 2
PROGRAM MSB 2

;establece el contador 1
;detiene el contador 1 y programa el LSB ;programa el MSB del contador 1 y lo inicia
;establece el contador 2
;detiene el contador 2 y programa el LSB ;programa el MSB del contador 2 y lo inicia

```

Modos de operación. Hay seis modos (modo 0-modo 5) de operación disponibles para cada uno de los contadores 8254. Observe en la figura 11-35 cómo funciona cada uno de estos modos con la entrada CLK, la señal de control de compuerta (G) y la señal OUT. A continuación se muestra una descripción de cada modo:

- MODO 0** Permite que el contador 8254 se utilice como un contador de eventos. En este modo la salida se vuelve un 0 lógico cuando se escribe la palabra de control y permanece en ese nivel hasta N más el número de cuentas programadas. Por ejemplo, si se programa una cuenta de 5 la salida permanecerá en 0 lógico durante 6 cuentas, empezando con N. La entrada de compuerta (G) debe ser un 1 lógico para permitir que el contador cuente. Si G se vuelve un 0 lógico a mitad de la cuenta, el contador se detendrá hasta que G vuelva a ser un 1 lógico.
- MODO 1** Hace que el contador funcione como un multivibrador monoestable redispersable (one-shot). En este modo, la entrada G dispara el contador de manera que desarrolle un pulso en la conexión OUT, que se vuelve un 0 lógico durante toda la cuenta. Si la cuenta es de 10, entonces la conexión OUT se va al nivel bajo durante 10 períodos de reloj cuando se dispara. Si la entrada G ocurre durante el pulso de entrada, el contador se vuelve a cargar con la cuenta y la conexión OUT continúa durante toda la cuenta.
- MODO 2** Permite que el contador genere una serie de pulsos continuos cuya anchura es de un pulso de reloj. La separación entre los pulsos se determina mediante la cuenta. Por ejemplo, para una cuenta de 10 la salida es un 1 lógico durante nueve períodos de reloj y se va a nivel bajo durante un período de reloj. Este ciclo se repite hasta que el contador se programa con una nueva cuenta o hasta que la terminal G se coloca en nivel de 0 lógico. La entrada G debe ser un 1 lógico para que este modo genere una serie continua de pulsos.
- MODO 3** Genera una onda cuadrada continua en la conexión OUT, siempre y cuando la terminal G sea un 1 lógico. Si la cuenta es par, la salida se va al nivel alto durante la mitad de la cuenta y se va a nivel bajo durante la otra mitad. Si la cuenta es impar, la salida dura un período de reloj más en nivel alto que en bajo. Por ejemplo, si el contador se programa para una cuenta de 5, la salida se va al nivel alto durante tres períodos de reloj y se va al nivel bajo durante dos períodos de reloj.
- MODO 4** Permite que el contador produzca un solo pulso en la salida. Si la cuenta se programa como 10, la salida se va al nivel alto durante 10 períodos de reloj y se va al nivel bajo durante un período de reloj. El ciclo no empieza sino hasta que el contador esté cargado con su cuenta completa. Este modo opera como un “one-shot” disparado por software. Al igual que los modos 2 y 3, este modo también utiliza la entrada G para habilitar el contador. La entrada G debe ser un 1 lógico para que el contador opere para estos tres modos.
- MODO 5** Un “one-shot” disparado por hardware que funciona como el modo 4, sólo que se inicia mediante un pulso de disparo en la terminal G, en vez de software. Este modo también es similar al modo 1 ya que es redispersable.

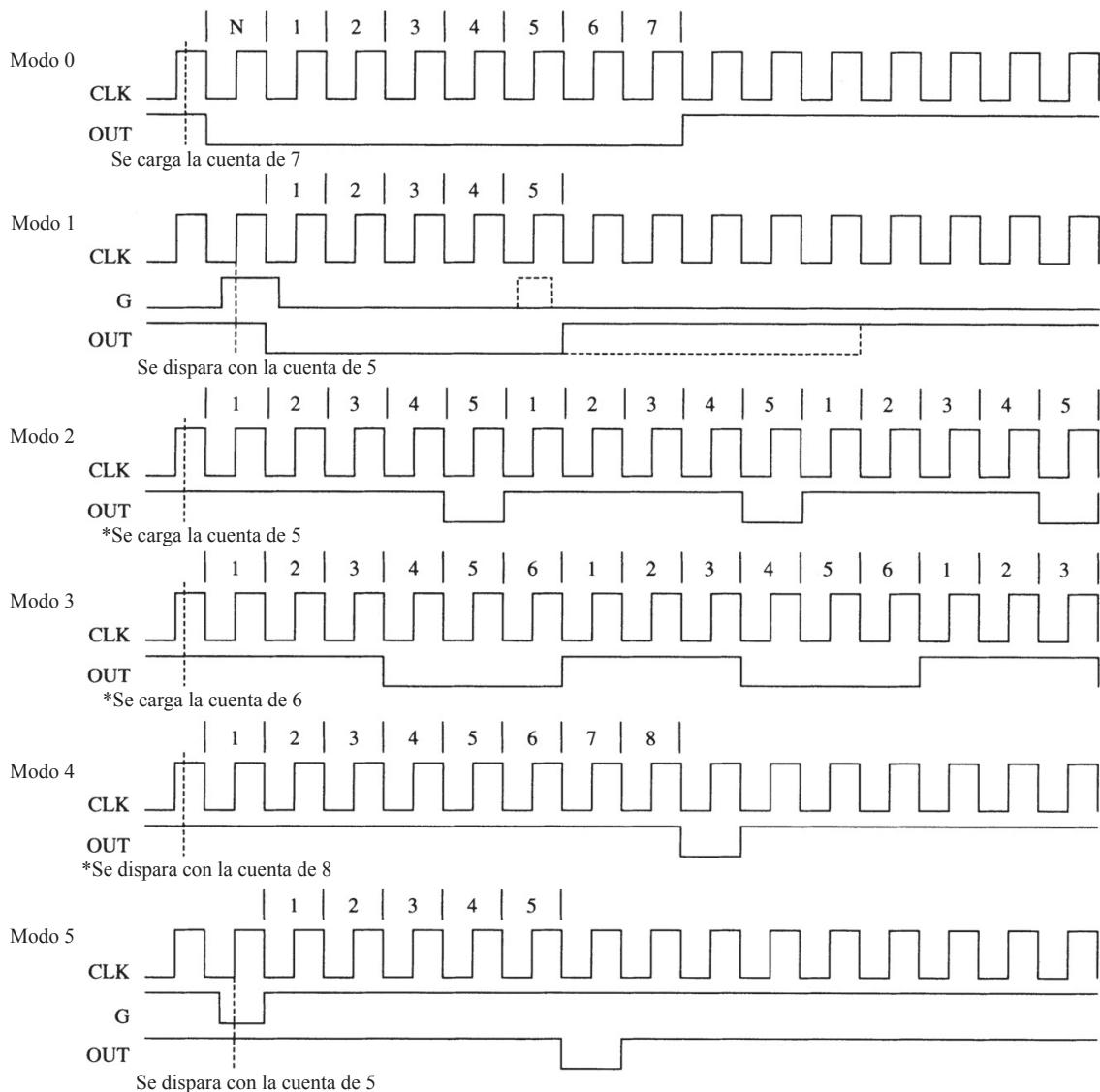


FIGURA 11-35 Los seis modos de operación para el temporizador 8254-2 de intervalos programables. *La entrada G detiene la cuenta cuando es 0 en los modos 2, 3 y 4.

Generación de una forma de onda con el 8254. La figura 11-36 muestra un 8254 conectado para funcionar en los puertos de E/S 0700H, 0702H, 0704H y 0706H de un microprocesador 80386SX. Las direcciones se decodifican mediante el uso de un PLD que también genera una señal de estrobo de escritura para el 8254, la cual se conecta a las conexiones de menor orden del bus de datos. El PLD también genera una señal de espera para el microprocesador, la cual produce dos estados de espera cuando se accede al 8254. El generador de estados de espera conectado al microprocesador es el que controla el número de estados de espera que se insertan en la sincronización. No mostraremos aquí el programa para el PLD, ya que es el mismo que muchos de los ejemplos anteriores.

Observe en el ejemplo 11-23 el programa que genera una onda cuadrada de 100 KHz en OUT0 y un pulso continuo de 200 KHz en OUT1. El contador 0 utiliza el modo 3 y el contador 1 utiliza el modo 2. La cuenta que se programa en el contador 0 es de 80 y la cuenta para el contador 1 es de 40. Estas cuentas generan las frecuencias de salida deseadas con un reloj de entrada de 8 MHz.

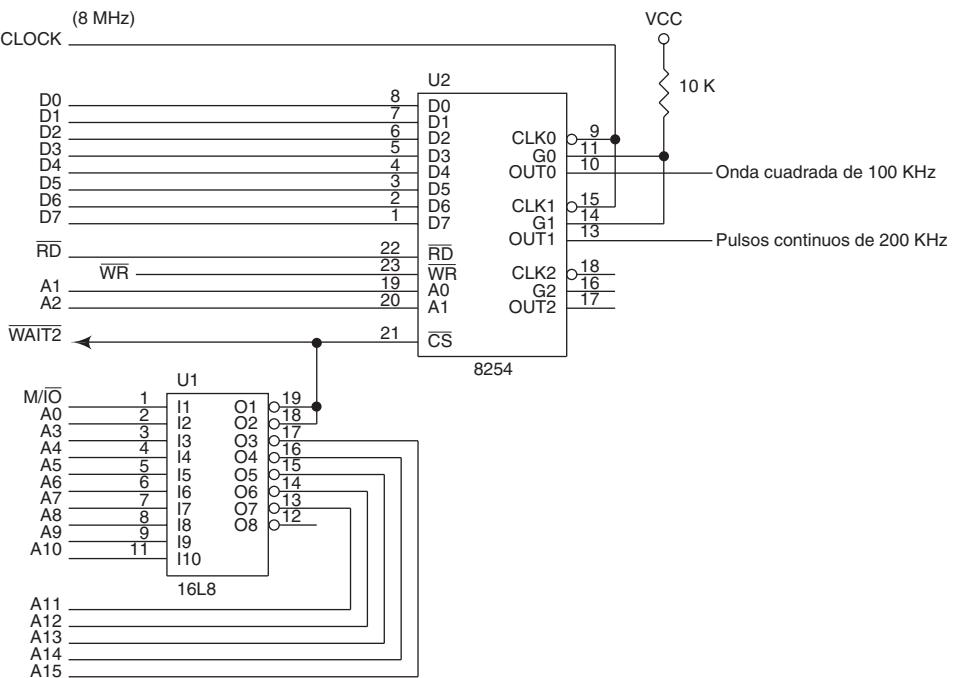


FIGURA 11-36 El 8254 conectado a un 8086 de 8 MHz, de manera que genere una onda cuadrada de 100 KHz en OUT0 y un pulso continuo de 200 KHz en OUT1.

EJEMPLO 11-23

;Un procedimiento que programa el temporizador 8254 para funcionar
;como se indica en la figura 11-34

```

TEMPO PROC NEAR USES AX DX
    MOV DX,706H ;programa el contador 0 para el modo 3
    MOV AL,00110110B
    OUT DX,AL
    MOV AL,01110100B ;programa el contador 1 para el modo 2
    OUT DX,AL

    MOV DX,700H ;programa el contador 0 con 80
    MOV AL,80
    OUT DX,AL
    MOV AL,0
    OUT DX,AL

    MOV DX,702H ;programa el contador 1 con 40
    MOV AL,40
    OUT DX,AL
    MOV AL,0
    OUT DX,AL

    RET
TEMPO ENDP

```

Lectura de un contador. Cada contador tiene un enclavamiento interno que se lee mediante la operación de lectura del puerto del contador. Por lo general, estos enclavamientos siguen al contador. Si se necesita el contenido del contador, entonces el enclavamiento puede recordar la cuenta mediante la programación de la palabra de control del enclavamiento del contador (vea la figura 11-37), la cual hace que el contenido del contador se guarde en un enclavamiento hasta que se lea.

FIGURA 11-37 La palabra de control del enclavamiento del contador 8254-2.

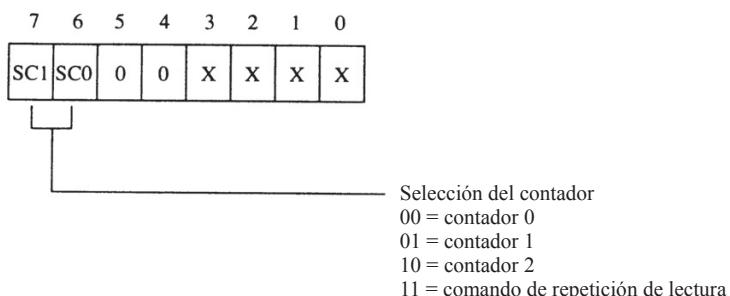


FIGURA 11-38 La palabra de control de repetición de lectura del 8254-2.

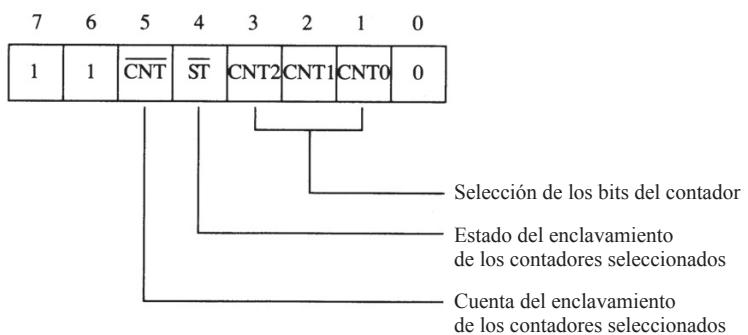
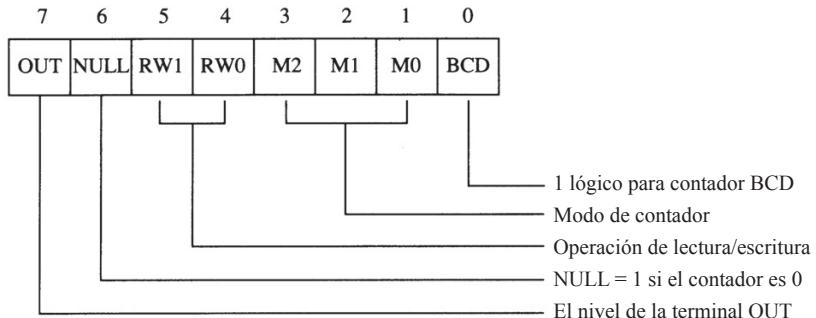


FIGURA 11-39 El registro de estado del 8254-2.



Cada vez que se programa una lectura del enclavamiento o del contador, el enclavamiento rastrea el contenido del contador.

Cuando es necesario que se lea el contenido de más de un contador al mismo tiempo utilizamos la palabra de control de repetición de lectura, la cual se muestra en la figura 11-38. Con esta palabra de control, el bit $\overline{\text{CNT}}$ es un 0 lógico para que los contadores seleccionados mediante CNT0, CNT1 y CNT2 se enclaven. Si se va a enclavar el registro de estado, entonces el bit $\overline{\text{ST}}$ se coloca en 0 lógico. En la figura 11-39 puede ver el registro de estado, el cual muestra varias cosas: el estado de la terminal de salida, si el contador está en su estado nulo (0) y cómo se programa.

Control de la velocidad y la dirección de un motor de CD

El temporizador 8254 puede utilizarse como un controlador de la velocidad de un motor de CD. La figura 11-40 muestra el diagrama esquemático del motor y de sus circuitos controladores asociados. También muestra la interconexión del 8254, un flip-flop y el motor junto con su controlador.

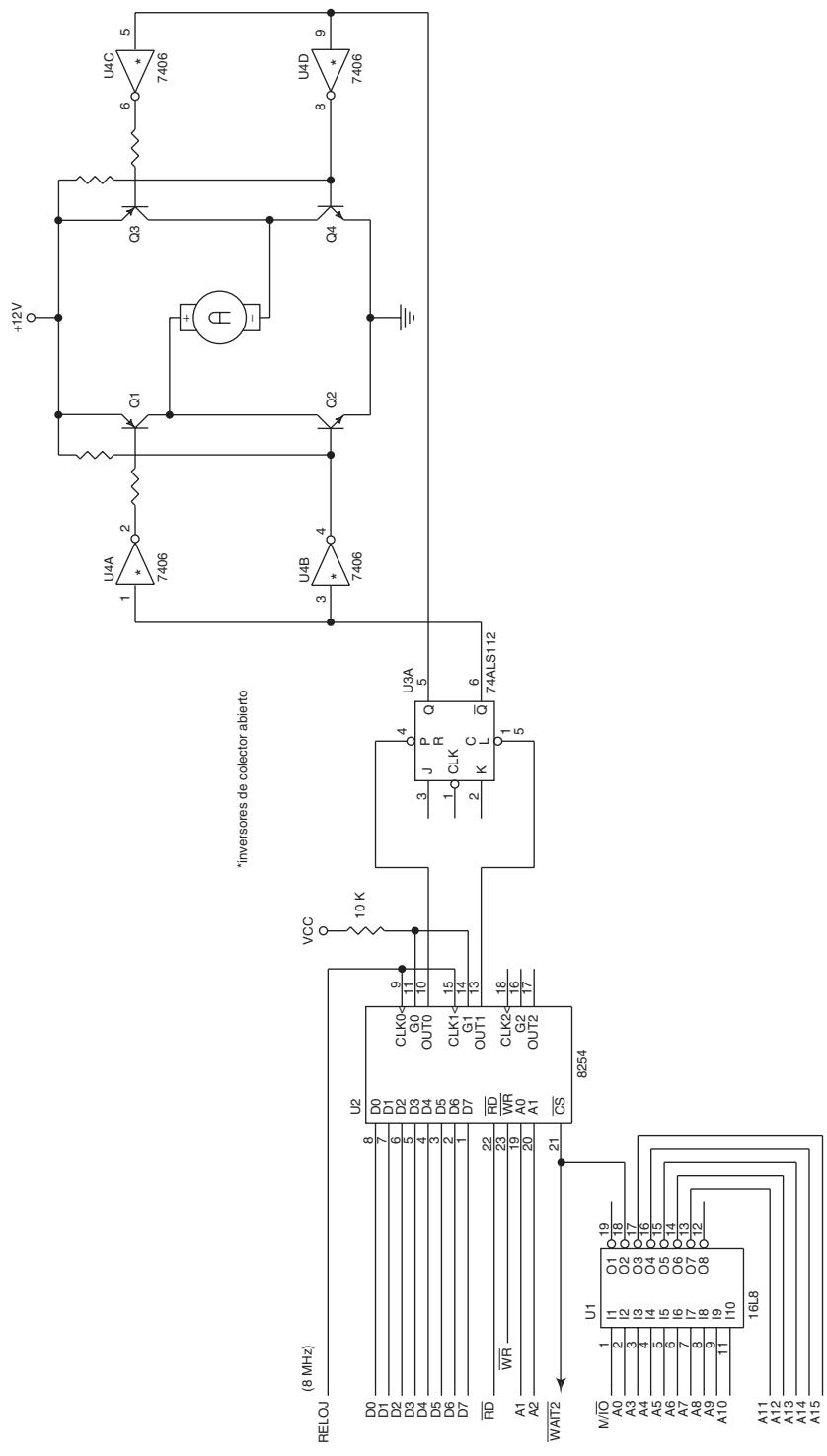


FIGURA 11-40 Control de la velocidad y dirección de un motor mediante el uso del temporizador 8254.

La operación del circuito controlador del motor es simple. Si la salida Q del 74ALS112 es un 1 lógico, la base Q₂ se lleva hasta +12 V a través de la resistencia elevadora de la base y queda como circuito abierto. Esto significa que Q₁ se desconecta y Q₂ se activa, y se aplica tierra a la terminal positiva del motor. Las bases de Q₃ y de Q₄ llevan a nivel bajo para aterrizarlas a través de los inversores. Esto hace que Q₃ conduzca o se active y que Q₄ se desconecte, con lo que se aplica tierra a la terminal negativa del motor. Por lo tanto, el 1 lógico en la salida Q del flip-flop conecta +12 V a la terminal positiva del motor y aterra la terminal negativa. Esta conexión hace que el motor gire hacia delante. Si el estado de la salida Q del flip-flop se vuelve un 0 lógico, entonces se invierten las condiciones de los transistores y se conectan +12 V a la terminal negativa del motor; además la terminal positiva se aterra. Esto hace que el motor gire en dirección inversa.

Si la salida del flip-flop se alterna entre un 1 y un 0 lógico, el motor gira en ambas direcciones a diversas velocidades. Si el ciclo de trabajo de la salida Q es del 50%, el motor no girará y exhibirá cierto momento de torsión de sujeción, ya que la corriente fluye a través de él. La figura 11-41 muestra

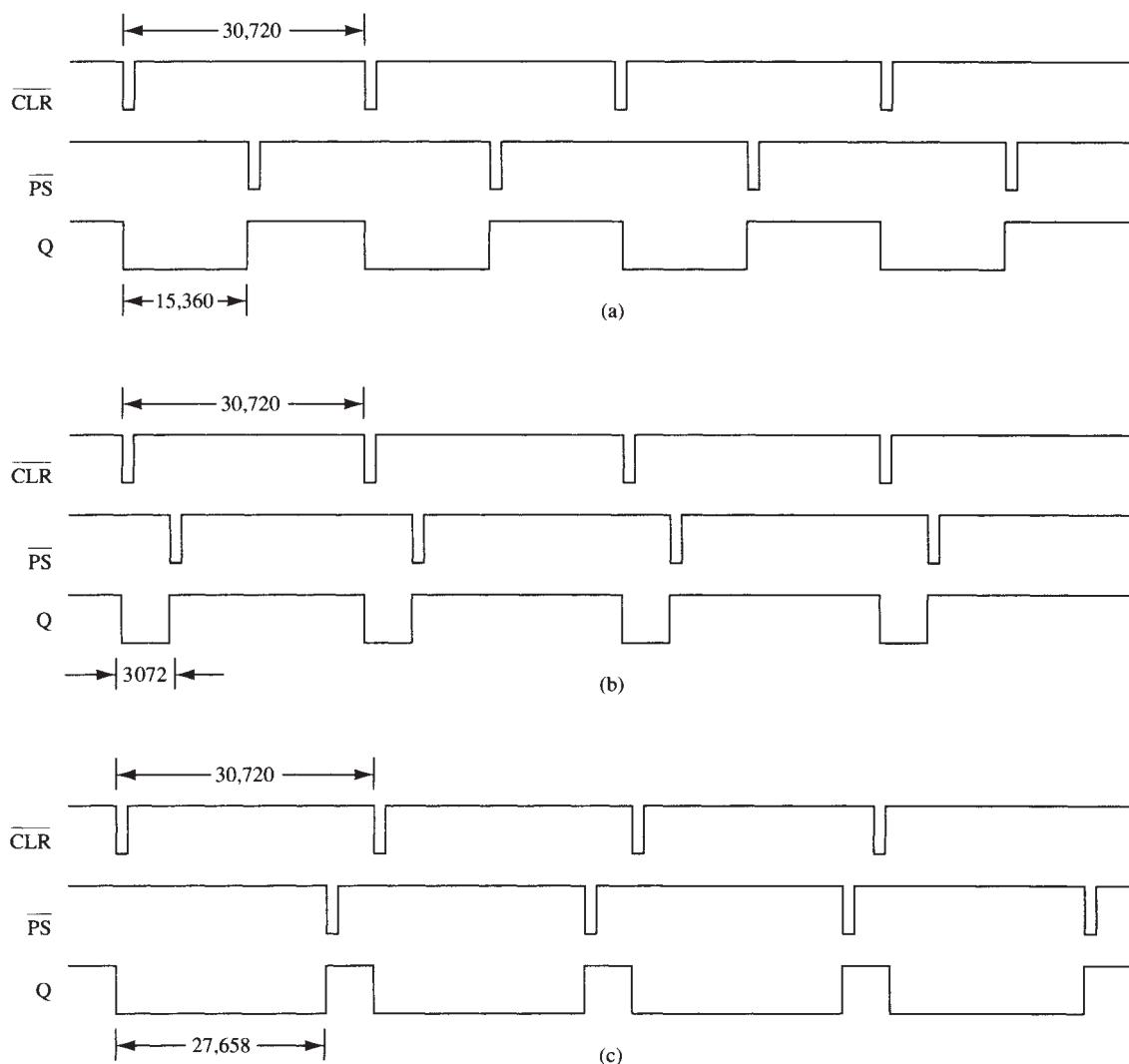


FIGURA 11-41 Sincronización para el circuito de control de velocidad y dirección del motor de la figura 11-45. (a) Sin rotación, (b) rotación de alta velocidad en dirección inversa y (c) rotación de alta velocidad hacia delante.

algunos diagramas de sincronización y sus efectos sobre la velocidad y la dirección del motor. Observe cómo cada contador genera pulsos en distintas posiciones para variar el ciclo de trabajo en la salida Q del flip-flop. A esta salida también se le conoce como *modulación de amplitud de pulso*.

Para generar estas formas de onda, los contadores 0 y 1 se programan para dividir el reloj de entrada (PCLK) entre 30,720. Para cambiar el ciclo de trabajo de Q se cambia el punto en el que se inicia el contador 1, con relación al contador 0. Esta acción cambia la dirección y la velocidad del motor. Pero ¿por qué se divide el reloj de 8 MHz entre 30,720? Este número es divisible entre 256, por lo que podemos desarrollar un programa corto que permita 256 velocidades distintas. Esto también produce una frecuencia básica de operación para el motor de aproximadamente 260 Hz, la cual es lo bastante baja como para poder operar el motor. Es importante mantener esta frecuencia de operación menor a 1000 Hz, pero mayor a 60 Hz.

Observe en el ejemplo 11-24 un procedimiento que controla la velocidad y la dirección del motor. La velocidad se controla mediante el valor de AH cuando se hace una llamada a este procedimiento. Como tenemos un número de 8 bits para representar la velocidad, un ciclo de trabajo del 50% para un motor detenido representa una cuenta de 128. Si cambiamos el valor en AH cuando se hace la llamada al procedimiento, podremos ajustar la velocidad del motor. Ésta aumentará en cualquier dirección si se cambia el número en AH cuando se haga la llamada a este procedimiento. A medida que el valor en AH se aproxime a 00H, el motor empezará a incrementar su velocidad en la dirección inversa. A medida que AH se aproxime a FFH, el motor incrementará su velocidad en dirección hacia delante.

EJEMPLO 11-24

```
;Un procedimiento que controla la velocidad y la dirección del motor
;de la figura 11-40.
;
;AH determina la velocidad y la dirección del motor, en donde
;AH está entre 00H y FFH.

CNTR    EQU  706H
CNT0    EQU  700H
CNT1    EQU  702H
CUENTA  EQU  30720

VELOC  PROC NEAR USES BX DX AX

        MOV  BL,AH           ;calcula la cuenta
        MOV  AX,120
        MOV  BL
        MOV  BX,AX
        MOV  AX,CUENTA
        SUB  AX,BX
        MOV  BX,AX

        MOV  DX,CNTR
        MOV  AL,00110100B      ;programa las palabras de control
        OUT  DX,AL
        MOV  AL,01110100B
        OUT  DX,AL

        MOV  DX,CNT1          ;programa el contador 1
        MOV  AX,CUENTA         ;para generar una señal de borrado
        OUT  DX,AL
        MOV  AL,AH
        OUT  DX,AL

.REPEAT
        IN   AL,DX
        XCHG AL,AH
        IN   AL,DX
        XCHG AL,AH
.UNTIL    BX == AX

        MOV  DX,CNT0          ;programa el contador 0
        MOV  AX,CUENTA         ;para generar una activación
```

```

    OUT  DX, AL
    MOV  AL, AH
    OUT  DX, AL

    RET

VELOC    ENDP

```

Para ajustar la forma de onda en Q, este procedimiento primero calcula la cuenta con la que va a iniciar el contador 0, en relación con el contador 1. Esto se logra mediante la multiplicación de AH por 120; después se restan 32,720 al resultado de esta multiplicación. Esto se requiere debido a que los contadores hacen un conteo descendente, partiendo de la cuenta programada hasta llegar a 0 antes de volver a iniciar. Después, el contador 1 se programa con una cuenta de 30,720 y se inicia de manera que genere la forma de onda de borrado para el flip-flop. Después de iniciar el contador 1, se lee y se compara con la cuenta calculada. Una vez que llegue a esta cuenta se inicia el contador 0 con una cuenta de 30,720. De este punto en adelante, ambos contadores continúan generando las formas de onda de borrado y de activación hasta que se vuelva a hacer otra llamada al procedimiento para ajustar la velocidad y la dirección del motor.

11-5

INTERFAZ DE COMUNICACIONES PROGRAMABLE 16550

El componente PC16550D de National Semiconductor Corporation es una interfaz de comunicaciones programable, diseñada para conectarse con casi cualquier tipo de interfaz serial. El 16550 es un receptor/transmisor universal asíncrono (UART) 100% compatible con los microprocesadores Intel. Este dispositivo es capaz de operar a 0-1.5 Mbaudios. La velocidad de transmisión en baudios es el número de bits transferidos por segundo (bps), incluyendo los de inicio, de paro, de datos y de paridad. (Bps significa bytes por segundo y bps es bits por segundo.) El 16550 también incluye un generador de velocidad en baudios programable y FIFOs individuales para entrada y salida de datos, para aligerar la carga del microprocesador. Cada FIFO contiene 16 bytes de almacenamiento. Ésta es la interfaz de comunicaciones más común en el equipo basado en microprocesadores, incluyendo la computadora personal y muchos módems.

Datos asíncronos en serie

Este tipo de datos se transmite y se recibe sin necesidad de una señal de reloj o de sincronización. La figura 11-42 muestra dos tramas de datos asíncronos en serie. Cada trama contiene un bit de inicio, siete bits de datos, uno de paridad y un bit de paro. La figura muestra una trama que contiene un carácter ASCII y 10 bits. La mayoría de los sistemas de comunicaciones de acceso telefónico del pasado, como CompuServe, Prodigy y America Online, utilizaban 10 bits para los datos asíncronos en serie con paridad par. La mayoría de los servicios de Internet y de tablones de anuncios también utilizan 10 bits, pero por lo general no utilizan paridad, sino que se transfieren ocho bits de datos para sustituir el bit de paridad con un bit de datos. Esto hace que las transferencias de bytes de datos que no sean código ASCII sean más fáciles de realizar.

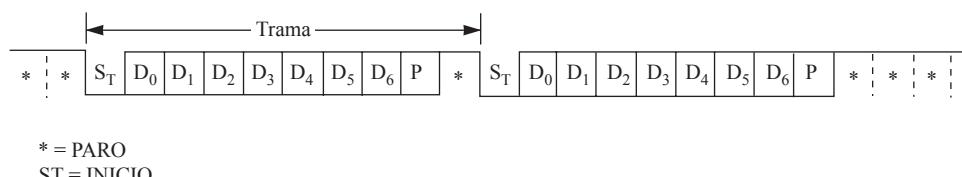
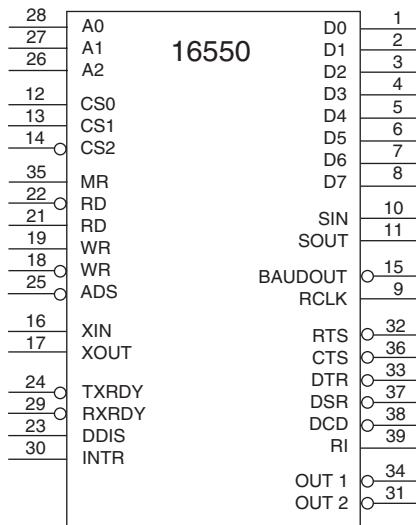


FIGURA 11-42 Datos asíncronos en serie.

FIGURA 11-43 Diagrama de terminales del UART 16550.



Descripción funcional del 16550

Observe en la figura 11-43 el diagrama de terminales del UART 16550. Este dispositivo está disponible como DIP (**paquete dual en línea**) de 40 terminales o como PLCC (**portador de chip sin plomo de plástico**) de 44 terminales. Hay dos secciones completamente separadas que son las responsables de las comunicaciones de datos: el receptor y el transmisor. Como cada una de estas secciones es independiente, el 16550 puede funcionar en modos simplex, half-duplex o full-duplex. Una de las principales características del 16550 son las memorias FIFO (primero en entrar, primero en salir) de su receptor y de su transmisor internos. Como cada una es de 16 bytes, el UART sólo requiere atención del microprocesador hasta después de haber recibido 16 bytes de datos. También guarda 16 bytes antes de que el microprocesador deba esperar al transmisor. La memoria FIFO hace a este UART ideal cuando se conecta a los sistemas de alta velocidad, ya que se requiere menos tiempo para atenderlo.

Un ejemplo de un sistema **simplex** es cuando el transmisor o receptor se utiliza por sí solo, como en una estación de radio FM (**modulación de frecuencia**). Un ejemplo de un sistema **half-duplex** es el radio CB (**banda de ciudadanos**), en el que podemos transmitir y recibir, pero no al mismo tiempo. El sistema **full-duplex** permite la transmisión y la recepción en ambas direcciones, en forma simultánea. Un ejemplo de un sistema full-duplex es el teléfono.

El 16550 puede controlar un **módem** (**modulador/demodulador**), el cual es un dispositivo que convierte los niveles TTL de los datos seriales en tonos de audio que pueden pasar a través del sistema telefónico. Hay seis terminales en el 16550 dedicadas al control del módem: **DSR** (**conjunto de datos listo**), **DTR** (**terminal de datos lista**), **CTS** (**libre para enviar**), **RTS** (**peticIÓN de envÍo**), **RI** (**indicador de timbre**) y **DCD** (**detección de portadora de datos**). Al módem se le conoce como el **conjunto de datos** y al 16550 se le conoce como la **terminal de datos**.

Funciones de las terminales del 16550

- A₀, A₁, A₂** Las **entradas de dirección** se utilizan para seleccionar un registro interno para su programación y también para transferir datos. En la tabla 11-5 podrá ver una lista de cada una de las combinaciones de las entradas de dirección y los registros seleccionados.
- ADS** La entrada **estrobo de datos** se utiliza para enclavar las líneas de dirección y las de selección de chip. Si no se necesita (como en el sistema Intel), conecte esta terminal a tierra. La terminal **ADS** está diseñada para utilizarse con microprocesadores Motorola.
- BAUDOUT** La terminal **Salida en baudios** es en donde está disponible la señal de reloj generada por el generador de velocidad en Baudios para la sección del transmisor. Por lo general, se conecta a la entrada RCLK para generar un reloj receptor igual al reloj transmisor.

TABLA 11-5 Los registros seleccionados por A₀, A₁ y A₂.

A ₂	A ₁	A ₀	Función
0	0	0	Búfer receptor (lectura) y almacenamiento de transmisor (escritura)
0	0	1	Habilitación de interrupciones
0	1	0	Identificación de interrupción (lectura) y control de FIFO (escritura)
0	1	1	Control de línea
1	0	0	Control de módem
1	0	1	Estado de línea
1	1	0	Estado de módem
1	1	1	Residuo

CS₀, CS₁, CS₂	Las entradas selección de chip deben estar todas activas para habilitar el UART 16550.
CTS	La señal libre para enviar (si está en nivel bajo) indica que el módem o conjunto de datos está listo para intercambiar información. Esta terminal se utiliza a menudo en un sistema half-duplex para compartir la línea.
D₀-D₇	Las terminales bus de datos se conectan al bus de datos del microprocesador.
DCD	El módem utiliza la entrada detección de portadora de datos para indicar al 16550 que hay una portadora presente.
DDIS	La salida deshabilita controlador se vuelve un 0 lógico para indicar que el microprocesador va a leer datos del UART. DDIS puede usarse para modificar la dirección del flujo de datos a través de un búfer.
DSR	Conjunto de datos listo es una entrada que va al 16550, la cual indica que el módem o conjunto de datos está listo para operar.
DTR	Terminal de datos lista es una salida que indica que la terminal de datos (16550) está lista para funcionar.
INTR	Petición de interrupción es una salida que va al microprocesador y se utiliza para solicitar una interrupción (INTR = 1) cada vez que el 16550 tiene un error en el receptor, cuando ha recibido datos y cuando el transmisor está vacío.
MR	Reinicio maestro inicializa el 16550 y debe conectarse a la señal RESET del sistema.
OUT1, OUT2	Las terminales de salida definidas por el usuario pueden proporcionar señales a un módem o a cualquier otro dispositivo en un sistema, según sea necesario.
RCLK	Reloj receptor es la entrada de reloj que va a la sección del receptor del UART. Esta entrada es siempre 16 veces la velocidad en baudios deseada en el receptor.
RD, RD̄	Las entradas de lectura (puede usarse cualquiera) provocan la lectura de datos desde el registro especificado por las entradas de dirección, hacia el UART.
RI	El módem coloca la entrada indicador de timbre en el nivel de 0 lógico para indicar que el teléfono está timbrando.
RTS	Petición de envío es una señal para el módem que indica que el UART desea enviar datos.
SIN, SOUT	Éstas son las terminales de datos en serie . SIN acepta datos en serie y SOUT transfiere datos en serie.
RXRDY	Receptor listo es una señal que se utiliza para transferir los datos recibidos mediante técnicas de DMA (consulte el texto).
TXRDY	Transmisor listo es una señal que se utiliza para transferir los datos del transmisor mediante técnicas de DMS (consulte el texto).

- WR, \overline{WR}** Las terminales **Escritura** (puede usarse cualquiera) se conectan a la señal de escritura del microprocesador para transferir comandos y datos al 16550.
- XIN, XOUT** Éstas son las conexiones principales del **reloj**. Se conecta un cristal a través de estas terminales para formar un oscilador de cristal, o XIN se conecta a una fuente de sincronización externa.

Programación del 16550

La programación del 16550 es simple, aunque puede parecer un poco más complicada si se le compara con algunas de las otras interfaces programables que describimos en este capítulo. La programación es un proceso de dos partes que incluye el diálogo de inicialización y el diálogo de operación.

En la computadora personal, que utiliza el 16550 o su equivalente en programación, las direcciones de los puertos de E/S se decodifican desde 3F8H hasta 3FFH para el puerto COM 0 y desde 2F8H hasta 2FFH para el puerto COM 2. Aunque los ejemplos en esta sección del capítulo no están escritos de manera específica para la computadora personal, pueden adaptarse si se modifican los números de puerto para controlar los puertos COM en la PC.

Inicialización del 16550. El diálogo de inicialización, que ocurre después de un reinicio de hardware o de software, consiste de dos partes: la programación del registro de control de línea y del generador de velocidad en Baudios. El registro de control de línea selecciona el número de bits de datos, el número de bits de paro y la paridad (si es par o impar, o si la paridad se envía como un 1 o un 0). El generador de velocidad en Baudios se programa mediante un divisor que determina la velocidad en Baudios de la sección del transmisor.

La figura 11-44 muestra el registro de control de línea. Este registro se programa mediante el envío de información al puerto de E/S 011 (A_2, A_1, A_0). Los dos bits de más a la derecha del registro de control de líneas seleccionan el número de bits de datos transmitidos (5, 6, 7 u 8). El número de bits de paro se selecciona mediante el bit S en el registro de control de línea. Si $S = 0$ se utiliza un bit de paro; si $S = 1$ se utilizan 1.5 bits de paro para cinco bits de datos, y se utilizan dos bits de paro con seis, siete u ocho bits de datos.

Los siguientes tres bits se utilizan en conjunto para enviar una paridad par o impar, para no enviar paridad o para enviar un 1 o un 0 en la posición del bit de paridad. Para enviar una paridad par o impar, el bit ST (**stick**) debe colocarse en el nivel de 0 lógico y el bit de habilitación de paridad debe ser un 1 lógico. El valor del bit de paridad determina entonces si la paridad es par o impar. Para no enviar

FIGURA 11-44
El contenido del registro de control de línea del 16550.

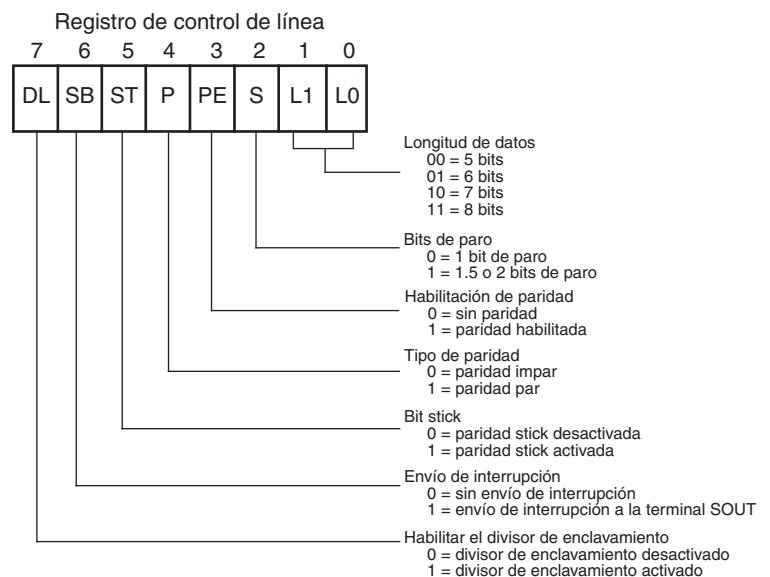


TABLA 11-6 La operación de los bits ST y de paridad.

<i>ST</i>	<i>P</i>	<i>PE</i>	<i>Función</i>
0	0	0	Sin paridad
0	0	1	Paridad impar
0	1	0	Sin paridad
0	1	1	Paridad par
1	0	0	Indefinido
1	0	1	Envía/recibe 1
1	1	0	Indefinido
1	1	1	Envía/recibe 0

paridad (algo común en conexiones de Internet), ST = 0 y el bit de habilitación de paridad también. De esta forma se pueden enviar y recibir datos sin paridad. Por último, si se debe enviar y recibir un 1 o un 0 en la posición del bit de paridad para todos los datos, ST = 1 y el bit de habilitación de paridad también. Para enviar un 1 en la posición del bit de paridad, coloque un 0 en este bit; para enviar un 0, coloque un 1 en el bit de paridad. (En la tabla 11-6 podrá consultar la operación de los bits stick y de paridad.)

El resto de los bits en el registro de control de línea se utilizan para enviar una interrupción y para seleccionar la programación para el divisor de velocidad en Baudios. Si la posición 6 del registro de control de línea es un 1 lógico, se transmite una interrupción. Mientras que este bit sea 1, la interrupción se envía desde la terminal SOUT. Por definición, una interrupción está compuesta de por lo menos dos tramas de datos en forma de 0 lógico. El software en el sistema es responsable de sincronizar la transmisión de la interrupción. Para terminarla, la posición de bit 6 del registro de control de línea se regresa al nivel de 0 lógico. El divisor de velocidad en Baudios sólo podrá programarse cuando la posición 7 del registro de control de línea sea un 1 lógico.

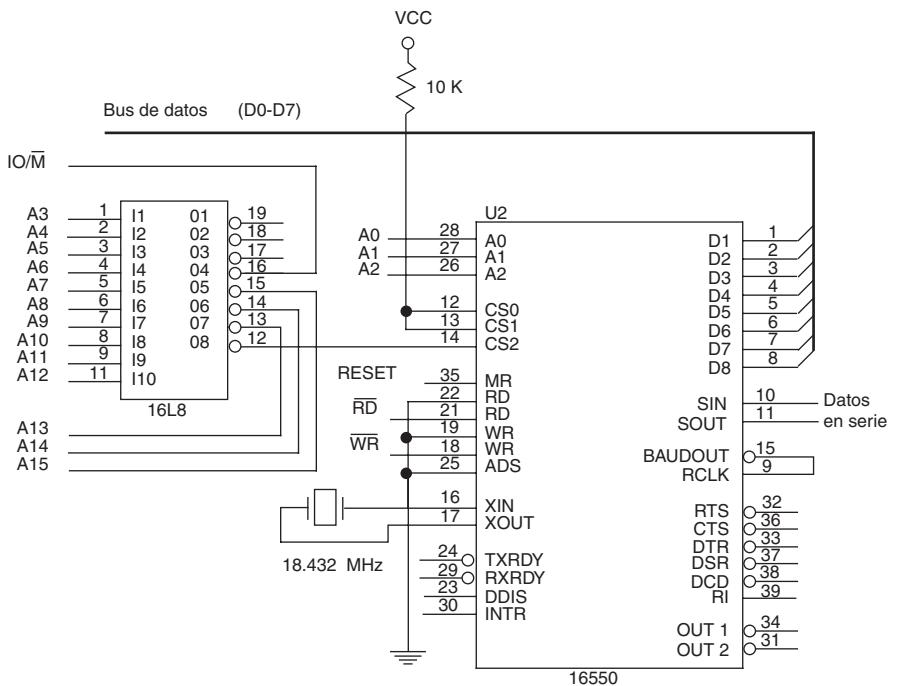
Programación de la velocidad en Baudios. El generador de velocidad en Baudios se programa en las direcciones de E/S 000 y 001 (A_2 , A_1 , A_0). El puerto 000 se utiliza para guardar la parte menos significativa del divisor de 16 bits y el puerto 001 se utiliza para guardar la parte más significativa. El valor que se utilice para el divisor dependerá de la frecuencia del reloj o del cristal externo. Observe en la tabla 11-7 las velocidades en Baudios comunes que pueden obtenerse si se utiliza un cristal de 18.432 MHz como fuente de sincronización. También vea los valores del divisor que se programan en el generador de velocidad en Baudios para obtener estas velocidades. El número que se programa en el generador de velocidad en Baudios hace que produzca un reloj que es 16 veces la velocidad en baudios deseada. Por ejemplo, si se programa un 240 en el divisor de velocidad en Baudios, la velocidad es de $(18.432 \text{ MHz} \div 16) \times 240 = 4800$ Baudios.

Inicialización de muestra. Suponga que un sistema asíncrono requiere siete bits de datos, una paridad impar a una velocidad en Baudios de 9600 y un bit de paro. El ejemplo 11-25 muestra un procedimiento

TABLA 11-7 Velocidades en Baudios comunes para el divisor que se utiliza con el generador de velocidad en Baudios para un cristal de 18.432 MHz.

<i>Velocidad de Baudios</i>	<i>Valor del divisor</i>
110	10,473
300	3840
1200	920
2400	480
4800	240
9600	120
19,200	60
38,400	30
57,600	20
115,200	10

FIGURA 11-45 Interfaz entre el 16550 y el microprocesador 8088 en los puertos 00F0H-00F7H.



que inicializa el 16550 para funcionar de esta manera. En la figura 11-45 puede ver la interfaz para el microprocesador 8088, en la que se utiliza un PLD para decodificar las direcciones de puerto de 8 bits F0H a F7H. (No se muestra el programa para el PLD.) Aquí, el puerto F3H accede al registro de control de línea y los puertos F0H y F1H acceden a los registros del divisor de velocidad en Baudios. Describiremos la última parte del ejemplo 11-25 mediante la función del registro de control FIFO en los siguientes párrafos.

EJEMPLO 11-25

;Diálogo de inicialización para la figura 11-45
;Velocidad en baudios de 9600, 7 bits de datos, paridad impar, 1 bit de paro

```

LINEA EQU 0F3H
LSB   EQU 0F0H
MSB   EQU 0F1H
FIFO  EQU 0F2H

INIC  PROC NEAR

        MOV AL,10001010B      ;habilita el divisor de velocidad en Baudios
        OUT LINEA,AL

        MOV AL,120            ;programa 9600 Baudios
        OUT LSB,AL
        MOV AL,0
        OUT MSB,AL

        MOV AL,00001010B      ;programa 7 bits de datos, paridad
        OUT LINEA,AL          ;impar, 1 bit de paro

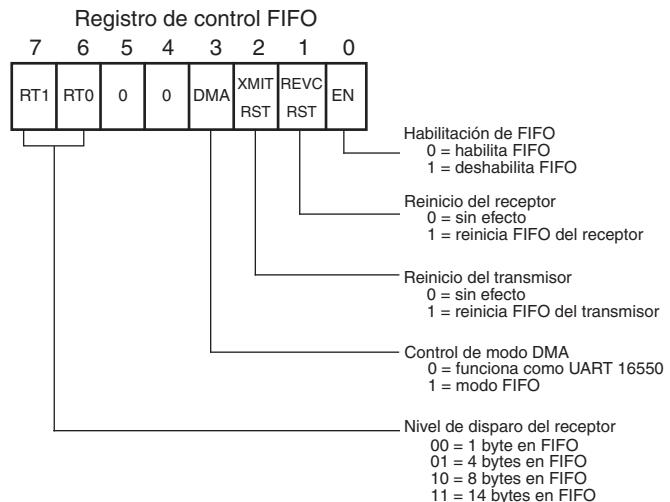
        MOV AL,00000111B      ;habilita el transmisor y
        OUT FIFO,AL           ;el receptor

        RET

INIC  ENDP

```

FIGURA 11-46 El registro de control FIFO del UART 16550.

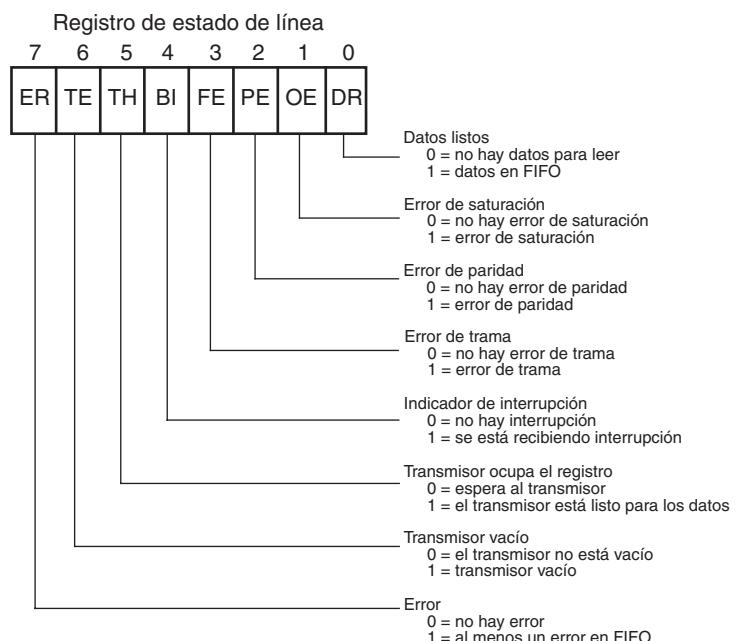


Una vez que se programan el registro de control de línea y el divisor de velocidad en Baudios en el 16550, todavía no está listo para funcionar. Después de programar estos componentes debemos programar el registro de control FIFO, el cual se encuentra en el puerto F2H en el circuito de la figura 11-45.

Observe en la figura 11-46 el registro de control FIFO para el 16550. Este registro habilita el transmisor y el receptor (bit 0 = 1) y borra las memorias FIFO del transmisor y del receptor. También proporciona el control para las interrupciones del 16550, que se describen en el capítulo 12. Observe que en la última sección del ejemplo 11-25 se coloca un 7 en el registro de control FIFO. Con esto se habilitan el transmisor y el receptor, y se borran ambas FIFOs. Ahora el 16550 está listo para operar, pero sin interrupciones. Éstas se deshabilitan de manera automática cuando la entrada MR (reinicio maestro) se coloca en 1 lógico mediante la señal RESET del sistema.

Envío de datos en serie. Antes de que puedan enviarse o recibirse datos en serie a través del 16550, necesitamos conocer la función del registro de estado de línea (vea la figura 11-47). Este registro contiene

FIGURA 11-47
El contenido del registro de estado de línea del UART 16550.



información sobre las condiciones de error y sobre el estado del transmisor y del receptor. Este registro se evalúa antes de que pueda transmitirse o recibirse un byte.

Suponga que se escribe un procedimiento (vea el ejemplo 11-26) para transmitir el contenido de AH al 16550 y enviarlo a través de su terminal de datos en serie (SOUT). El bit TH se sondea mediante software para determinar si el transmisor está listo para recibir datos. Este procedimiento utiliza el circuito de la figura 11-45.

EJEMPLO 11-26

```
;Un procedimiento que transmite AH mediante el UART 16550

ESTADL EQU 0F5H
DATOS EQU 0FOH

ENVIA PROC NEAR USES AX

    .REPEAT           ;evalúa el bit TH
        IN AL,ESTADL
        TEST AL,20H
    .UNTIL !ZERO?

        MOV AL,AH           ;envía los datos
        OUT DATOS,AL
        RET

ENVIA ENDP
```

Recepción de datos en serie. Para leer la información que se recibe del 16550, se evalúa el bit DR del registro de estado de línea. El ejemplo 11-27 muestra un procedimiento que evalúa el bit DR para decidir si el 16550 ha recibido datos o no. Al momento de recibir los datos, el procedimiento comprueba si hay errores. Si se detecta un error, el procedimiento regresa con AL igual a un ‘?’ ASCII. Si no se produce un error, entonces el procedimiento regresa con AL igual al carácter recibido.

EJEMPLO 11-27

```
;Un procedimiento que recibe datos del UART 16550 y
;los regresa en AL.
```

```
ESTADL EQU 0F5H
DATOS EQU 0FOH

RECIBE PROC NEAR

    .REPEAT           ;evalúa el bit DR
        IN AL,ESTADL
        TEST AL,1
    .UNTIL !ZERO?

        TEST AL,0EH           ;comprueba si hay error
        .IF ZERO?             ;no hay error
            IN AL,DATOS
        .ELSE                  ;cualquier error
            MOV AL,'?'
        .ENDIF
        RET

RECIBE ENDP
```

Errores del UART. Los tipos de errores que detecta el 16550 son: error de paridad, error de trama y error de saturación. Un **error de paridad** indica que los datos que se recibieron contienen la paridad incorrecta. Un **error de trama** indica que los bits de inicio y de paro no están en sus posiciones correctas. Un **error de saturación** indica que los datos han saturado el búfer FIFO del receptor interno.

Estos errores no deben ocurrir durante la operación normal. Si se produce un error de paridad, indica que se encontró ruido durante la recepción. Un error de trama se produce cuando el receptor recibe datos a una velocidad en Baudios incorrecta. Un error de saturación ocurre sólo si el software no puede leer los datos del UART antes de que la memoria FIFO del receptor esté llena. Este ejemplo no evalúa el bit BI (bit indicador de interrupción) para una condición de interrupción. Tenga en cuenta que una interrupción está compuesta de dos tramas consecutivas de 0s lógicos en la terminal SIN del UART. En el capítulo 12 veremos el resto de los registros, que se utilizan para el control de interrupciones y del módem.

11-6

CONVERTIDORES DE ANALÓGICO A DIGITAL (ADC) Y DE DIGITAL A ANALÓGICO (DAC)

Estos convertidores se utilizan como interfaz entre el microprocesador y el mundo analógico. Muchos eventos que el microprocesador supervisa y controla son eventos analógico. Éstos pueden variar desde el monitoreo de todo tipo de eventos, incluyendo el habla, hasta el control de motores y dispositivos similares. Para poder conectar el microprocesador con estos eventos, debemos tener una comprensión de la interfaz y el control de los convertidores ADC y DAC, que realizan conversiones entre datos analógicos y digitales.

El convertidor DAC0830 de analógico a digital

El DAC0830 es uno de los convertidores de analógico a digital bastante comunes y de bajo costo (un producto de National Semiconductor Corporation). Este dispositivo es un convertidor de 8 bits que transforma un número binario de 8 bits en un voltaje analógico. Hay otros convertidores disponibles que transforman los números binarios de 10, 12 o 16 bits en voltajes analógicos. El número de escalones de voltaje que genera el convertidor es igual al número de combinaciones de entrada binarias. Por lo tanto, un convertidor de 8 bits genera 256 niveles de voltaje distintos, un convertidor de 10 bits genera 1024 niveles, y así sucesivamente. El DAC0830 es un convertidor de mediana velocidad que transforma una entrada digital en una salida analógica, en un tiempo aproximado de 1.0 μ s.

Observe en la figura 11-48 el diagrama de terminales del DAC0830. Este dispositivo tiene un conjunto de ocho conexiones de bus de datos para la aplicación del código de entrada digital, y un par de salidas analógicas etiquetadas como IOUT1 e IOUT2, que están diseñadas como entradas para un amplificador operacional externo. Como éste es un convertidor de 8 bits, su voltaje de escalón de salida se define como $-V_{REF}$ (voltaje de referencia) dividido entre 256. Por ejemplo, si el voltaje de referencia es de -5.0 V, su voltaje de escalón de salida es de +0.196 V. Observe que el voltaje de salida tiene la polaridad opuesta al voltaje de referencia. Si se aplica una entrada de 1001 0010₂ al dispositivo, el voltaje de salida será el voltaje de escalón por 1001 0010₂ o, en este caso, +2.862 V. Si se cambia el voltaje de referencia a -5.1 V, el voltaje de escalón se vuelve +.02 V. Con frecuencia, a este voltaje de escalón también se le conoce como la **resolución** del convertidor.

FIGURA 11-48 Diagrama de terminales del convertidor DAC9830 de digital a analógico.

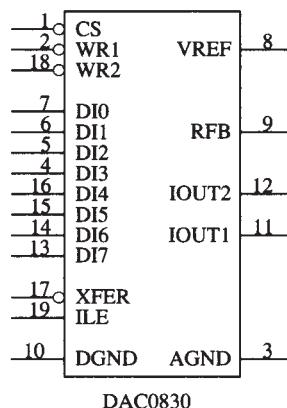
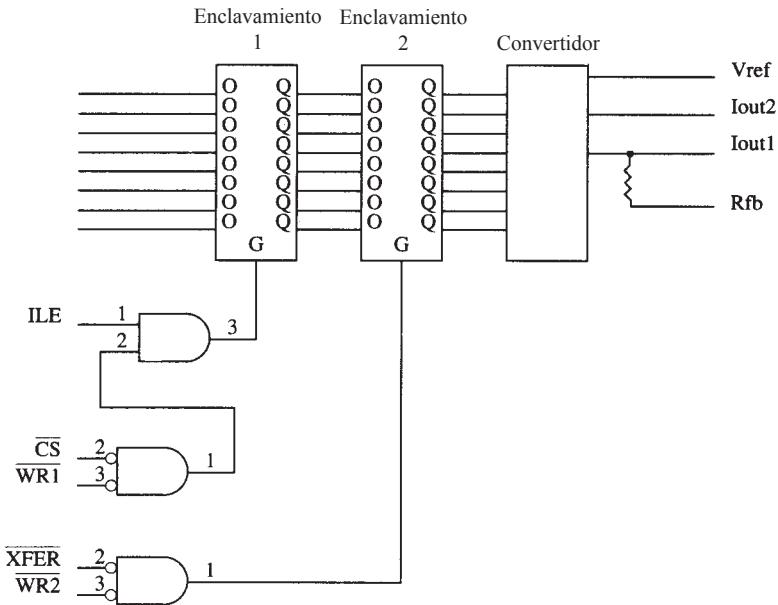


FIGURA 11-49 Estructura interna del DAC0830.



Estructura interna del DAC0830. La figura 11-49 muestra la estructura interna del DAC0830. Observe que este dispositivo contiene dos registros internos. El primero es un registro de almacenamiento y el segundo se conecta al convertidor de escalera interno R-2R. Los dos enclavamientos permiten almacenar un byte mientras se convierte otro. En muchos casos se deshabilita el primer enclavamiento y sólo se utiliza el segundo para introducir datos en el convertidor. Esto se logra mediante la conexión de un 1 lógico a ILE y de un 0 lógico a CS (selección de chip).

Ambos enclavamientos dentro del DAC0830 son transparentes. Es decir, cuando la entrada G que va al enclavamiento es un 1 lógico, los datos pasan a través de éste, pero cuando la entrada G se vuelve un 0 lógico los datos se enclavan o almacenan. El convertidor tiene una terminal de entrada de referencia (V_{REF}) que establece el voltaje de salida de escala completa. Si se colocan -10 V en V_{REF} , el voltaje de salida de escala completa (1111111_2) es de +10 V. La salida de la escalera R-2R dentro del convertidor aparece en IOUT1 e IOUT2. Estas salidas están diseñadas para aplicarse a un amplificador operacional, tal como un 741 o un dispositivo similar.

Conexión del DAC0830 al Microprocesador. En la figura 11-50 puede ver la conexión del DAC0830 al microprocesador. Aquí se utiliza un PLD para decodificar el DAC0830 en la dirección de puerto de E/S 20H de 8 bits. Cada vez que se ejecuta una instrucción OUT 20H, AL se pasa el contenido de las conexiones AD₀-AD₇ del bus de datos al convertidor dentro del DAC0830. El amplificador operacional 741, junto con el voltaje de referencia zener -12 V, hace que el voltaje de salida de escala completa sea igual a +12 V. La salida del amplificador operacional alimenta a un controlador que opera un motor de 12 V CD. Este controlador es un amplificador Darlington para motores grandes. En este ejemplo se muestra cómo el convertidor controla un motor, pero pueden usarse también otros dispositivos como salidas.

El convertidor ADC080X de análogo a digital

El ADC080X es un ADC común de bajo costo, el cual pertenece a una familia de convertidores que son todos idénticos, excepto en precisión. Este dispositivo es compatible con un amplio rango de microprocesadores tales como la familia Intel. Aunque hay ADCs más veloces en el mercado y algunos de ellos tienen resoluciones mayores de ocho bits, este dispositivo es ideal para muchas aplicaciones que no requieren de un alto grado de precisión. El ADC080X requiere hasta 100 μ s para convertir un voltaje de entrada analógico en un código de salida digital.

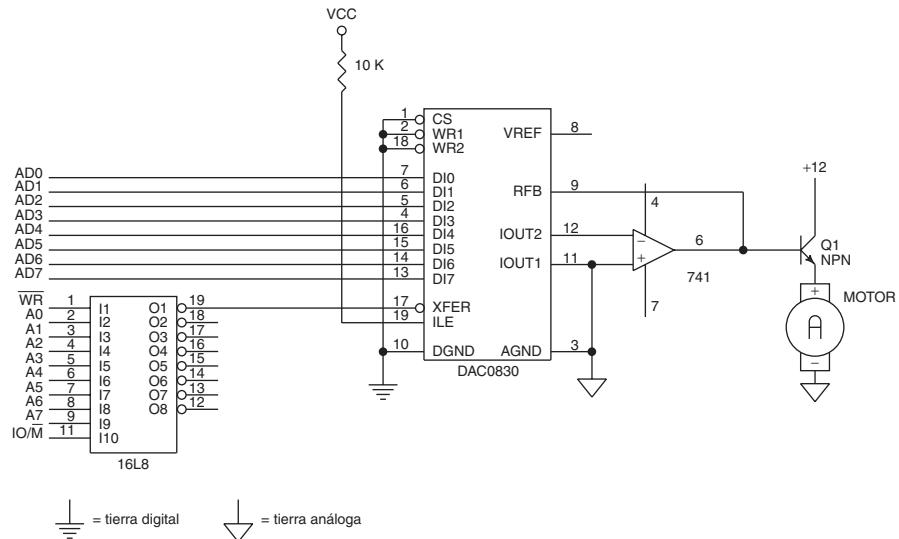


FIGURA 11-50 Interfaz entre el DAC0830 y el microprocesador 8086 en la posición de E/S de 8 bits 20H.

La figura 11-51 muestra el diagrama de terminales del convertidor ADC0804 (un producto de National Semiconductor Corporation). Para operar este convertidor, se aplica un pulso a la terminal WR y se aterriza CS para iniciar el proceso de conversión. Como este convertidor requiere una cantidad considerable de tiempo para la conversión, una terminal etiquetada como INTR indica el término de este proceso. En la figura 11-52 podrá consultar un diagrama de sincronización que muestra la interacción

FIGURA 11-51 Diagrama de terminales del convertidor ADC0804 de análogo a digital.

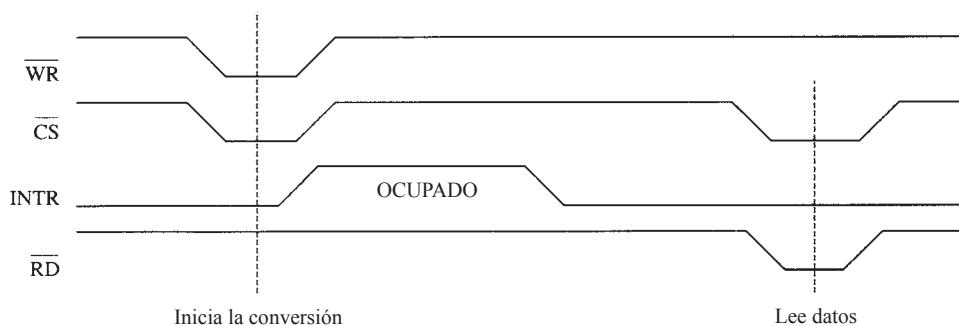
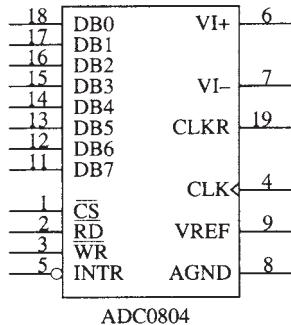
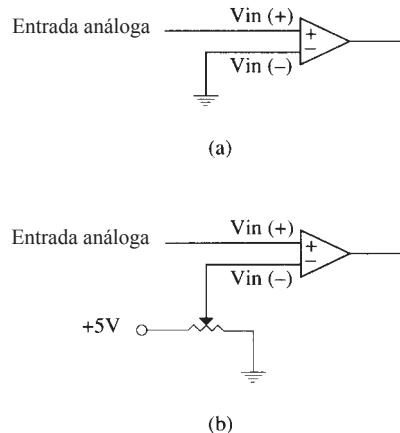


FIGURA 11-52 Sincronización para el convertidor ADC0804 de análogo a digital.

FIGURA 11-53 Las entradas analógicas para el convertidor ADC0804. (a) Para detectar una entrada de 0 a +5.0 V. (b) para detectar un desplazamiento de la entrada desde tierra.



de las señales de control. Como puede ver, iniciamos el convertidor con el pulso \overline{WR} , esperamos a que INTR regrese al nivel de 0 lógico y después leemos los datos del convertidor. Si se utiliza un retraso de tiempo que permita cuando menos 100 μ s, entonces no necesitamos evaluar la terminal INTR. Otra opción sería conectar la terminal INTR a una entrada de interrupción, de manera que cuando la conversión termine se produzca una interrupción.

La señal de entrada analógica. Antes de poder conectar el ADC0804 al microprocesador, debemos comprender el funcionamiento de sus entradas analógicas. Hay dos entradas analógicas en el ADC0804: VIN(+) y VIN(-). Estas entradas se conectan a un amplificador operacional interno y son entradas diferenciales, como se observa en la figura 11-53. El amplificador operacional suma las entradas diferenciales para producir una señal para el convertidor de analógico a digital interno. La figura 11-53 muestra unas cuantas formas de utilizar estas entradas diferenciales. La primera forma (vea la figura 11-53a) utiliza una sola entrada que puede variar entre 0 y +5.0 V. La segunda forma (vea la figura 11-53b) muestra un voltaje variable que se aplica a la terminal VIN(-), por lo que puede ajustarse la referencia cero para VIN(+).

Generación de la señal de reloj. El ADC0804 requiere una fuente que suministre el reloj para que pueda operar. Éste puede ser un reloj externo que se aplique a la terminal CLK IN, o puede generarse mediante un circuito RC. El rango permitido de frecuencias de reloj está entre 100 KHz y 1460 KHz. Es conveniente usar una frecuencia que esté lo más cerca posible de 1460 KHz, de manera que el tiempo de conversión se minimice lo más posible.

Si el reloj se genera mediante un circuito RC, utilizamos las terminales CLK IN y CLK R que se conectan a un circuito RC, como puede ver en la figura 11-54. Cuando se utiliza esta conexión, la frecuencia de reloj se calcula mediante la siguiente ecuación:

$$F_{clk} = \frac{1}{1.1 RC}$$

FIGURA 11-54 Conexión del circuito RC a las terminales CLK IN y CLK R del ADC0804.

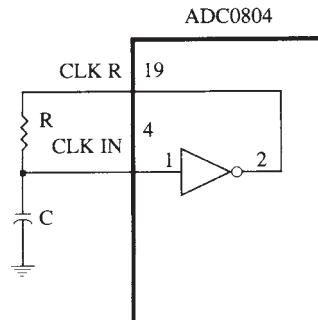
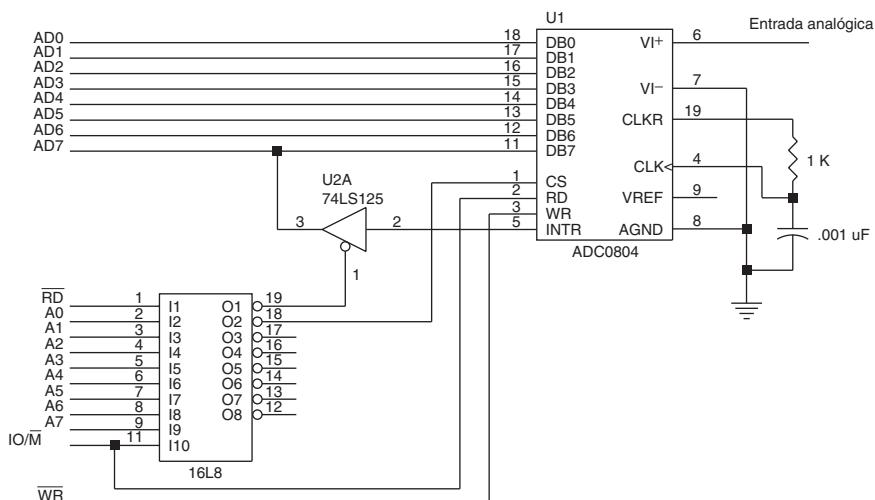


FIGURA 11-55 Interfaz entre el ADC0804 y el microprocesador.



Conexión del ADC0804 al microprocesador. Observe en la figura 11-55 la interfaz entre el ADC0804 y el microprocesador 8086. La señal V_{REF} no está conectada, lo cual es normal. Suponga que el ADC0804 se decodifica en la dirección de puerto de E/S de 8 bits 40H para los datos y en la dirección de puerto 42H para la señal INTR, y que se requiere un procedimiento para iniciar y leer datos del ADC. Este procedimiento se muestra en el ejemplo 11-28. El bit INTR se sondea y si se vuelve un 0 lógico el procedimiento termina, y AL contiene el código digital convertido.

EJEMPLO 11-28

```

ADC      PROC    NEAR
        OUT 40H,AL
        .REPEAT          ;evalúa INTR
                IN AL,42H
                TEST AL,80H
                .UNTIL ZERO?
                IN AL,40H
                RET
ADC      ENDP

```

Uso del ADC0804 y del DAC0830

En esta sección ilustraremos un ejemplo que utiliza el ADC0804 y el DAC0830 para capturar y reproducir señales de audio o de voz. En el pasado se utilizaba con frecuencia un sintetizador de voz para generar el habla, pero la calidad era pobre. Para una voz con calidad humana podemos usar el ADC0804 para capturar una señal de audio y almacenarla en memoria para reproducirla después mediante el uso del DAC0830.

La figura 11-56 muestra el circuito requerido para conectar el ADC0804 en los puertos de E/S 0700H y 0702H. El DAC0830 se conecta en el puerto de E/S 704H. Estos puertos de E/S se encuentran en el banco inferior de un microprocesador de 16 bits tal como el 8086 o el 80386SX. En el ejemplo 11-29 puede ver el software que se utiliza para operar estos convertidores. Este software lee una ráfaga de un segundo de voz luego lo reproduce 10 veces. Uno de los procedimientos, llamado LEEV, lee la voz y el otro, llamado REPRV, la reproduce. La voz se muestrea y se almacena en una sección de memoria llamada PALABRAV. La velocidad de muestreo se ajusta a 2048 muestras por segundo, la cual produce un sonido de voz aceptable.

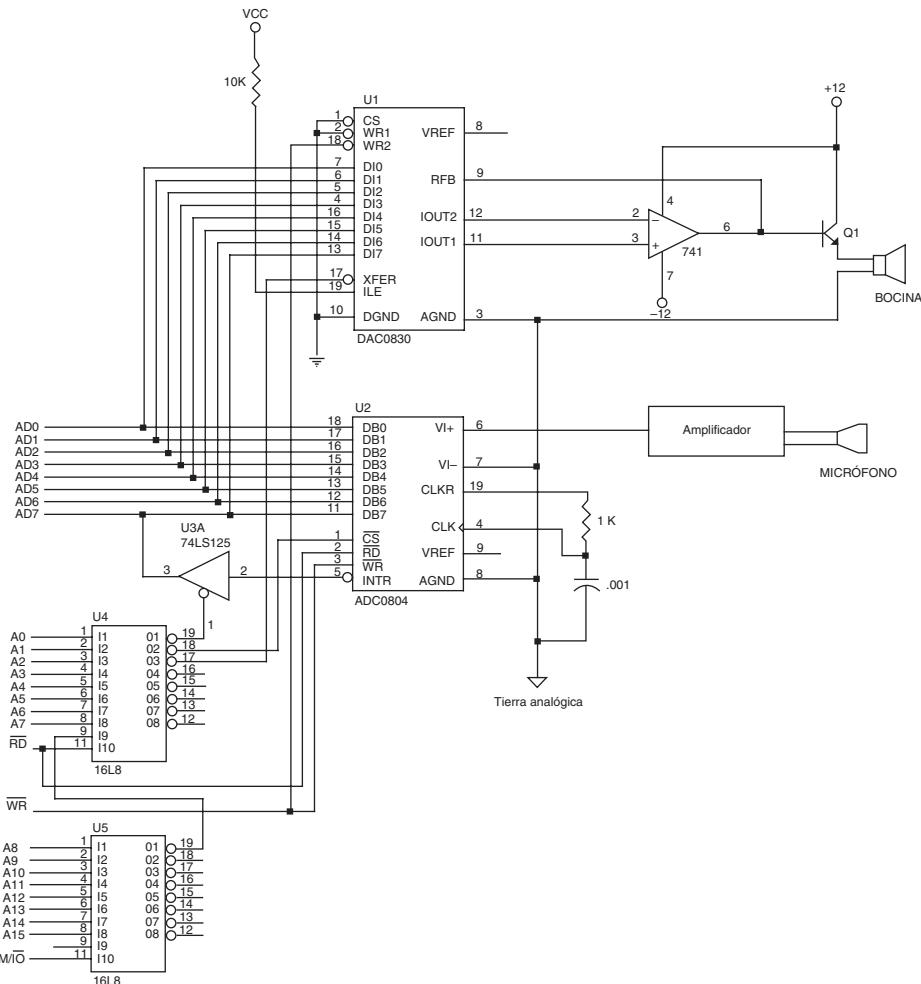


FIGURA 11-56 Un circuito que almacena voz y la reproduce a través de la bocina.

EJEMPLO 11-29

;Software que graba un segundo de voz y la reproduce
;10 veces.

;Se supone que la frecuencia de reloj es de 20 MHz en un microprocesador 80386SX

```

LEEV      PROC    NEAR USES ECX DX
        MOV ECX,2048          ;cuenta = 2048
        MOV DX,700H            ;direcciona el puerto 700H
        .REPEAT
        OUT DX,AL              ;inicia la conversión
        ADD DX,2                ;direcciona el puerto de estado
        .REPEAT
        IN AL,DX                ;espera al convertidor
        TEST AL,80H
        .UNTIL ZERO?
        SUB DX,2                ;direcciona el puerto de datos
        IN AL,DX                ;obtiene los datos
        MOV PALABRAS[ECX-1]
        CALL RETRASO             ;espera 1/2048 seg.

```

```

.UNTILCXZ
RET

LEEV    ENDP

REPRV   PROC  NEAR  USES DX ECX
        MOV   ECX,2048          ;cuenta = 2048
        MOV   DX,704H           ;direcciona el DAC
        .REPEAT
        MOV   AL,PALABRAS[EAX-1]
        OUT  DX,AL              ;envía byte al DAC
        CALL  RETRASO            ;espera 1/2048 seg.
        .UNTILCXZ
        RET
REPRV   ENDP

RETRASO  PROC  NEAR  USES CX
        MOV   CX,888             ;desperdicia 1/2048 seg.
        .REPEAT
        .UNTILCXZ
        RET
RETRASO  ENDP

```

11-7**RESUMEN**

1. Los microprocesadores del 8086 al Pentium 4 tienen dos tipos básicos de instrucciones de E/S: IN y OUT. La instrucción IN introduce datos desde un dispositivo de E/S externo hacia el registro AL (8 bit) o AX (16 bits). La instrucción IN está disponible como instrucción de puerto fijo, como instrucción de puerto variable o como una instrucción de cadena (80286-Pentium 4) INSB o INSW. La instrucción OUT envía datos desde AL o AX hacia un dispositivo de E/S externo y está disponible como instrucción fija, variable o instrucción de cadena OUTSB o OUTSW. La instrucción de puerto fijo utiliza una dirección de puerto de E/S de 8 bits, mientras que las instrucciones de E/S variable y de cadena utilizan un número de puerto de 16 bits, el cual se encuentra en el registro DX.
2. La E/S aislada, algunas veces conocida como E/S directa, utiliza un mapa separado para el espacio de E/S, liberando toda la memoria para que el programa pueda utilizarla. La E/S aislada utiliza las instrucciones IN y OUT para transferir datos entre el dispositivo de E/S y el microprocesador. La estructura de control del mapa de E/S aislada utiliza IORC (control de lectura de E/S e IOWC (control de escritura de E/S) junto con las señales de selección de banco BHE y BLE (A0 en el 8086 y 80286) para efectuar la transferencia de E/S. Los microprocesadores 8086/8088 utilizaban la señal M/IO (IO/M) con RD y WR para generar las señales de control de E/S.
3. La E/S por asignación de memoria utiliza una porción del espacio de memoria para las transferencias de E/S. Esto reduce la cantidad de memoria disponible, pero evita la necesidad de utilizar las señales IORC e IOWC para las transferencias de E/S. Además, cualquier instrucción que dirccione una posición de memoria y que utilice cualquier modo de direccionamiento podrá utilizarse para transferir datos entre el microprocesador y el dispositivo de E/S mediante el uso de la E/S por asignación de memoria.
4. Todos los dispositivos de entrada usan búferes, de manera que los datos de E/S se conecten sólo al bus de datos durante la ejecución de la instrucción IN. El búfer se construye integrado a un periférico programable o se ubica por separado.
5. Todos los dispositivos de salida utilizan un enclavamiento para capturar los datos de salida durante la ejecución de la instrucción OUT. Esto es necesario, ya que los datos aparecen en el bus de datos durante menos de 100 ns para una instrucción OUT, y la mayoría de los dispositivos de salida requieren los datos durante un tiempo mayor. En muchos casos, el enclavamiento se construye integrado al periférico.

6. El protocolo de intercambio (*handshaking*) o sondeo es el acto de sincronizar dos dispositivos independientes con unas cuantas líneas de control. Por ejemplo, la computadora pregunta a una impresora si está ocupada al recibir como entrada la señal BUSY proveniente de la impresora. Si no está ocupada, la computadora envía datos a la impresora y le informa que los datos están disponibles mediante una señal de estrobo de datos (DS). A esta comunicación entre la computadora y la impresora se le conoce como protocolo de intercambio, o sondeo.
7. Las interfaces se requieren para la mayoría de los dispositivos de entrada basados en interruptores y para la mayoría de los dispositivos de salida que no son compatibles con TTL.
8. El número de puerto de E/S aparece en las conexiones A₇-A₀ del bus de direcciones para una instrucción de puerto de E/S fijo, y en las conexiones A₁₅-A₀ para una instrucción de puerto de E/S variable (considere que A₁₅-A₈ contienen ceros para un puerto de 8 bits). En ambos casos, los bits de dirección por encima del A₁₅ no están definidos.
9. Como los microprocesadores 8086/80286/80386SX contienen un bus de datos de 16 bits y las direcciones de E/S hacen referencia a posiciones de E/S tipo byte, el espacio de E/S también se organiza en bancos, de manera similar al sistema de memoria. Para poder conectar un dispositivo de E/S de 8 bits al bus de datos de 16 bits, a menudo se requieren estrobo de escritura separados (uno superior y uno inferior) para las operaciones de escritura de E/S. De igual forma, los microprocesadores 80486 y del Pentium al Pentium 4 también organizan la E/S en bancos.
10. El decodificador de puertos de E/S es muy parecido al decodificador de direcciones de memoria, sólo que en vez de decodificar toda la dirección, el decodificador de puertos de E/S sólo decodifica una dirección de 16 bits para las instrucciones con puerto variable y a menudo decodifica un número de puerto de 8 bits para las instrucciones con E/S fija.
11. El 82C55 es una interfaz periférica programable (PIA) con 24 terminales de E/S que se programan en dos grupos de 12 terminales cada uno (grupo A y grupo B). El 82C55 opera en tres modos: E/S simple (modo 0), E/S estroboscópica (modo 1) y E/S bidireccional (modo 2). Cuando el 82C55 se conecta al 8086 que opera a 8 MHz, insertamos dos estados de espera ya que la velocidad del microprocesador es mucho más de lo que el 82C55 puede manejar.
12. El dispositivo de pantalla LCD requiere una buena cantidad de software, pero muestra información en código ASCII.
13. El 8254 es un temporizador de intervalos programables que contiene tres contadores de 16 bits que cuentan en binario o en decimal codificado en binario (BCD). Cada contador es independiente y opera en seis modos distintos: (1) contador de eventos, (2) multivibrador monoestable redispersable, (3) generador de pulsos, (4) generador de onda cuadrada, (5) generador de pulso activado por software y (6) generador de pulso activado por hardware.
14. El 16550 es una interfaz de comunicaciones programable, capaz de recibir y transmitir datos asíncronos en serie.
15. El DAC0830 es un convertidor digital-analógico de 8 bits, el cual convierte una señal digital en un voltaje analógico dentro de un rango no mayor a 1.0 μ s.
16. El ADC0804 es un convertidor analógico-digital de 8 bits, el cual transforma una señal analógica en una señal digital dentro de un rango no mayor a 100 μ s.

11-8

PREGUNTAS Y PROBLEMAS

1. Explique en qué sentido fluyen los datos para unas instrucciones IN y OUT.
2. ¿En dónde se almacena el número de puerto de E/S para una instrucción de E/S fija?
3. ¿En dónde se almacena el número de puerto de E/S para una instrucción de E/S variable?
4. ¿En dónde se almacena el número de puerto de E/S para una instrucción de E/S de cadena?
5. ¿En cuál registro se introducen datos mediante la instrucción IN de 16 bits?
6. Describa la operación de la instrucción OUTSB.
7. Describa la operación de la instrucción INSW.
8. Compare un sistema de E/S por asignación de memoria con un sistema de E/S aislada.
9. ¿Cuál es la interfaz básica de entrada?
10. ¿Cuál es la interfaz básica de salida?

11. Explique el término *protocolo de intercambio (handshaking)* en cuanto a su aplicación en los sistemas computacionales de E/S.
12. En el microprocesador 8086, la dirección de un puerto de E/S con numeración par se encuentra en el banco de E/S _____.
13. En el Pentium 4, ¿cuál banco contiene el número de puerto de E/S 000AH?
14. ¿Cuántos bancos de E/S tiene el microprocesador Pentium 4?
15. Muestre el circuito que genera los estrobos de escritura de E/S superior e inferior.
16. ¿Cuál es el propósito de un eliminador de rebotes por contacto?
17. Desarrolle una interfaz para controlar un relevador de manera apropiada. El relevador es de 12 V y requiere una corriente de bobina de 150 mA.
18. Desarrolle un controlador para bobina de relevador que pueda controlar un relevador de 5.0 V y que requiera 60 mA de corriente de bobina.
19. Desarrolle un decodificador de puertos de E/S mediante el uso de un 74ALS138, que genere estrobos de E/S en el banco inferior, para las siguientes direcciones de puerto de E/S de 8 bits de un microprocesador de 16 bits: 10H, 12H, 14H, 16H, 18H, 1AH, 1CH y 1EH.
20. Desarrolle un decodificador de puertos de E/S mediante el uso de un 74ALS138, que genere estrobos de E/S en el banco superior, para las siguientes direcciones de puerto de E/S de 8 bits de un microprocesador de 16 bits: 11H, 13H, 15H, 17H, 19H, 1BH, 1DH y 1FH.
21. Desarrolle un decodificador de puertos de E/S mediante un PLD, que genere estrobos de E/S de 16 bits para las siguientes direcciones de puerto de E/S de 16 bits: 1000H-1001H, 1002H-1003H, 1004H-1005H, 1006H-1007H, 1008H-1009H, 100AH-100BH, 100CH-100DH y 100EH-100FH.
22. Desarrolle un decodificador de puertos de E/S mediante un PLD, que genere los siguientes estrobos de E/S de banco inferior: 00A8H, 00B6H y 00EEH.
23. Desarrolle un decodificador de puertos de E/S mediante un PLD, que genere los siguientes estrobos de E/S de banco superior: 300DH, 300BH, 1005H y 1007H.
24. ¿Por qué se ignoran las señales \overline{BHE} y \overline{BLE} (A_0) en un decodificador de direcciones de puertos de 16 bits?
25. ¿A qué conexiones de bus de un Pentium 4 se conecta un dispositivo de E/S de 8 bits, que se encuentra en la dirección de puerto de E/S 0010H?
26. ¿A qué conexiones de bus de un Pentium 4 se conecta un dispositivo de E/S de 8 bits, que se encuentra en la dirección de puerto de E/S 100DH?
27. ¿Cuántas conexiones de terminales de E/S programables tiene un 82C55?
28. Liste las terminales que pertenecen al grupo A y al grupo B en el 82C55.
29. ¿Cuáles son las dos terminales del 82C55 que se encargan de la selección de la dirección del puerto de E/S interno?
30. ¿A cuál conexión del bus de control del sistema 8086 se conecta la terminal \overline{RD} del 82C55?
31. Use un PLD para crear una interfaz entre un 82C55 y el microprocesador 8086, de manera que funcione en las posiciones de E/S 0380H, 0382H, 0384H y 0386H.
32. Cuando se reinicia el 82C55, sus puertos de E/S se inicializan como _____.
33. ¿Cuáles son los tres modos de operación disponibles para el 82C55?
34. ¿Cuál es el propósito de la señal STB en la operación de entrada estroboscópica del 82C55?
35. Desarrolle un procedimiento de retraso de tiempo para que un Pentium 4 de 2.0 GHz espere durante 80 μ s.
36. Desarrolle un procedimiento de retraso de tiempo para que un Pentium 4 de 3.0 GHz espere durante 12 ms.
37. Explique la operación de un motor simple de velocidad gradual con cuatro bobinas.
38. ¿Qué es lo que activa la terminal IBF en la operación de entrada estroboscópica del 82C55?
39. Escriba el software requerido para colocar un 1 lógico en la terminal PC7 del 82C55 durante la operación de entrada estroboscópica.
40. ¿Cómo se habilita la terminal de petición de interrupción (INTR) en el modo de operación de entrada estroboscópica del 82C55?
41. En la operación de salida estroboscópica del 82C55, ¿cuál es el propósito de la señal \overline{ACK} ?
42. ¿Qué es lo que borra la señal \overline{OBF} en la operación de salida estroboscópica del 82C55?
43. Escriba el software requerido para decidir si PC4 es un 1 lógico cuando el 82C55 se opera en el modo de salida estroboscópica.

44. ¿Qué grupo de terminales se utiliza durante la operación bidireccional del 82C55?
45. ¿Qué terminales son de E/S de propósito general durante la operación en modo 2 del 82C55?
46. Describa cómo se borra la pantalla LCD.
47. ¿Cómo se selecciona una posición en la pantalla LCD?
48. Escriba un procedimiento corto que coloque una cadena ASCII nula en la posición 6 de la pantalla LCD.
49. ¿Cómo se evalúa la bandera de ocupado en la pantalla LCD?
50. ¿Qué cambios deben realizarse a la figura 11-25 para que funcione con una matriz de teclado que contenga tres filas y cinco columnas?
51. ¿Cuánto tiempo se utiliza por lo general para eliminar los rebotes en un teclado?
52. Desarrolle la interfaz para un teclado estilo telefónico, de tres por cuatro teclas. Necesitará utilizar una tabla de búsqueda para convertir los datos al código de tecla correcto.
53. El temporizador de intervalos 8254 funciona mediante CD, a _____ Hz.
54. ¿En cuántos modos distintos funciona cada contador en el 8254?
55. Conecte un 8254 para que funcione en las direcciones de puertos de E/S XX10H, XX12H, XX14H y XX16H.
56. Escriba el software que programe el contador 2 para generar una onda cuadrada de 80 KHz, si la entrada CLK que va al contador 2 es de 8 MHz.
57. ¿Qué número se programa en un contador 8254 para contar 300 eventos?
58. Si se programa una cuenta de 16 bits en el 8254, ¿cuál byte de la cuenta se programa primero?
59. Explique la función de la palabra de control de repetición de lectura en el 8254.
60. Programe el contador 1 del 8254 de manera que genere una serie de pulsos continuos con un tiempo en nivel alto de 100 μ s y un tiempo en nivel bajo de 1 μ s. Asegúrese de indicar la frecuencia de CLK requerida para esta tarea.
61. ¿Por qué un ciclo de trabajo del 50% hace que el motor permanezca inerte en el circuito de control de velocidad y dirección de motores que presentamos en este capítulo?
62. ¿Qué son los datos asíncronos en serie?
63. ¿Qué es la velocidad en Baudios?
64. Programe el 16550 para que opere con seis bits de datos, paridad par, un bit de paro y una velocidad en Baudios de 19,200 mediante el uso de un reloj de 18.432 MHz. (Suponga que los puertos de E/S se enumeran como 20H y 22H.)
65. Si el 16550 debe generar una señal en serie a una velocidad de 2400 Baudios y el divisor de velocidad en Baudios se programa con 16, ¿cuál es la frecuencia de la señal?
66. Describa los siguientes términos: *sinplex*, *half-duplex* y *full-duplex*.
67. ¿Cómo se reinicia el 16550?
68. Escriba un procedimiento para que el 16550 transmita 16 bytes desde un pequeño búfer en la dirección del segmento de datos (DS se carga en forma externa) mediante SI (SI se carga en forma externa).
69. El DAC0830 convierte una entrada digital de 8 bits en una salida digital en un tiempo aproximado de _____.
70. ¿Cuál es el voltaje de escalón en la salida del DAC0830, si el voltaje de referencia es de -2.55 V?
71. Conecte un DAC0830 al 8086, de manera que opere en el puerto de E/S 400H.
72. Desarrolle un programa para la interfaz de la pregunta 71, de manera que el DAC0830 genere una forma de onda de voltaje triangular. La frecuencia de esta forma de onda debe ser un valor aproximado a 100 Hz.
73. El ADC080X requiere un tiempo aproximado de _____ para convertir un voltaje analógico en código digital.
74. ¿Para qué se utiliza la terminal \overline{WR} en el ADC080X?
75. Conecte un ADC080X al puerto de E/S 0260H para los datos y al puerto 0207H para evaluar la terminal INTR.
76. Desarrolle un programa para el ADC080X de la pregunta 75, de manera que lea un voltaje de entrada una vez cada 100 ms y que almacene los resultados en un arreglo de memoria con una longitud de 100H bytes.
77. Vuelva a escribir el ejemplo 11-29 mediante el uso de C++ con código ensamblador en línea.

CAPÍTULO 12

Interrupciones

INTRODUCCIÓN

En este capítulo extenderemos la cobertura de la E/S básica y las interfaces periféricas programables, mediante el análisis de una técnica llamada E/S procesada por interrupciones. Una interrupción es un procedimiento iniciado por hardware que interrumpe cualquier programa que esté ejecutándose. En este capítulo proporcionaremos ejemplos y una explicación detallada de la estructura de interrupciones de toda la familia de microprocesadores Intel.

OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

1. Explicar la estructura de interrupciones de la familia de microprocesadores Intel.
2. Explicar la operación de las instrucciones de interrupción de software INT, INTO, INT 3 y BOUND.
3. Explicar cómo el bit de bandera habilita la interrupción (IF) para modificar la estructura de interrupciones.
4. Describir la función del bit de bandera para atrapar interrupciones (TF) y la operación del rastreo generado por atrapamiento.
5. Desarrollar procedimientos de servicio de interrupciones que controlen dispositivos periféricos externos de baja velocidad.
6. Expandir la estructura de interrupciones del microprocesador mediante el uso del controlador de interrupciones programable 8259A y otras técnicas.
7. Explicar el propósito y la operación de un reloj en tiempo real.

12-1

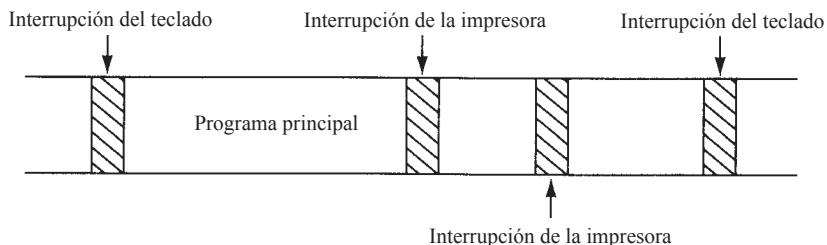
PROCESAMIENTO BÁSICO DE INTERRUPCIONES

En esta sección hablaremos sobre la función de una interrupción en un sistema basado en microprocesador como también sobre la estructura y características de las interrupciones disponibles en la familia de microprocesadores Intel.

El propósito de las interrupciones

Las interrupciones son útiles en especial cuando se conectan dispositivos de E/S que proporcionan o requieren datos a velocidades de transferencia relativamente bajas. Por ejemplo, en el capítulo 11 mostramos el ejemplo de un teclado que utilizaba la operación de entrada estroboscópica del 82C55. En ese

FIGURA 12-1 Una línea de tiempo que indica el uso de las interrupciones en un sistema común.



ejemplo, el software sondeaba el 82C55 y su bit IBF para decidir si había datos disponibles desde el teclado. Si la persona que utilizaba el teclado escribía un carácter por segundo, el software para el 82C55 esperaba todo un segundo completo entre cada pulsación de tecla para que la persona pulsara otra tecla. Este proceso desperdiciaba mucho tiempo, lo cual hizo que los diseñadores desarrollaran otro, llamado *procesamiento de interrupciones*, para encargarse de esta situación.

A diferencia de la técnica de sondeo, el procesamiento de interrupciones permite al microprocesador ejecutar otro software mientras el operador del teclado piensa sobre cuál tecla desea pulsar a continuación. Tan pronto como se oprime una tecla, el codificador del teclado elimina los rebotes del interruptor y coloca un pulso de salida que interrumpe al microprocesador. Éste ejecuta otro software hasta que realmente se oprime una tecla, momento en el cual lee la tecla y regresa al programa que había sido interrumpido. Como resultado, el microprocesador puede imprimir informes o completar cualquier otra tarea mientras el operador está escribiendo un documento y pensando qué escribir a continuación.

La figura 12-1 muestra una línea de tiempo que indica cómo un mecanógrafo escribe datos con un teclado, cómo una impresora extrae datos de la memoria y cómo se ejecuta un programa. El programa es el programa principal que se interrumpe para cada pulsación de tecla y cada carácter que se va a imprimir en la impresora. Observe que el procedimiento de servicio de interrupciones del teclado que llama la interrupción del teclado y el procedimiento de servicio de interrupción de la impresora tardan cierto tiempo en ejecutarse.

Interrupciones

Las interrupciones de toda la familia de microprocesadores Intel incluyen dos terminales de hardware que solicitan interrupciones (INTR y NMI) y una terminal de hardware (INTA) que reconoce la interrupción solicitada a través de INTR. Además de las terminales, el microprocesador también tiene las interrupciones de software INT, INTO, INT 3 y BOUND. También se utilizan dos bits de bandera con la estructura de interrupciones: IF (bandera de interrupción) y TF (bandera de atrapamiento), además de una instrucción especial de retorno llamada IRET (o IRETD en los microprocesadores 80386, 80486 o Pentium-Pentium 4).

Vectores de interrupción. Los vectores de interrupción y la tabla de vectores son elementos imprescindibles para comprender las interrupciones de hardware y de software. La **tabla de vectores de interrupción** se encuentra en los primeros 1024 bytes de memoria, en las direcciones 000000H-0003FFH. Esta tabla contiene 256 vectores de interrupción de 4 bytes distintos. Un **vector de interrupción** contiene la dirección (segmento y desplazamiento) del procedimiento de servicio de interrupciones.

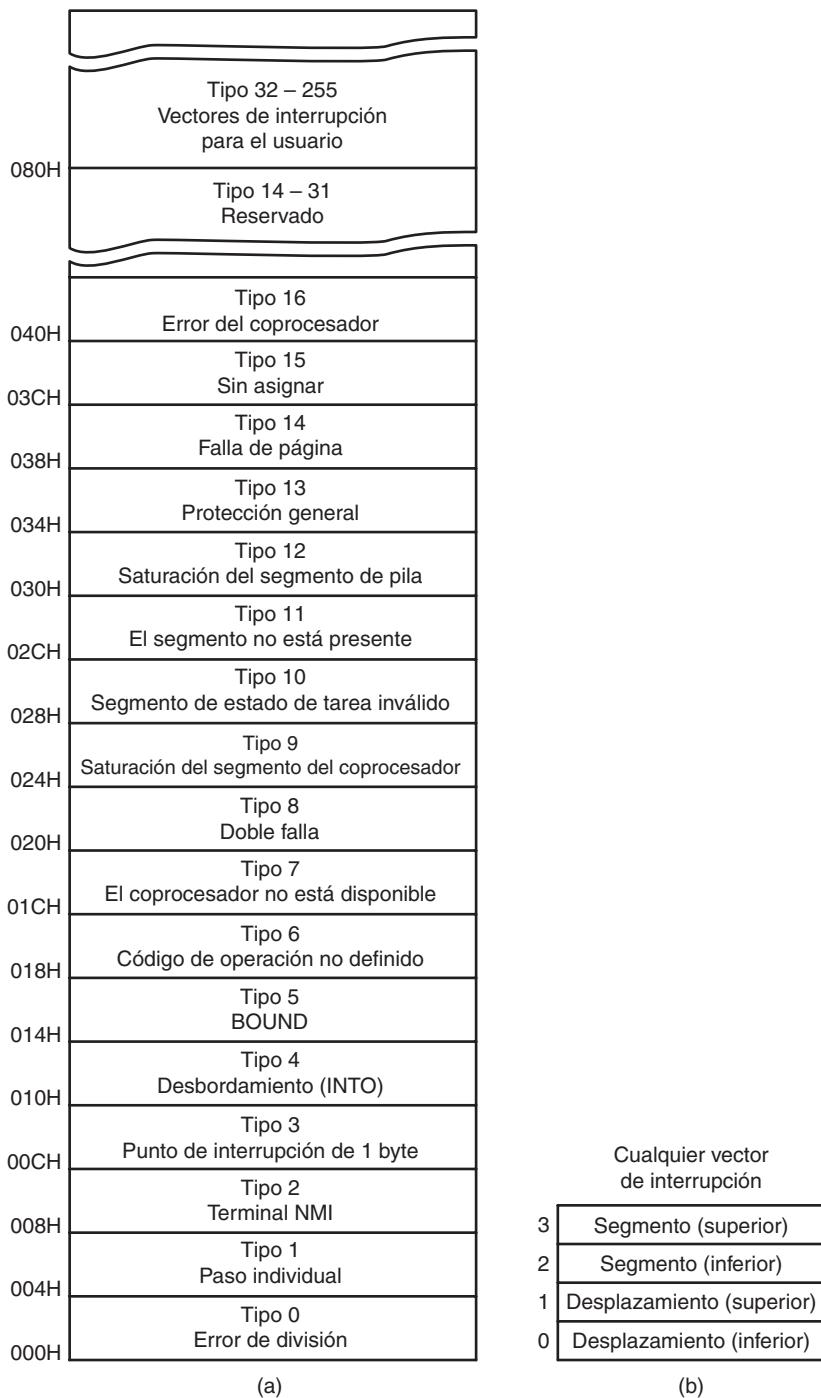
La figura 12-2 muestra la tabla de vectores de interrupción para el microprocesador. Los primeros cinco vectores de interrupción son idénticos en todos los miembros de la familia de microprocesadores Intel, del 8086 al Pentium. Hay otros vectores de interrupción para el 80286 que tienen compatibilidad ascendente con los microprocesadores 80386, 80486 y Pentium-Pentium 4, pero no tienen compatibilidad descendente con el 8086 o el 8088. Intel reserva los primeros 32 vectores de interrupción para usarlos en diversos miembros de su familia de microprocesadores. Los últimos 224 vectores están disponibles para el usuario. Cada vector tiene 4 bytes de longitud en el modo real y contiene la **dirección inicial** del procedimiento de servicio de interrupciones. Los primeros dos bytes del vector contienen la dirección de desplazamiento y los últimos dos bytes contienen la dirección de segmento.

La siguiente lista describe la función de cada interrupción dedicada en el microprocesador:

TIPO 0

El **error de división** se presenta cada vez que se desborda el resultado de una división o cuando se produce un intento de dividir entre cero.

FIGURA 12-2 (a) La tabla de vectores de interrupción para el microprocesador y (b) el contenido de un vector de interrupción.



TIPO 1

Paso individual o atrapamiento ocurre después de la ejecución de cada instrucción, si el bit de bandera de atrapamiento (TF) está activo. Al aceptar la interrupción el bit TF se borra, de manera que el procedimiento de servicio de interrupciones se ejecute a toda su velocidad. (Más adelante en esta sección veremos mayores detalles sobre esta interrupción.)

TIPO 2

La interrupción no enmascarable se produce cuando se coloca un 1 lógico en la terminal de entrada NMI que va al microprocesador. No enmascarable significa que la entrada no puede deshabilitarse.

TIPO 3	Una instrucción especial de un byte (INT 3) que utiliza el vector para acceder a su procedimiento de servicio de interrupciones. La instrucción INT 3 se utiliza a menudo para almacenar un punto de interrupción en un programa, para su depuración.
TIPO 4	Desbordamiento es un vector especial que se utiliza con la instrucción INTO. Esta instrucción interrumpe el programa si existe una condición de desbordamiento, como lo indica la bandera de desbordamiento (OF).
TIPO 5	La instrucción BOUND compara un registro con los límites almacenados en la memoria. Si el contenido del registro es mayor o igual que la primera palabra en memoria, y menor o igual que la segunda palabra, no se produce una interrupción ya que el contenido del registro está dentro de los límites. Si el contenido del registro está fuera de los límites se produce una interrupción tipo 5.
TIPO 6	Una interrupción tipo código de operación inválido se produce cada vez que se encuentra un código de operación no definido en un programa.
TIPO 7	La interrupción coprocesador no disponible se produce cuando no se encuentra un coprocesador en el sistema, como lo indican los bits de control del coprocesador de la palabra de estado del equipo (MSW o CR0). Si se ejecuta una instrucción ESC o WAIT y no se encuentra el coprocesador, se produce una excepción o interrupción de tipo 7.
TIPO 8	Una interrupción doble fallo se activa cada vez que se producen dos interrupciones separadas durante la misma instrucción.
TIPO 9	La interrupción saturación de segmento del coprocesador se produce si el operando de memoria de la instrucción ESC (código de operación del coprocesador) se extiende más allá de la dirección de desplazamiento FFFFH en modo real.
TIPO 10	Una interrupción segmento de estado de tarea inválido se produce en el modo protegido si el TSS es inválido debido a que el campo de límite de segmento no es 002BH o mayor. En la mayoría de los casos esto se debe a que el TSS no se inicializa.
TIPO 11	La interrupción segmento no está presente se produce cuando el bit P (P = 0) en modo protegido en un descriptor indica que el segmento no está presente o no es válido.
TIPO 12	La interrupción saturación del segmento de pila se produce si el segmento de pila no está presente (P = 0) en el modo protegido o si se excede el límite del segmento de pila.
TIPO 13	La interrupción fallo de protección general se produce en la mayoría de las violaciones de protección en el sistema 80286-Pentium 4 en modo protegido. (Estos errores ocurren en Windows como fallos de protección general.) A continuación se muestra una lista de estas violaciones de protección: (a) Se excedió el límite de la tabla de descriptores. (b) Violación a las reglas de privilegios. (c) Se cargó el tipo de segmento de descriptor inválido. (d) Escritura en un segmento de código protegido. (e) Lectura de un segmento de código de sólo ejecución. (f) Escritura en un segmento de datos de sólo lectura. (g) Se excedió el límite del segmento. (h) CPL = IOPL cuando se ejecuta CTS, HLT, LGDT, LIDT, LLDT, LMSW o LTR. (i) CPL > IOPL cuando se ejecuta CLI, IN, INS, LOCK, OUT, OUTS y STI.
TIPO 14	Las interrupciones fallo de página ocurren para cualquier fallo de página de acceso a memoria o al código en los microprocesadores 80386, 80486 y Pentium-Pentium 4.
TIPO 16	La interrupción error del coprocesador se produce cada vez que ocurre un error en el coprocesador (ERROR = 0) para las instrucciones ESCape o WAIT, sólo en los microprocesadores 80386, 80486 y Pentium-Pentium 4.
TIPO 17	Las interrupciones comprobaciones de alineación indican que se están direccio- nando datos tipo palabra y doble palabra en una posición de memoria impar (o en una

posición incorrecta, en el caso de una doble palabra). Esta interrupción está activa en los microprocesadores 80486 y Pentium-Pentium 4.

- TIPO 18** Una **comprobación de equipo** activa una interrupción en modo de administración de memoria del sistema en los microprocesadores Pentium-Pentium 4.

Instrucciones de interrupción: BOUND, INTO, INT, INT3 e IRET

De las cinco instrucciones de interrupción por software disponibles para el microprocesador, INT e INT 3 son muy similares, BOUND e INTO son condicionales e IRET es una instrucción especial de retorno de interrupción.

La instrucción BOUND, que tiene dos operandos, compara un registro con dos palabras de datos de memoria. Por ejemplo, si se ejecuta la instrucción BOUND AX,DATOS, AX se compara con el contenido de DATOS y de DATOS+1, y también con DATOS+2 y DATOS+3. Si AX es menor que el contenido de DATOS y de DATOS+1, se produce una interrupción tipo 5. Si AX es mayor que DATOS+2 y de DATOS+3, se produce una interrupción tipo 5. Si AX está dentro de los límites de estas dos palabras de memoria, no se produce ninguna interrupción.

La instrucción INTO evalúa la bandera de desbordamiento (OF). Si OF = 1, la instrucción INTO llama al procedimiento cuya dirección está almacenada en el tipo de vector de interrupción número 5. Si OF = 0, entonces la instrucción INTO no realiza ninguna operación y se ejecuta la siguiente instrucción secuencial en el programa.

La instrucción INT n llama al procedimiento de servicio de interrupciones que empieza en la dirección representada en el vector número n. Por ejemplo, una instrucción INT 80H o INT 128 llama al procedimiento de servicio de interrupciones cuya dirección se almacena en el tipo de vector número 80H (00200H-00203H). Para determinar la dirección del vector, sólo se multiplica el número de tipo de vector (n) por 4, lo cual proporciona la dirección inicial del vector de interrupción de cuatro bytes. Por ejemplo, INT 5 = 4 × 5 o 20 (14H). El vector para INT 5 comienza en la dirección 0014H y continúa hasta 0017H. Cada instrucción INT se almacena en dos bytes de memoria: el primer byte contiene el código de operación y el segundo byte contiene el número de tipo de interrupción. La única excepción a esto es la instrucción INT 3, que es una instrucción de un byte. Esta instrucción se utiliza a menudo como interrupción de punto de interrupción, ya que es fácil insertar una instrucción de un byte en un programa. Los puntos de interrupción se utilizan con frecuencia para depurar errores en el software.

La instrucción IRET es una instrucción especial de retorno que se utiliza para regresar de las interrupciones de software y de hardware. La instrucción IRET es muy parecida a una instrucción RET lejana, ya que recupera la dirección de retorno de la pila. Es distinta a la instrucción de retorno cercana, ya que también recupera de la pila una copia del registro de banderas. Una instrucción IRET extrae seis bytes de la pila: dos para el registro IP, dos para CS y dos para las banderas.

En los microprocesadores del 80386 al Pentium 4 también hay una instrucción IRETD, ya que estos microprocesadores pueden meter el registro EFLAG (32 bits) en la pila, así como el registro EIP de 32 bits en modo protegido y el registro del segmento de código 16 bits. Si se opera en el modo real, utilizamos la instrucción IRET con los microprocesadores del 80386 al Pentium 4.

La operación de una interrupción en modo real

Cuando el microprocesador termina de ejecutar la instrucción actual, determina si hay una interrupción activa mediante la comprobación de (1) las ejecuciones de instrucciones, (2) un paso individual, (3) NMI, (4) el desbordamiento del segmento del coprocesador, (5) INTR y (6) la instrucción INT, todo en el orden presentado. Si hay una o más de estas condiciones de interrupción presentes, se produce la siguiente secuencia de eventos:

1. El contenido del registro de banderas se mete a la pila.
2. Se borran las banderas de interrupción (IF) y de atrapamiento (TF). Esto deshabilita la terminal INTR y la característica de atrapamiento o de paso individual.
3. El contenido del registro del segmento de código (CS) se mete en la pila.
4. El contenido del apuntador de instrucciones (IP) se mete en la pila.

5. Se obtiene el contenido del vector de interrupción y luego se coloca en IP y en CS para que la siguiente instrucción se ejecute en el procedimiento de servicio de interrupciones direccionado por el vector.

Cada vez que se acepta una interrupción, el microprocesador apila el contenido del registro de banderas, de CS y de IP; después salta al procedimiento direccionado por el vector de interrupción. Una vez que se meten las banderas en la pila, se borra el contenido de IF y de TF. Estas banderas se regresan al estado que tenían antes de la interrupción cuando se encuentra la instrucción IRET al final del procedimiento de servicio de interrupciones. Por lo tanto, si se habilitan las interrupciones antes del procedimiento de servicio de interrupciones, se rehabilitan automáticamente gracias a la instrucción IRET al final del procedimiento.

La dirección de retorno (en CS e IP) se mete en la pila durante la interrupción. Algunas veces la dirección de retorno apunta a la siguiente instrucción en el programa; algunas veces apunta a la instrucción o al punto en el programa en donde ocurrió la interrupción. Los números de tipo de interrupción 0, 5, 6, 7, 8, 10, 11, 12 y 13 meten una dirección de retorno que apunta a la instrucción ofensora, en vez de apuntar a la siguiente instrucción en el programa. Esto permite al procedimiento de servicio de interrupciones la posibilidad de volver a tratar de ejecutar la instrucción en ciertos casos de error.

Algunas de las interrupciones en modo protegido (los tipos 8, 10, 11, 12 y 13) colocan un código de error en la pila después de la dirección de retorno. El código de error identifica el selector que produjo la interrupción. En casos en los que no hay selector involucrado, el código de error es 0.

Operación de una interrupción en modo protegido

En el modo protegido, las interrupciones tienen las mismas asignaciones que en el modo real, sólo que la tabla de vectores de interrupción es distinta. En lugar de vectores de interrupción, el modo protegido utiliza un conjunto de 256 descriptores de interrupción que se almacenan en una tabla de descriptores de interrupción (IDT). Esta tabla es de 256×8 (2 K) bytes de longitud, en donde cada descriptor contiene ocho bytes. La tabla de descriptores de interrupción se ubica en cualquier posición de memoria en el sistema mediante el registro de dirección de la tabla de descriptores (IDTR).

Cada entrada en la IDT contiene la dirección del procedimiento de servicio de interrupciones en la forma de un selector de segmento y una dirección de desplazamiento de 32 bits. También contiene el bit P (presente) y los bits DPL para describir el nivel de privilegios de la interrupción. La figura 12-3 muestra el contenido del descriptor de interrupción.

Los vectores de interrupción en modo real pueden convertirse en interrupciones en modo protegido si se copian las direcciones de los procedimientos de interrupción de la tabla de vectores de interrupción y se convierten en direcciones de desplazamiento de 32 bits que se almacenan en los descriptores de interrupción. Puede colocarse un solo selector y descriptor de segmento en la tabla de descriptores global que identifica el primer 1 Mbyte de memoria como el segmento de interrupciones.

Con la excepción de la IDT y los descriptores de interrupción, la interrupción en modo protegido funciona de igual forma que la interrupción en modo real. Para regresar de ambas interrupciones se utiliza la instrucción IRET o IRETD. La única diferencia es que en modo protegido el microprocesador accede a la IDT, en vez de a la tabla de vectores de interrupción.

FIGURA 12-3

El descriptor de interrupción en modo protegido.

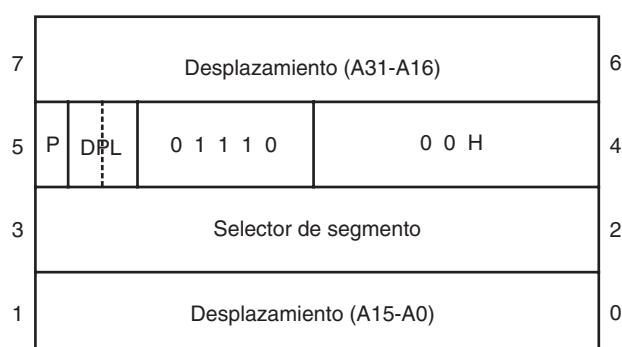
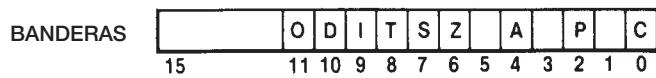


FIGURA 12-4 El registro de banderas. (Cortesía de Intel Corporation.)



Bits de bandera de interrupción

La bandera de interrupción (IF) y la bandera de atrapamiento (TF) se borran después de meter a la pila el contenido del registro de banderas durante una interrupción. La figura 12-4 muestra el contenido del registro de banderas junto con la posición de IF y TF. Cuando se activa el bit IF, éste permite que la terminal INTR produzca una interrupción. Cuando TF = 1, éste produce una interrupción de atrapamiento (número de tipo 1) después de la ejecución de cada instrucción. Ésta es la razón por la que a menudo al proceso de atrapar se le conoce como *paso individual*. Cuando TF = 0 se produce la ejecución normal del programa. Este bit de bandera permite la depuración, como explicaremos en los capítulos del 17 al 19, en los que se detallan los microprocesadores del 80386 al Pentium 4.

La bandera de interrupción se activa y se borra mediante las instrucciones STI y CLI, respectivamente. No hay instrucciones especiales que activen o que borren la bandera de atrapamiento. El ejemplo 12-1 muestra un procedimiento de servicio de interrupciones que activa el rastreo al activar el bit de bandera de atrapamiento en la pila, desde el interior del procedimiento. El ejemplo 12-2 muestra un procedimiento de servicio de interrupciones que desactiva el rastreo al borrar la bandera de atrapamiento en la pila, desde el interior del procedimiento.

EJEMPLO 12-1

```
;Un procedimiento que activa el bit de bandera TRAP para habilitar el atrapamiento
TRON      PROC    FAR USES AX BP
          MOV     BP,SP           ;obtiene SP
          MOV     AX,[BP+8]        ;recupera las banderas de la pila
          OR      AH,1            ;activa la bandera de atrapamiento
          MOV     [BP+8],AX
          IRET
TRON      ENDP
```

EJEMPLO 12-2

```
;Un procedimiento que borra la bandera TRAP para deshabilitar el atrapamiento
TROFF     PROC    FAR USES AX BP
          MOV     BP,SP           ;obtiene SP
          MOV     AX,[BP+8]        ;recupera las banderas de la pila
          AND     AH,0FEH         ;borra la bandera de atrapamiento
          MOV     [BP+8],AX
          IRET
TROFF     ENDP
```

En ambos ejemplos, el registro de banderas se recupera de la pila mediante el uso del registro BP, el cual direcciona el segmento de pila de manera predeterminada. Una vez que se recuperan las banderas, se activa (TRON) o se borra (TROFF) el bit TF antes de regresar del procedimiento de servicio de interrupciones. La instrucción IRET restaura el registro de banderas con el nuevo estado de la bandera de atrapamiento.

Procedimiento de rastreo. Vamos a suponer que una instrucción INT 40H accede a TRON y que una instrucción INT 41H accede a TROFF. El ejemplo 12-3 rastrea mediante un programa inmediatamente después de la instrucción INT 40H. El procedimiento de servicio de interrupciones que se muestra en el ejemplo 12-3 responde al número de tipo de interrupción 1, o interrupción de atrapamiento. Cada vez que ocurre un atrapamiento (después de que cada instrucción ejecuta la siguiente INT 40H) el procedimiento RASTREA almacena el contenido de todos los registros de 32 bits del microprocesador en un

arreglo llamado REGS. Esto proporciona un rastreo de registros de todas las instrucciones entre INT 40H (TRON) e INT 41H (TROFF), si se almacena el contenido de los registros almacenados en el arreglo.

EJEMPLO 12-3

```

REGS      DD     8 DUP(?)           ;espacio para los registros

RASTREA  PROC  FAR  USES EBX

        MOV   EBX,OFFSET REGS
        MOV   [EBX],EAX          ;almacena EAX
        POP   EAX
        PUSH  EAX
        MOV   [EBX+4],EAX         ;almacena EBX
        MOV   [EBX+8],ECX         ;almacena ECX
        MOV   [EBX+12],EDX        ;almacena EDX
        MOV   [EBX+16],ESP        ;almacena ESP
        MOV   [EBX+20],EBP        ;almacena EBP
        MOV   [EBX+24],ESI        ;almacena ESI
        MOV   [EBX+28],EDI        ;almacena EDI
        IRET

RASTREA  ENDP

```

Almacenamiento de un vector de interrupción en la tabla de vectores

Para poder instalar un vector de interrupción (al cual se le conoce algunas veces como **gancho**), el ensamblador debe direccionar memoria absoluta. Vea en el ejemplo 12-4 cómo se agrega un nuevo vector a la tabla de vectores de interrupción mediante el uso del ensamblador y la llamada a una función del DOS. Aquí se instala en memoria el vector para INT 40H (para el procedimiento de interrupción NUEV40) en las posiciones de vector 100H-103H en modo real. Lo primero que hace el procedimiento es almacenar el contenido anterior del vector de interrupción, en caso de que necesitemos desinstalarlo. Podemos omitir este paso si no hay necesidad de desinstalar la interrupción.

La función AX = 3100H para INT 21H (la función de acceso al DOS) instala el procedimiento NUEV40 en memoria, que se mantendrá ahí hasta que la computadora se apague. El número en DX es la longitud del software en párrafos (trozos de 16 bytes). En el apéndice A podrá obtener más detalles acerca de esta función del DOS.

EJEMPLO 12-4

```

.MODEL  TINY
.CODE
.STARTUP

        JMP   INICIO
ANTER  DD     ?           ;espacio para el vector anterior

NUEV40  PROC  FAR

;
;Software de interrupción para INT 40H
;

NUEV40 ENDP

;inicio de la instalación

INICIO:
        MOV   AX,0             ;direcciona el segmento 0000H
        MOV   DS,AX
        MOV   AX,DS:[100H]       ;obtiene el desplazamiento de INT 40H
        MOV   WORD PTR CS:ANTER,AX ;lo almacena
        MOV   AX,DS:[102H]       ;obtiene el segmento de INT 40H
        MOV   WORD PTR CS:ANTER+2,AX ;lo almacena
        MOV   DS:[100H],OFFSET NUEV40 ;almacena el desplazamiento

```

```

MOV DS: [102H], CS           ;almacena el segmento
MOV DX, OFFSET INICIO
SHR DX, 4
INC DX
MOV AX, 3100H                ;hace residente a NUEV40
INT 21H

END

```

12-2**INTERRUPCIONES DE HARDWARE**

El microprocesador tiene dos entradas de interrupción de hardware: interrupción no enmascarable (NMI) y petición de interrupción (INTR). Cada vez que se activa la entrada NMI se produce una interrupción tipo 2 debido a que NMI se decodifica en forma interna. La entrada INTR debe decodificarse en forma externa para seleccionar un vector. Puede elegirse cualquier vector de interrupción para la terminal INTR, pero por lo general utilizamos un número de tipo de interrupción entre 20H y FFH. Intel ha reservado las interrupciones de 00H a 1FH para uso interno y una expansión a futuro. La señal INTA también es una terminal de interrupción en el microprocesador, pero es una salida que se utiliza en respuesta a la entrada INTR para aplicar un número de tipo de vector a las conexiones D₇-D₀ del bus de datos. La figura 12-5 muestra las tres conexiones de interrupción para el usuario en el microprocesador.

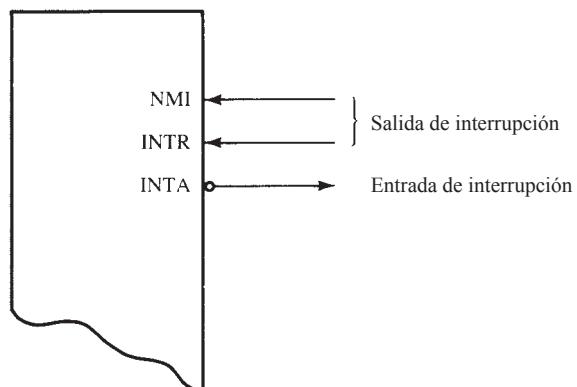
La **interrupción no enmascarable** (NMI) es una entrada activada por los bordes del ciclo de reloj; esta entrada solicita una interrupción en el borde positivo (transición de 0 a 1). Después de un borde positivo, la terminal NMI debe permanecer en 1 lógico hasta que sea reconocida por el microprocesador. Para que pueda reconocerse el borde positivo, la terminal NMI debe ser un 0 lógico durante dos períodos de reloj cuando menos.

La entrada NMI se utiliza a menudo para los errores de paridad y otras fallas importantes en el sistema, como los cortes de energía. Este tipo de fallas se detectan fácilmente mediante el monitoreo de la línea de energía CA, en donde se produce una interrupción cada vez que se pierde la energía CA. En respuesta a este tipo de interrupción, el microprocesador almacena todos los registros internos en una memoria con respaldo de batería o en una EEPROM. La figura 12-6 muestra un circuito de detección de falla de energía que proporciona un 1 lógico a la entrada NMI cada vez que se interrumpe la energía CA.

En este circuito, un aislador óptico proporciona el aislamiento de la línea de energía CA. La salida del aislador se forma mediante un inversor de disparo Schmitt, el cual proporciona un pulso de 60 Hz a la entrada de disparo del multivibrador 74LS122 monoestable y redispersable. Los valores de R y de C se eligen de manera que el 74LS122 tenga una anchura de pulso activo de 33 ms, o 2 períodos de entrada CA. Como el 74LS122 es redispersable, mientras se aplique energía CA la salida Q permanecerá disparada en un 1 lógico y \bar{Q} permanecerá en un 0 lógico.

Si falla la energía CA, el 74LS122 no seguirá recibiendo pulsos de disparo del 74ALS14, lo que significa que Q se volverá un 0 lógico y \bar{Q} se volverá un 1 lógico, con lo cual se interrumpirá al microprocesador mediante la terminal NMI. El procedimiento de servicio de interrupciones (que no se muestra aquí) almacena el contenido de todos los registros internos y demás datos en una memoria con respaldo

FIGURA 12-5
Las terminales de interrupción en todas las versiones del microprocesador Intel.



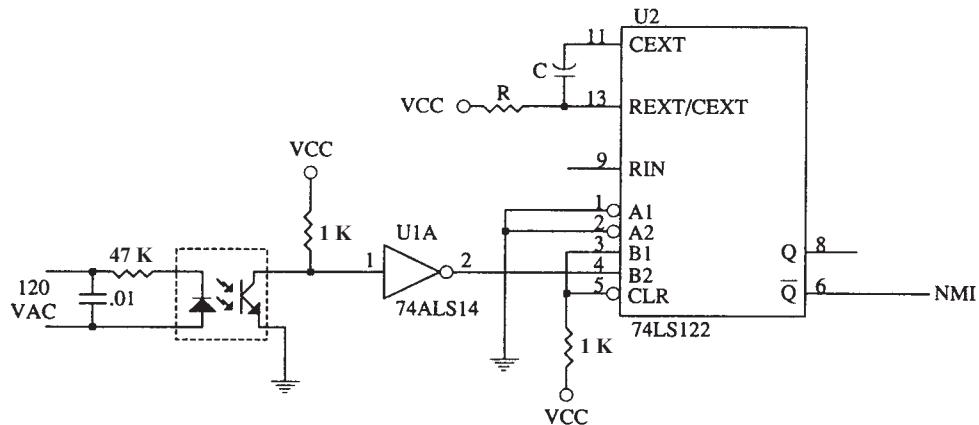
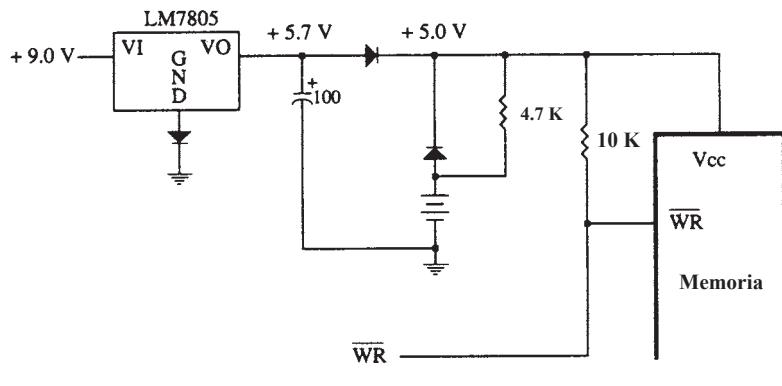


FIGURA 12-6 Un circuito de detección de falla de energía.

FIGURA 12-7 Un sistema de memoria con respaldo de batería de NiCad, litio o celda de gel.



de batería. Este sistema supone que la fuente de energía tiene un filtro capacitor lo suficientemente grande como para proporcionar energía durante al menos 75 ms después del corte de energía CA.

Observe en la figura 12-7 un circuito que suministra energía a una memoria cuando falla la energía CD. Aquí se utilizan diodos para intercambiar los voltajes de suministro entre la fuente de energía CD y la batería. Los diodos utilizados son diodos estándar de silicio ya que el suministro de energía para este circuito de memoria se eleva de +5.0 V a +5.7 V. La resistencia se utiliza para una carga continua de compensación de la batería, la cual puede ser de NiCad, litio o una celda de gel.

Cuando falla la energía CD, la batería proporciona un voltaje reducido a la conexión V_{CC} del dispositivo de memoria. La mayoría de los dispositivos de memoria pueden retener los datos con voltajes de V_{CC} tan bajos como 1.5 V, por lo que el voltaje de la batería no necesita ser de +5.0 V. La terminal WR se lleva a V_{CC} durante un corte de energía, por lo que no se escribirán datos en la memoria.

INTR e INTA

La entrada de petición de interrupción (INTR) es sensible al nivel, lo cual significa que debe mantenerse en un nivel de 1 lógico hasta que sea reconocida. La terminal INTR se activa mediante un evento externo y se borra dentro del procedimiento de servicio de interrupciones. Esta entrada se deshabilita de manera automática una vez que el microprocesador la acepta, y se rehabilita mediante la instrucción IRET al final del procedimiento de servicio de interrupciones. Los microprocesadores del 80386 al Pentium 4 utilizan la instrucción IRETD cuando operan en modo protegido.

El microprocesador responde a la terminal INTR mediante el envío de un pulso en la salida INTA antes de recibir un número de tipo de vector de interrupción en las conexiones D₇-D₀ del bus de datos. En la figura 12-8 verá el diagrama de sincronización para las terminales INTR e INTA del

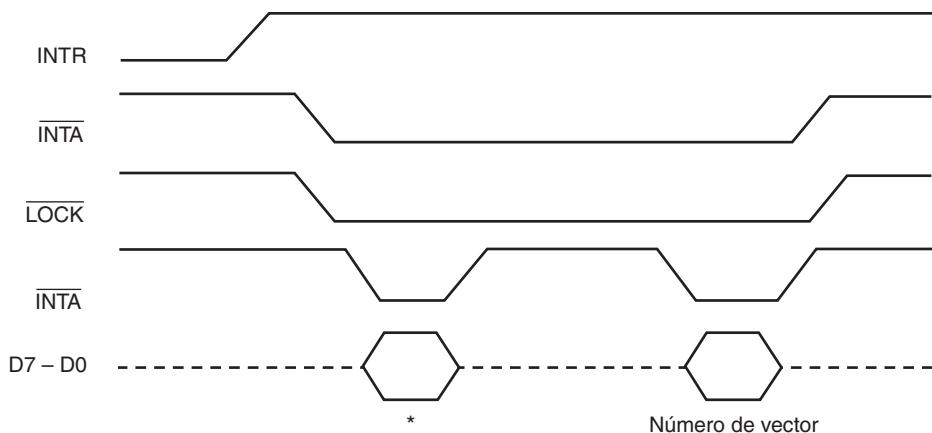
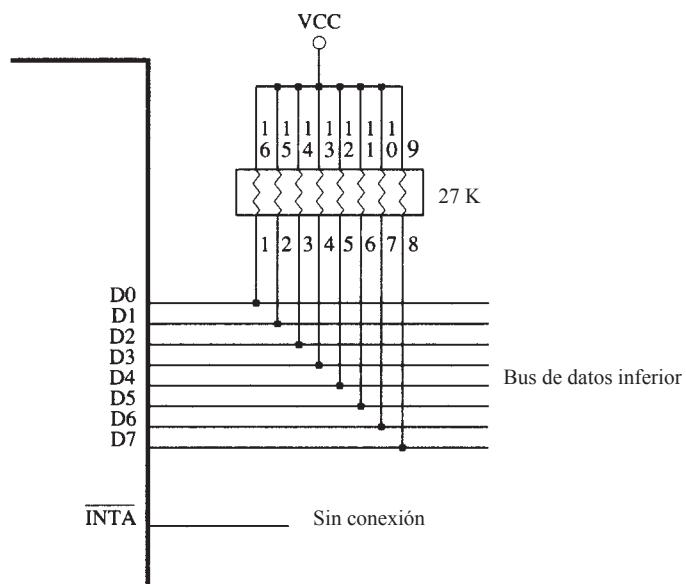


FIGURA 12-8 La sincronización de las entradas INTR e INTA. *Esta porción del bus de datos se ignora y por lo general contiene el número de vector.

FIGURA 12-9 Un método simple para generar el número de tipo de vector de interrupción FFH, en respuesta a INTR.



microprocesador. Hay dos pulsos INTA que se generan mediante el sistema y se utilizan para insertar el número de tipo de vector en el bus de datos.

La figura 12-9 muestra un circuito simple que aplica el número de tipo de vector de interrupción FFH al bus de datos, en respuesta a una INTR. Observe que la terminal INTA no está conectada en este circuito. Como se utilizan resistencias para llevar las conexiones del bus de datos (D₀-D₇) al nivel alto, el microprocesador ve de manera automática el número de tipo de vector FFH en respuesta a la entrada INTR. Ésta es la manera menos costosa de implementar la terminal INTR en el microprocesador.

Uso de un búfer de tres estados para INTA. Vea en la figura 12-10 cómo se aplica el número de tipo de vector de interrupción 80H al bus de datos (D₀-D₇) en respuesta a una INTR. Para responder a INTR, el microprocesador envía la señal INTA que se utiliza para habilitar un búfer octal 74ALS244 de tres estados. Este búfer octal aplica el número de tipo de vector de interrupción al bus de datos en respuesta al pulso INTA. Es fácil cambiar el número de tipo de vector mediante los interruptores DIP que se muestran en esta ilustración.

Cómo hacer que la entrada INTR se active en los bordes. A menudo se necesita una entrada que se active en los bordes, en vez de una entrada sensible al nivel. La entrada INTR puede convertirse a entrada que se active en los bordes mediante el uso de un flip-flop tipo D, como lo muestra la figura

FIGURA 12-10

Un circuito que aplica cualquier número de tipo de vector de interrupción en respuesta a INTA. Aquí, el circuito está aplicando el número de tipo 80H.

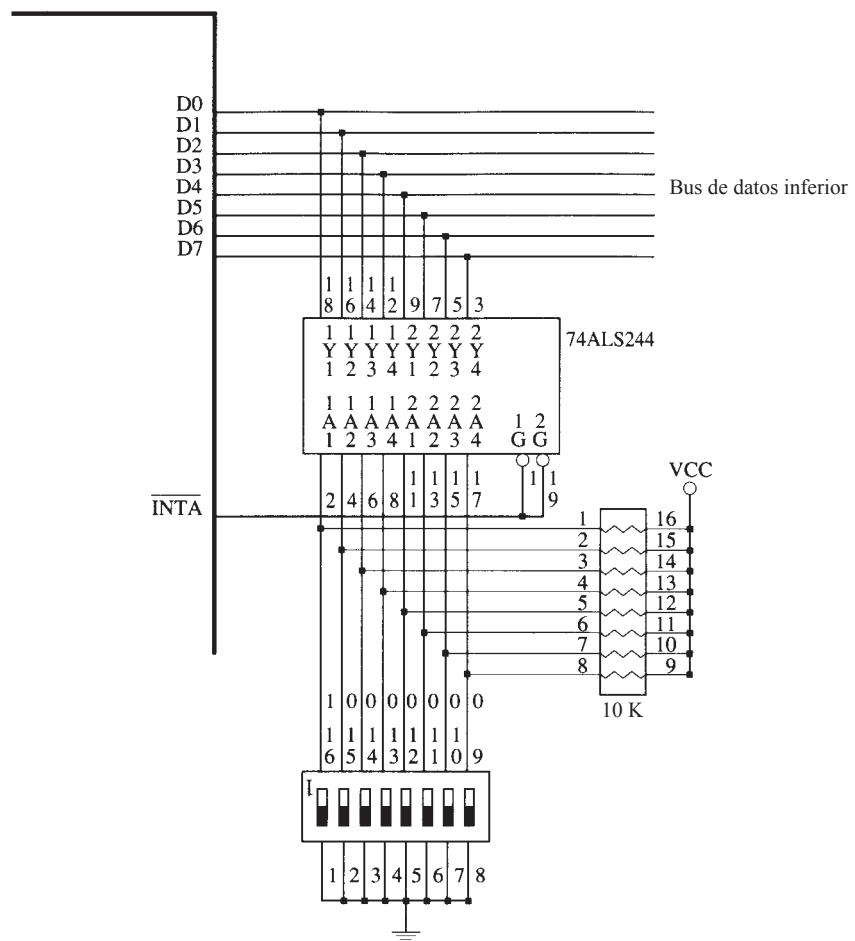
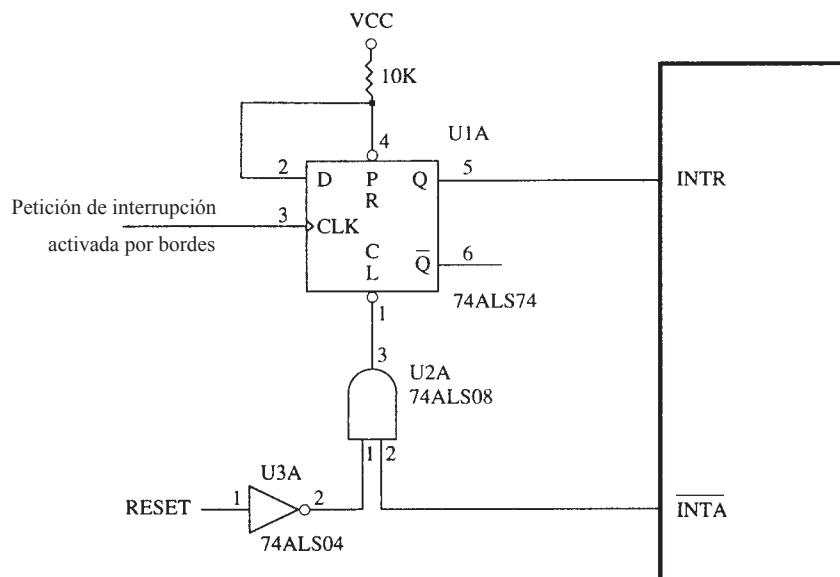


FIGURA 12-11 Conversión de INTR en una entrada de petición de interrupción activada por bordes.



12-11. Aquí, la entrada de reloj se convierte en una entrada de petición de interrupción activada por bordes y la entrada CL (borrado) se utiliza para borrar la petición cuando el microprocesador envía la señal INTA. En un principio, la señal RESET borra el flip-flop de manera que no se solicite ninguna interrupción cuando se enciende el sistema.

La interrupción de teclado del 82C55

El ejemplo del teclado que presentamos en el capítulo 11 es un ejemplo sencillo de la operación de la entrada INTR y de una interrupción. La figura 12-12 exhibe la interconexión del 82C55 con el microprocesador y el teclado. También muestra cómo se utiliza un búfer octal 74ALS244 para proporcionar al microprocesador el número de tipo de vector de interrupción 40H, en respuesta a la interrupción del teclado durante el pulso INTA.

El 82C55 se decodifica en las direcciones de puerto de E/S 0500H, 0502H, 0504H y 0506H del 80386SX mediante un PLD (no se muestra el programa). El 82C55 se opera en modo 1 (modo de

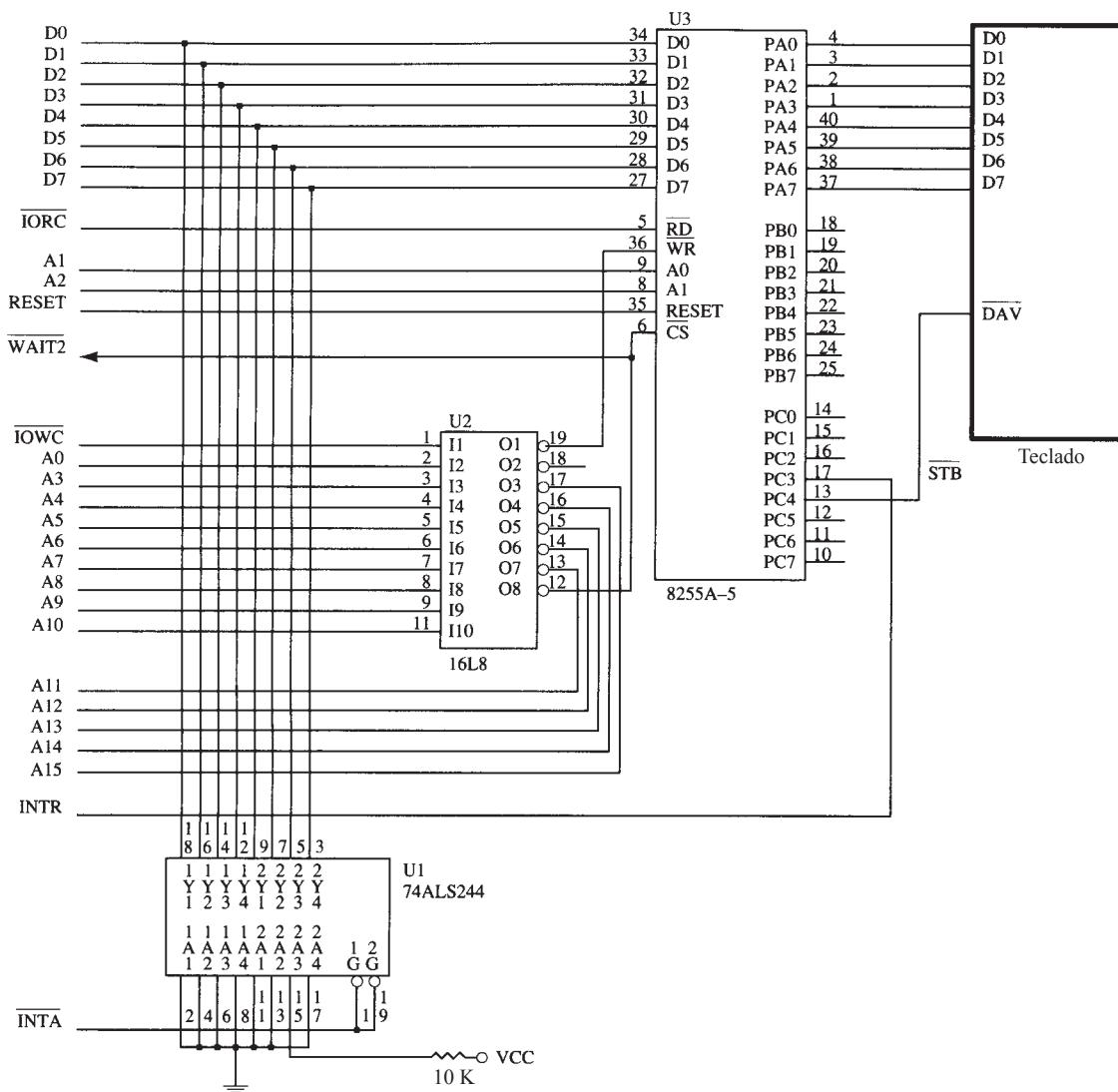


FIGURA 12-12 Un 82C55 conectado a un teclado desde el sistema de microprocesador, mediante el uso del vector de interrupción 40H.

entrada estroboscópica), por lo que cada vez que se pulsa una tecla, la salida INTR (PC3) se vuelve un 1 lógico y solicita una interrupción a través de la terminal INTR en el microprocesador. Esta terminal permanece en nivel alto hasta que se leen los datos ASCII del puerto A. En otras palabras, cada vez que se oprime una tecla el 82C55 solicita una interrupción tipo 40H a través de la terminal INTR. La señal DAV del teclado hace que los datos se enclaven en el puerto A y que INTR se vuelva un 1 lógico.

El ejemplo 12-5 expone el procedimiento de servicio de interrupciones para el teclado. Es muy importante que todos los registros que se vean afectados por una interrupción se almacenen antes de utilizarlos. En el software requerido para inicializar el 82C55 (el cual no se muestra aquí) la FIFO se inicializa de manera que ambos apuntadores sean iguales, la terminal de petición INTR se habilita a través del bit INTE dentro del 82C55 y se programa el modo de operación.

EJEMPLO 12-5

;Un procedimiento de servicio de interrupciones que lee una tecla desde
;el teclado que se muestra en la figura 12-12.

```

PUERTOA EQU    500H
CNTR    EQU    506H

FIFO    DB     256 DUP(?)      ;cola

INP     DD     FIFO           ;apuntador de entrada
OUTP   DD     FIFO           ;apuntador de salida

TECLA  PROC   FAR USES EAX EBX EDX EDI
        MOV EBX,CS:INP          ;obtiene los apuntadores
        MOV EDI,CS:OUTP

        INC BL
        .IF BX == DI           ;si está lleno
            MOV AL,8
            MOV DX,CNTR
            OUT DX,AL             ;deshabilita la interrupción del 82C55
        .ELSE
            DEC BL
            MOV DX,PUERTOA
            IN AL,DX               ;lee el código de tecla
            MOV CS:[BX]              ;lo almacena en la cola
            INC BYTE PTR CS:INP
        .ENDIF
        IRET
TECLA  ENDP

```

El procedimiento es corto debido a que el microprocesador ya sabe que los datos del teclado están disponibles cuando se hace la llamada a este procedimiento. Los datos se introducen desde el teclado y luego se almacenan en el búfer o cola FIFO (primero en entrar, primero en salir). La mayoría de las interfaces de teclado contienen una FIFO de cuando menos 16 bytes de longitud. Observe cómo se utiliza la instrucción INC BYTE PTR CX:INP para sumar 1 al apuntador de entrada y también para asegurarse que siempre direccione los datos en la cola.

Este procedimiento primero comprueba si la FIFO está llena. Esta condición se indica cuando el apuntador de entrada (INP) está un byte por debajo del apuntador de salida (OUTP). Si la FIFO está llena, la interrupción se deshabilita con un comando de activación/borrado de bit para el 82C55 y el programa regresa de la interrupción. Si la FIFO no está llena, los datos se introducen del puerto A y el apuntador de entrada se incrementa antes del retorno.

El ejemplo 12-6 muestra el procedimiento para extraer los datos de la FIFO. Este procedimiento primero determina si la FIFO está vacía mediante una comparación de los dos apuntadores. Si son iguales, la FIFO está vacía y el software espera en el ciclo VACIA, en donde evalúa los apuntadores de forma continua. El ciclo VACIA se interrumpe mediante la interrupción del teclado, la cual almacena los datos en la FIFO para que ya no esté vacía. Este procedimiento regresa con el carácter en el registro AH.

EJEMPLO 12-6

```
;Un procedimiento que lee datos de la cola del ejemplo 12-5
;y los regresa en AH;

LEEC      PROC    FAR USES EBX EDI EDX
          .REPEAT
              MOV EB,CS:INP           ;obtiene los apuntadores
              MOV EDI,CS:OUTP
              .UNTIL EBX == EDI       ;mientras esté vacía
              MOV AH,CS:[EDI]         ;obtiene los datos
              MOV AL,9
              MOV DX,CNTR
              OUT DX,AL               ;habilita la interrupción del 82C55
              INC BYTE PTR CS:OUTP
              RET
          LEEC      ENDP
```

12-3**EXPANSIÓN DE LA ESTRUCTURA DE INTERRUPCIONES**

En este libro cubriremos tres de los métodos más comunes para expandir la estructura de interrupciones del microprocesador. En esta sección explicaremos (mediante software y cierta modificación de hardware del circuito que se muestra en la figura 12-10) cómo es posible expandir la entrada INTR, de manera que acepte siete entradas de interrupción. También explicaremos cómo “encadenar en margarita” (daisy-chain) las interrupciones mediante el sondeo de software. En la siguiente sección describiremos una tercera técnica, en la que pueden agregarse hasta 63 entradas de interrupción mediante el controlador de interrupciones programable 8259A.

Uso del 74ALS244 para expandir las interrupciones

La modificación que se expone en la figura 12-13 permite que el circuito de la figura 12-10 acepte hasta siete entradas de interrupción adicionales. El único cambio en el hardware es la adición de una compuerta NAND de ocho entradas, la cual proporciona la señal INTR al microprocesador cuando se activa cualquiera de las entradas IR.

Operación. Si cualquiera de las entradas \overline{IR} se vuelve un 0 lógico, entonces la salida de la compuerta NAND se vuelve un 1 lógico y solicita una interrupción a través de la entrada INTR. El vector de interrupción que se obtenga durante el pulso INTA dependerá de cuál línea de petición de interrupción se active. La tabla 12-1 muestra los vectores de interrupción utilizados por una sola entrada de petición de interrupción.

Si se activan dos o más entradas de petición de interrupción en forma simultánea, se genera un nuevo vector de interrupción. Por ejemplo, si se activan $\overline{IR1}$ e $\overline{IR0}$, el vector de interrupción que se genera es FCH (252). La prioridad se resuelve en esta ubicación. Si la entrada $\overline{IR0}$ va a tener la prioridad más alta, la dirección del vector para $\overline{IR0}$ se almacena en la posición de vector FCH. Debe utilizarse toda la mitad superior de la tabla de vectores y sus 128 vectores de interrupción para dar cabida a todas las posibles condiciones de estas siete entradas de petición de interrupción. Esto parece un desperdicio, pero en muchas aplicaciones dedicadas es un método efectivo en costo para expandir las interrupciones.

Interrupción en cadena tipo margarita

La expansión mediante una interrupción en cadena tipo margarita es mejor en muchos aspectos que el uso del 74ALS244, ya que sólo requiere un vector de interrupción. La tarea de determinar la prioridad se deja al procedimiento de servicio de interrupciones. Para establecer la prioridad en una cadena tipo margarita no se requiere tiempo de ejecución adicional para el software, pero en general éste método es mucho mejor para expandir la estructura de interrupciones del microprocesador.

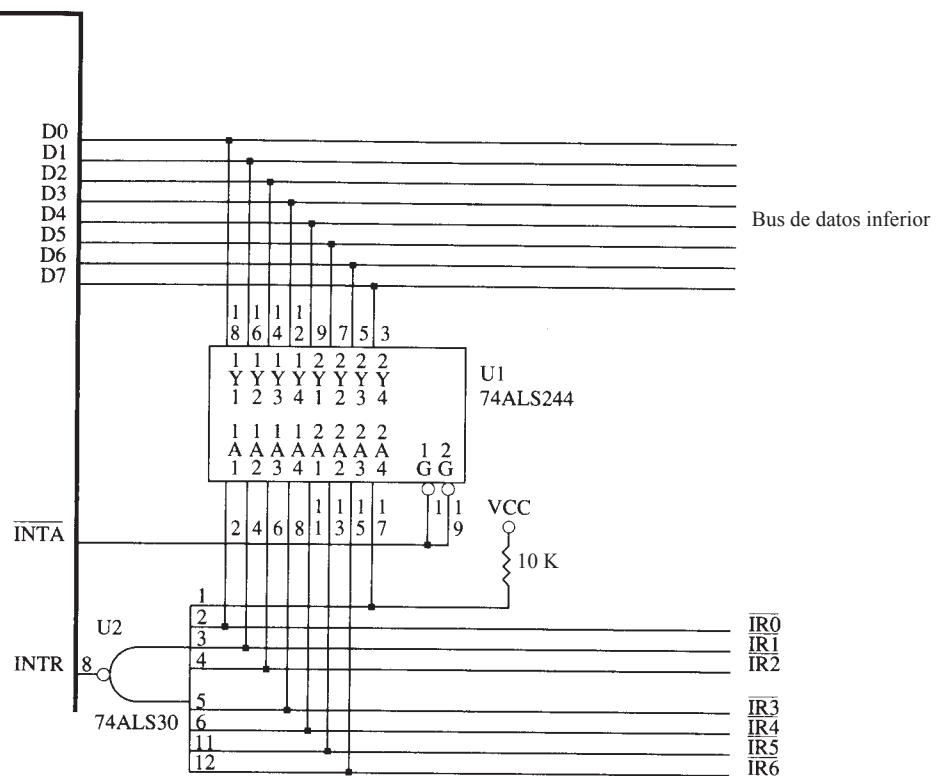


FIGURA 12-13 Expansión de la entrada INTR, de una a siete líneas de petición de interrupción.

TABLA 12-1 Peticiones de interrupción individuales para la figura 12-13.

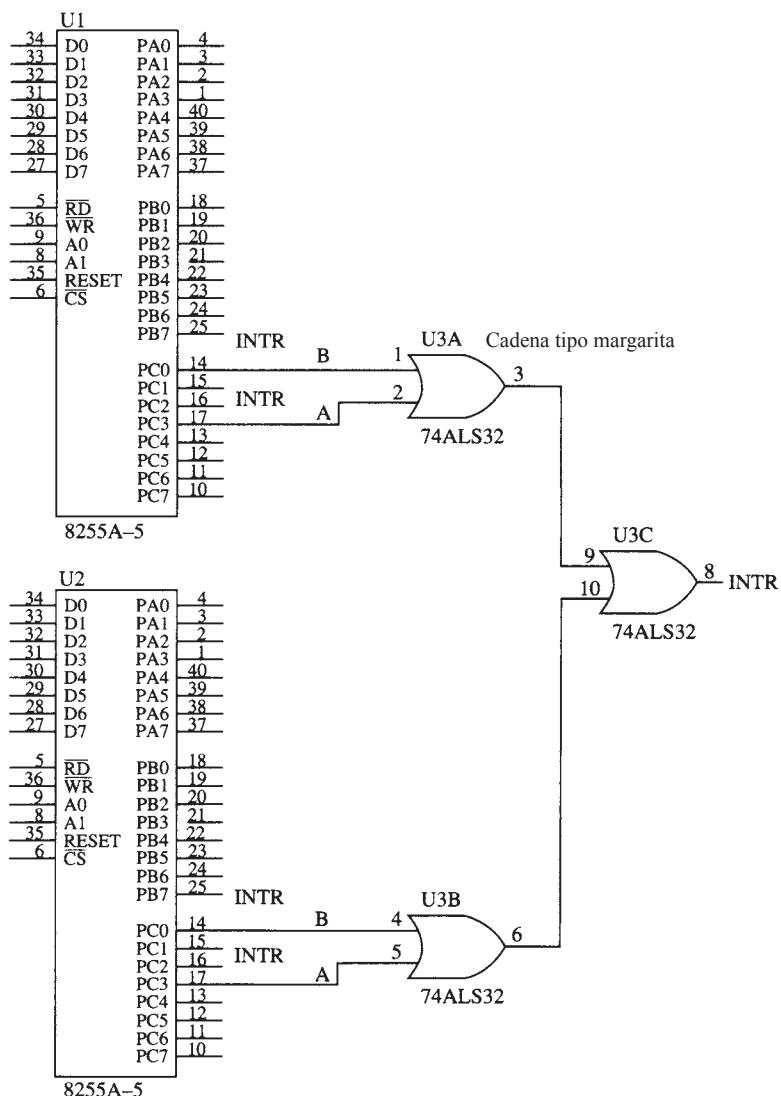
$\overline{IR6}$	$\overline{IR5}$	$\overline{IR4}$	$\overline{IR3}$	$\overline{IR2}$	$\overline{IR1}$	$\overline{IR0}$	Vector
1	1	1	1	1	1	0	FEH
1	1	1	1	1	0	1	FDH
1	1	1	1	0	1	1	FBH
1	1	1	0	1	1	1	F7H
1	1	0	1	1	1	1	EFH
1	0	1	1	1	1	1	DFH
0	1	1	1	1	1	1	BFH

La figura 12-14 muestra un conjunto de dos interfaces periféricas 82C55 con sus cuatro salidas INTR encadenadas en margarita y conectadas a la entrada INTR individual del microprocesador. Si cualquier salida de interrupción se vuelve un 1 lógico, también la entrada INTR del microprocesador se vuelve 1 lógico, con lo que se produce una interrupción.

Cuando se utiliza una cadena tipo margarita para solicitar una interrupción, es mejor llevar las conexiones del bus de datos (D_0-D_7) al nivel alto mediante el uso de resistencias elevadoras, de manera que se utilice el vector de interrupción FFH para la cadena. Utilice cualquier vector de interrupción para responder a una cadena tipo margarita. En el circuito, cualquiera de las cuatro salidas INTR de los dos dispositivos 82C55 harán que la terminal INTR del microprocesador se vaya al nivel alto y se solicitará una interrupción.

Cuando la entrada INTR se va al nivel alto con una cadena tipo margarita, el hardware no da una indicación directa acerca de cuál dispositivo 82C55 o cuál salida INTR ocasionó la interrupción. La tarea de localizar cuál de las salidas INTR se activó depende del procedimiento de servicio de interrupciones, el cual debe sondear los dispositivos 82C55 para determinar cuál de las salidas produjo la interrupción.

FIGURA 12-14 Dos PIAs 82C55 conectados a las salidas INTR se encadenan en margarita para producir una señal INTR.



El ejemplo 12-7 muestra el procedimiento de servicio de interrupciones que responde a la petición de interrupción de la cadena tipo margarita. El procedimiento sondea cada dispositivo 82C55 y cada salida INTR para decidir cuál procedimiento de servicio de interrupciones utilizará.

EJEMPLO 12-7

;Un procedimiento que da servicio a la interrupción de la cadena tipo margarita
;de la figura 12-14.

```
C1      EQU    504H          ;primer 82C55
C2      EQU    604H          ;segundo 82C55
MASC1  EQU    1             ;INTRB
MASC2  EQU    8             ;INTRA

SONDEA PROC  FAR   USES EAX EDX
          MOV     DX,C1          ;direcciona el primer 82C55
          IN      AL,DX
          TEST   AL,MASC1         ;evalúa INTRB
          .IF !ZERO?
          ;aquí va el software de interrupción NIVEL 1
```

```

.ENDIF
TEST AL,MASC2 ;evalúa INTRA
.IF !ZERO?

;aquí va el software de interrupción NIVEL 2

.ENDIF
MOV DX,C2 ;direcciona el segundo 82C55
TEST AL,MASC1 ;evalúa INTRB
.IF !ZERO?

;aquí va el software de interrupción NIVEL 3

.ENDIF

;aquí va el software de interrupción NIVEL 4

SONDEA ENDP

```

12-4**CONTROLADOR DE INTERRUPCIONES PROGRAMABLE 8259A**

Este controlador de interrupciones programable (PIC) agrega ocho interrupciones codificadas de prioridad vectorizada al microprocesador. Este controlador puede expandirse, sin necesidad de hardware adicional, para aceptar hasta 64 peticiones de interrupción. Para esta expansión se requiere un 8259A maestro y ocho 8259A esclavos. En los conjuntos de chips más recientes de Intel y otros fabricantes aún se utiliza un par de estos controladores, los cuales se programan de la misma forma que mostraremos aquí.

Descripción general del 8259A

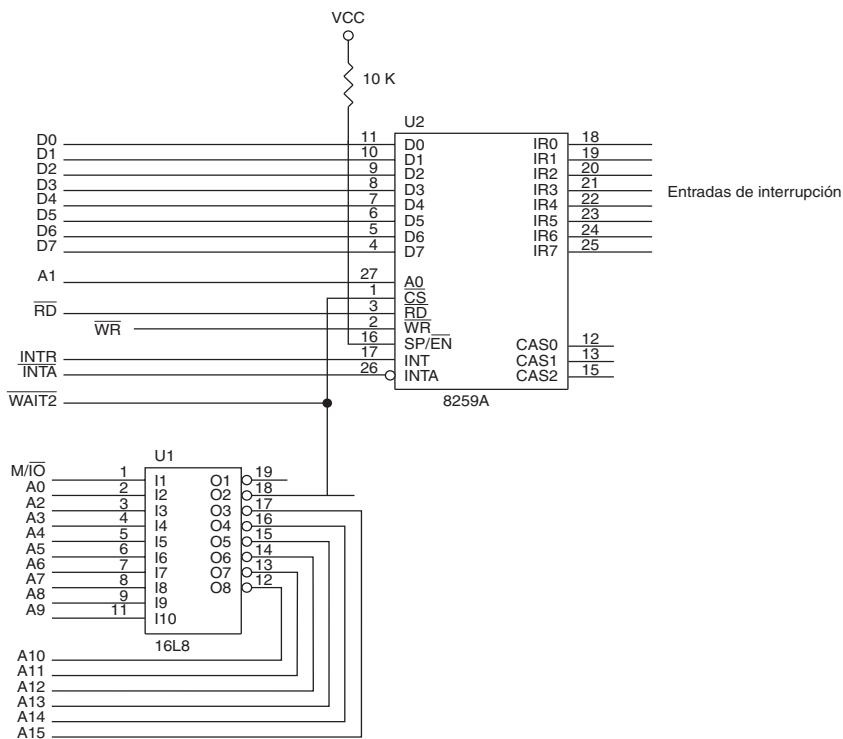
La figura 12-15 muestra el diagrama de terminales del 8259A. Este controlador se conecta con facilidad al microprocesador ya que todas sus terminales son conexiones directas excepto la terminal \overline{CS} , que debe decodificarse, y la terminal \overline{WR} , la cual debe tener un pulso de escritura en banco de E/S. A continuación se despliega una descripción de cada terminal del 8259A:

- | | |
|--------------------------------------|--|
| D₀-D₇ | Las conexiones de datos bidireccionales por lo general se conectan al bus de datos en el microprocesador. |
| IR₀-IR₇ | Las entradas de petición de interrupción se utilizan para solicitar una interrupción y para conectarse a un esclavo en un sistema con varios controladores 8259A. |
| WR | La entrada de escritura se conecta a la señal de estrobo de escritura (\overline{IOWC}) en el microprocesador. |
| RD | La entrada de lectura se conecta a la señal \overline{IORC} . |

FIGURA 12-15
Diagrama de terminales
del controlador de
interrupciones programable
(PIC) 8259A.

8259A		
11	D0	IR0
10	D1	IR1
9	D2	IR2
8	D3	IR3
7	D4	IR4
6	D5	IR5
5	D6	IR6
4	D7	IR7
27	A0	
1	<u>CS</u>	
3	<u>RD</u>	
2	<u>WR</u>	
16	SP/EN	CAS0
17	INT	CAS1
26	INTA	CAS2

FIGURA 12-16 Interfaz entre un 8259A y el microprocesador 8086.



- INT** La **salida de interrupción** se conecta a la terminal INTR en el microprocesador que proviene del controlador maestro, y se conecta a una terminal IR maestra en un esclavo.
- INTA** **Reconocimiento de interrupción** es una entrada que se conecta a la señal INTA en el sistema. En un sistema con controlador maestro y controladores esclavos sólo se conecta la señal INTA maestra.
- A₀** La **entrada de dirección A₀** selecciona distintas palabras de comando dentro del 8259A.
- CS** **Selección de chip** habilita el 8259A para la programación y el control.
- SP/EN** **Programa esclavo/habilita búfer** es una terminal de doble función. Cuando el 8259A se encuentra en modo con búfer, esta terminal es una salida que controla los receptores del bus de datos en un sistema grande basado en microprocesador. Cuando el 8259A no se encuentra en el modo con búfer, esta terminal programa el dispositivo como maestro (1) o como esclavo (0).
- CAS₀-CAS₂** Las **líneas en cascada** se utilizan como salidas del controlador maestro para los controladores esclavos, para conectar en cascada varios 8259A en un sistema.

Conexión de un solo 8259A

La figura 12-16 muestra un solo 8259A conectado al microprocesador. Aquí, la terminal SP/EN se lleva al nivel alto para indicar que es un controlador maestro. El 8259A se decodifica en los puertos de E/S 0400H y 0401H mediante el PLD (no se muestra el programa). Al igual que otros periféricos que vimos en el capítulo 11, el 8259A requiere cuatro estados de espera para funcionar de manera correcta con un 80386SX de 16 MHz, y requiere más estados de espera para otras versiones de la familia de microprocesadores Intel.

Conexión en cascada de varios controladores 8259A

Observe en la figura 12-17 dos controladores 8259A conectados al microprocesador de una forma que se utiliza a menudo en la computadora estilo ATX, la cual tiene dos controladores 8259A para las

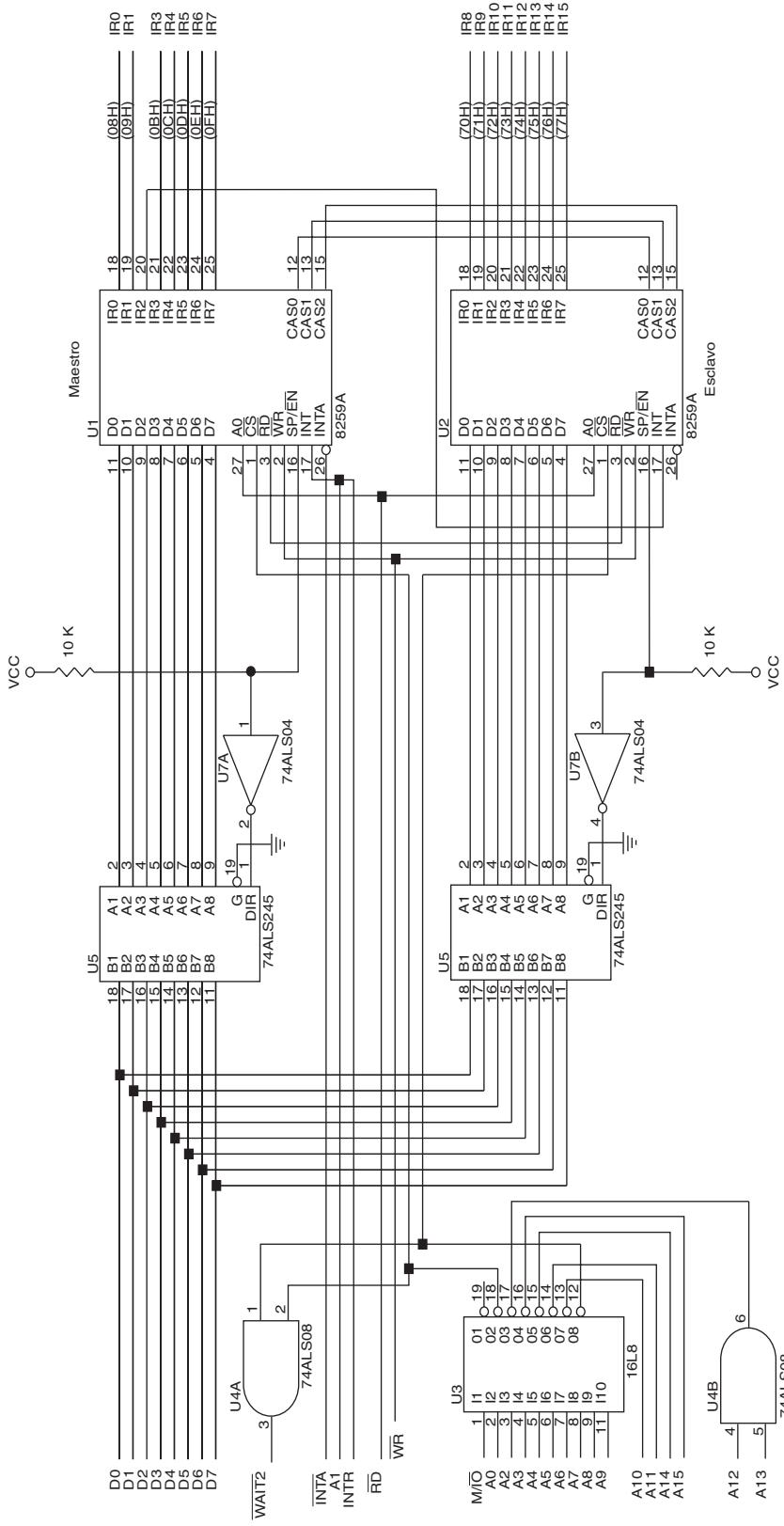


FIGURA 12-17 Dos controladores 8259A conectados al 8259A en los puertos de E/S 0300H y 0302H para el controlador maestro, y en los puertos de E/S 0304H y 0306H para el controlador esclavo.

interrupciones. Las computadoras estilo XT o PC utilizan un solo controlador 8259A en los vectores de interrupción 08H-0FH. La computadora estilo ATX utiliza el vector de interrupción 0AH como una entrada en cascada de un segundo controlador 8259A ubicado en los vectores del 70H al 77H. El apéndice A contiene una tabla que muestra las funciones de todos los vectores de interrupción utilizados.

Este circuito utiliza los vectores 08H-0FH y los puertos de E/S 0300H y 0302H para U1, el controlador maestro, y los vectores 70H-77H y los puertos de E/S 0304H y 0306H para U2, el controlador esclavo. Observe que también incluimos búferes en el bus de datos para ilustrar el uso de la terminal SP/EN en el 8259A. Estos búferes se utilizan sólo en sistemas muy grandes que tienen muchos dispositivos conectados a sus conexiones del bus de datos. En la práctica es raro encontrar estos búferes.

Programación del 8259A

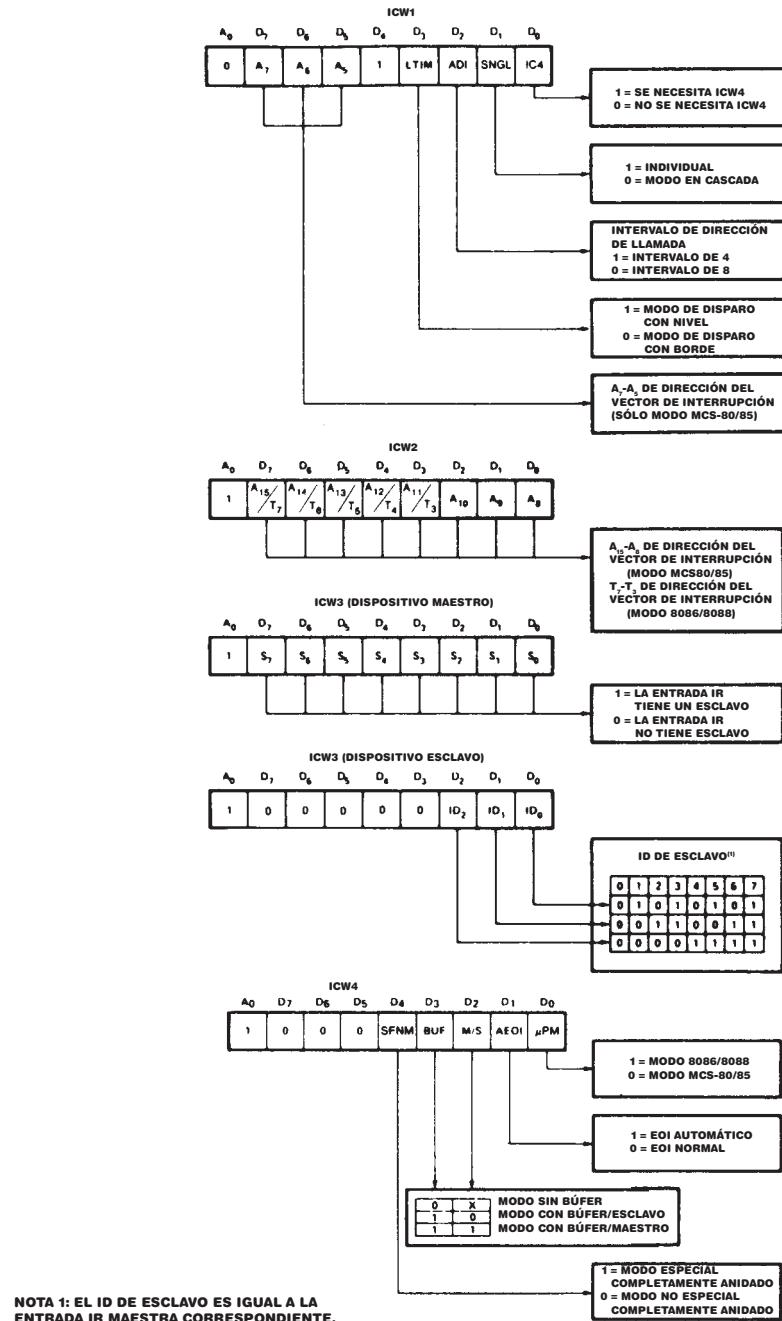
El controlador 8259A se programa mediante palabras de comandos de inicialización y de operación. Las **palabras de comandos de inicialización** (ICW) se programan antes de que el 8259A pueda funcionar en el sistema y dictan la operación básica de este controlador. Las **palabras de comandos de operación** (OCW) se programan durante el curso normal de operación. Las OCW controlan la operación del 8259A.

Palabras de comandos de inicialización. Existen cuatro palabras de comandos de inicialización (ICW) para el 8259A, las cuales se seleccionan cuando la terminal A₀ es un uno lógico. Cuando aplique energía al 8259A, envíe ICW₁, ICW₂ e ICW₄. Programe el 8259A en modo de cascada mediante ICW₁, y también el ICW₃. Por lo tanto, si se utiliza un solo 8259A en un sistema, programe ICW₁, ICW₂ e ICW₄. Si utiliza el modo en cascada en un sistema, entonces programe las cuatro ICW.

En la figura 12-18 podrá consultar el formato de las cuatro ICW. A continuación se muestra una descripción de cada ICW:

- ICW₁** Programa la operación básica del 8259A. Para programar esta ICW para la operación con los microprocesadores del 8086 al Pentium 4, coloque un 1 lógico en el bit IC₄. Los valores de los bits AD₁, A₇, A₆ y A₅ no importan para la operación del microprocesador y sólo se aplican al 8259A cuando se utiliza con un microprocesador 8085 de 8 bits (el cual no se cubre en este libro). La ICW selecciona la operación individual o en cascada mediante la programación del bit SNGL. Si selecciona la operación en cascada, también programe ICW₃. El bit LTIM determina si las entradas de petición de interrupción se disparan con el borde positivo o con el nivel.
- ICW₂** Selecciona el número de vector que se utiliza con las entradas de petición de interrupción. Por ejemplo, si decide programar el 8259A para que funcione en las posiciones de vector 08H-0FH, coloque un 08H en esta palabra de comando. De igual forma, si decide programar el 8259A para los vectores 70H-77H, coloque un 70H en esta ICW.
- ICW₃** Sólo se utiliza cuando ICW₁ indica que el sistema se opera en el modo en cascada. Esta ICW indica en dónde se conecta el controlador esclavo al maestro. Por ejemplo, en la figura 12-18 conectamos un esclavo a IR₂. Para programar ICW₃ para esta conexión, tanto en el controlador maestro como en el esclavo, coloque un 04H en ICW₃. Suponga que tiene dos esclavos conectados a un maestro mediante el uso de IR₀ e IR₁. El maestro se programa con una ICW₃ de 03H; un esclavo se programa con una ICW₃ de 01H y la otra con una ICW₃ de 02H.
- ICW₄** Se programa para utilizarse con los microprocesadores del 8086 al Pentium 4, pero no se programa en un sistema que funcione con el microprocesador 8085. El bit de más a la derecha debe ser un 1 lógico para seleccionar la operación con los microprocesadores del 8086 al Pentium 4, y el resto de los bits se programa de la siguiente manera: SFNM: si se coloca un 1 lógico en este bit, se selecciona el modo de operación especial de anidación completa para el 8259A. Esto permite que la petición de interrupción de mayor prioridad de un esclavo sea reconocida por el maestro, mientras éste procesa otra interrupción de un esclavo. Por lo general, sólo se procesa una petición de interrupción a la vez, y las demás se ignoran hasta que termine el proceso.

FIGURA 12-18 Las palabras de comando de inicialización del 8259A (ICWs). (Cortesía de Intel Corporation.)



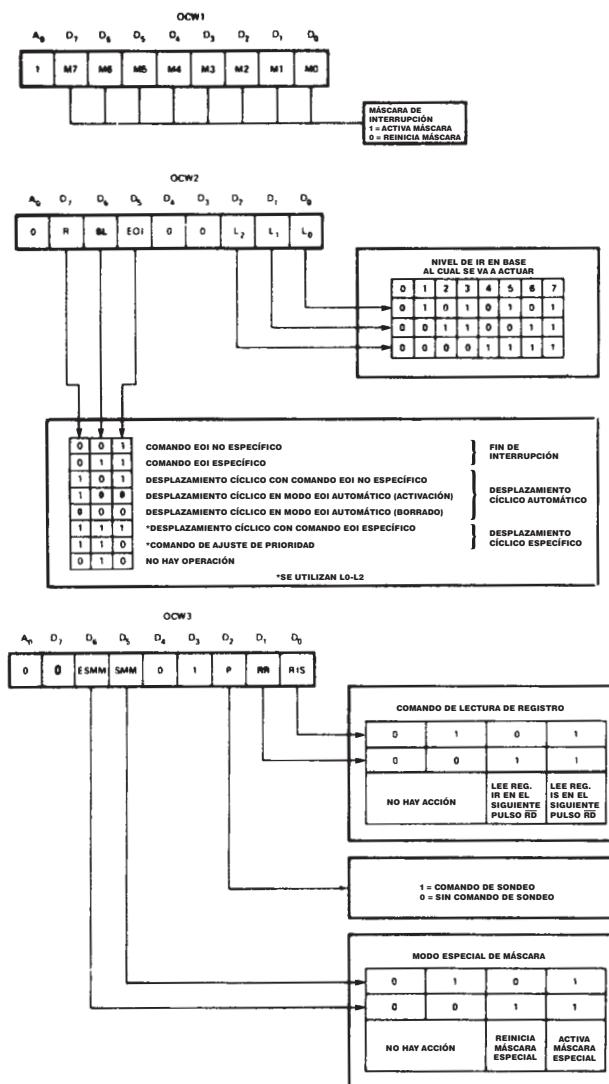
BUF y M/S: los bits de búfer y maestro esclavo se utilizan en conjunto para seleccionar la operación con búfer o sin búfer para el 8259A como maestro o como esclavo.

AE0I: selecciona el término automático o normal de la interrupción (lo cual veremos más adelante con las palabras de comandos de operación). Los comandos EOI de OCW₂ se utilizan sólo si ICW₄ no selecciona el modo AE0I. Si se selecciona este modo, la interrupción reinicia de manera automática el bit de petición de interrupción y no modifica la prioridad. Éste es el modo preferido de operación para el 8259A y reduce la longitud del procedimiento de servicio de interrupciones.

Palabras de comandos de operación. Las palabras de comandos de operación (OCW) se utilizan para dirigir la operación del 8259A, una vez que se programa con el ICW. Las OCW se seleccionan cuando la terminal A₀ se encuentra en el nivel de 0 lógico, excepto OCW₁, que se selecciona cuando A₀ es un 1 lógico. La figura 12-19 muestra los patrones de bits binarios para las tres palabras de comandos de operación del 8259A. A continuación se despliega una lista que describe la función de cada OCW:

- OCW₁** Se utiliza para activar y leer el registro de máscara de interrupción. Cuando se activa un bit de máscara, éste desactiva (enmascara) la entrada de interrupción correspondiente. El registro de máscara se lee al mismo tiempo que OCW₁. Como el estado de los bits de máscara se desconoce cuando el 8259A se inicializa por primera vez, OCW₁ debe programarse después de programar la ICW en la inicialización.
- OCW₂** Se programa sólo cuando no se selecciona el modo AEOI para el 8259A. En este caso, esta OCW selecciona la manera en que el 8259A responde a una interrupción. Los modos son los siguientes:

FIGURA 12-19 Las palabras de comandos de operación del 8259A (OCWs). (Cortesía de Intel Corporation.)



Fin de interrupción no específico: un comando que envía el procedimiento de servicio de interrupciones para indicar el término de la interrupción. El 8259A determina en forma automática cuál nivel de interrupción estaba activo y reinicia el bit correcto del registro del estado de interrupciones. Al reiniciar el bit de estado la interrupción puede actuar de nuevo, o se puede producir una interrupción de nivel más bajo.

Fin de interrupción específica: un comando que permite reiniciar una petición de interrupción específica. La posición exacta se determina con los bits L₂-L₀ de OCW₂.

Desplazamiento cíclico con EOI no específico: un comando que funciona de la misma forma que el comando Fin de interrupción específica, sólo que realiza un desplazamiento cíclico en las prioridades de interrupción después de reiniciar el bit de registro de estado de interrupciones. El nivel que reinicie este comando se convertirá en la interrupción de menor prioridad. Por ejemplo, si se acaba de dar servicio a IR₄, esta entrada se convertirá en la interrupción de menor prioridad e IR₅ se convertirá en la entrada con mayor prioridad.

Desplazamiento cíclico con EOI automático: un comando que selecciona el EOI automático con desplazamiento cíclico de prioridades. Este comando sólo debe enviarse una vez al 8259A si se desea este modo. Si debe desactivarse este modo, utilice el comando para borrar.

Desplazamiento cíclico en EOI específico: funciona como el EOI específico, sólo que selecciona el desplazamiento cíclico de prioridades.

Ajuste de prioridad: permite al programador establecer la entrada de interrupción de menor prioridad mediante el uso de los bits L₂-L₀.

OCW₃

Selecciona el registro a leer, la operación del registro de máscara especial y el comando de sondeo. Si se selecciona el sondeo debe activarse el bit P y luego se debe enviar al 8259A. La siguiente operación de lectura leerá la palabra de sondeo. Los tres bits de más a la derecha de la palabra de sondeo indican la petición de interrupción activa con la mayor prioridad. El bit de más a la izquierda indica si hay una interrupción y debe verificarse para determinar si los tres bits de más a la derecha contienen información válida.

Registro de estado. Hay tres registros de estado que pueden leerse en el 8259A: el registro de petición de interrupción (IRR), el registro de servicio de entrada (ISR) y el registro de máscara de interrupción (IMR). (Consulte en la figura 12-20 los tres registros de estado: todos tienen la misma configuración de bits.) El IRR es un registro de 8 bits que indica cuáles entradas de petición de interrupción están activas. El ISR es un registro de 8 bits que contiene el nivel de la interrupción a la que se está dando servicio. El IMR es un registro de 8 bits que almacena los bits de máscara de interrupción e indica cuáles interrupciones están enmascaradas.

Los registros IRR e ISR se leen mediante la programación de OCW₃ y el registro IMR se lee a través de OCW₁. Para leer el IMR, A₀ = 1; para leer el IRR o el ISR, A₀ = 0. Cuando A₀ = 0, las posiciones de bit D₀ y D₁ de OCW₃ seleccionan cuál de los dos registros se lee (IRR o ISR).

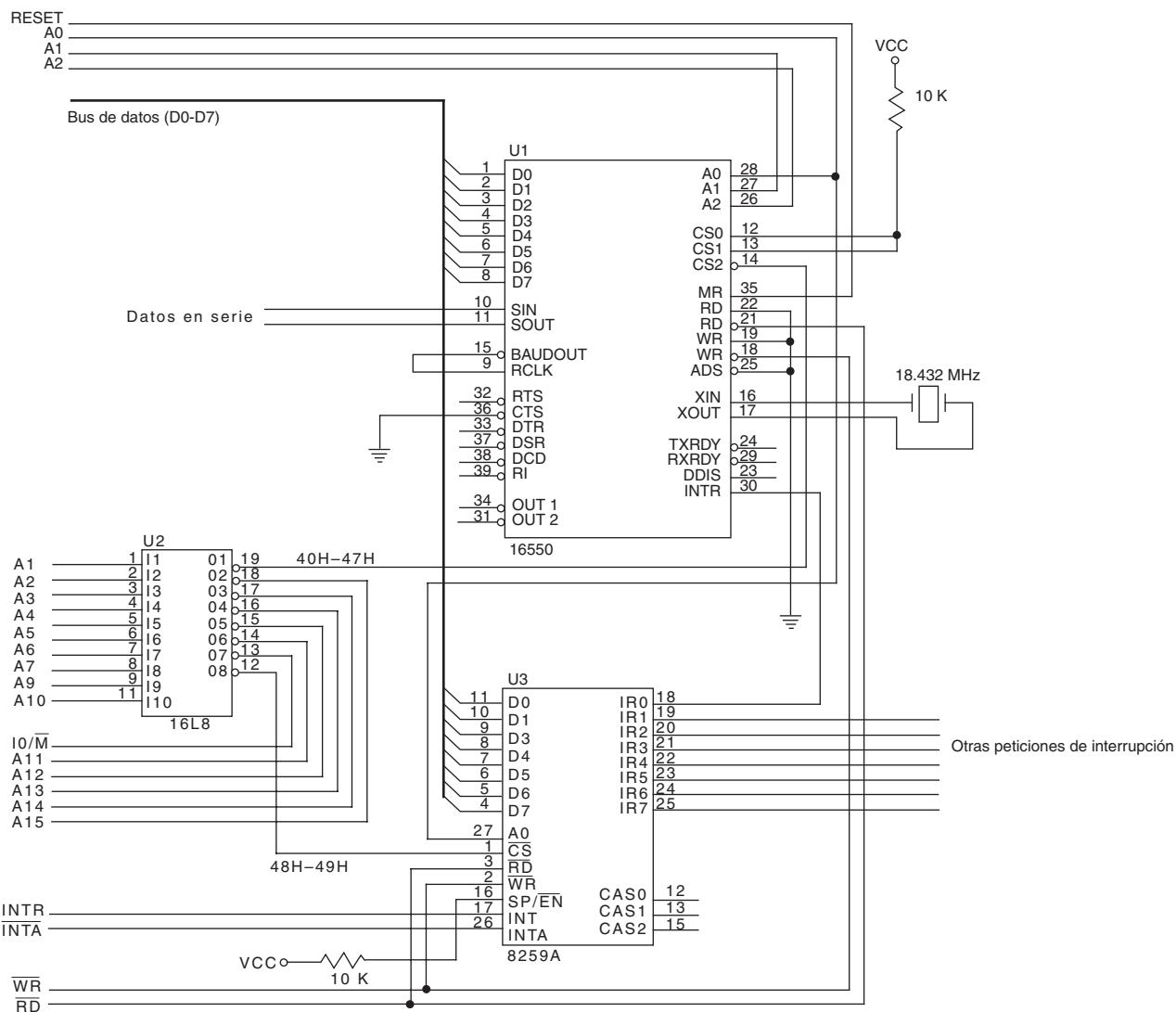


FIGURA 12-20 El registro de servicio de entrada (ISR) del 8259A. (a) Antes de aceptar IR₄ y (b) después de aceptar IR₄. (Cortesía de Intel Corporation.)

Ejemplo de programación del 8259A

Observe en la figura 12-21 el controlador de interrupciones programable 8259A conectado a un controlador de comunicaciones programable 16550. En este circuito, la terminal INTR que proviene del 16550 se conecta a la entrada IR₀ de petición de interrupción del controlador de interrupciones programable. Se produce una IR₀ cada vez que (1) el transmisor está listo para enviar otro carácter, (2) el receptor ha recibido un carácter, (3) se detecta un error al recibir los datos y (4) se produce una interrupción del módem. Observe que el 16550 se decodifica en los puertos de E/S 40H y 47H, y el 8259A se decodifica en los puertos de E/S de 8 bits 48H y 49H. Ambos dispositivos se conectan al bus de datos de un microprocesador 8088.

Software de inicialización. La primera porción del software para este sistema debe programar tanto el 16550 como el 8259A, y después debe habilitar la terminal INTR en el 8088 para que puedan producirse interrupciones. El ejemplo 12-8 muestra el software requerido para programar ambos dispositivos y habilitar INTR. Este software utiliza dos FIFO de memoria que almacenan datos para el transmisor y para el receptor. Cada FIFO de memoria es de 16 Kbytes y se direcciona mediante un par de apuntadores (entrada y salida).



EJEMPLO 12-8

```

;Software de inicialización para el 16550 y el 8259A
;del circuito de la figura 12-21

PIC1    EQU    48H          ;controla 8259A, A0 = 0
PIC2    EQU    49H          ;controla 8259A, A0 = 1
ICW1    EQU    1BH          ;8259A, ICW1
ICW2    EQU    80H          ;8259A, ICW2
ICW4    EQU    3             ;8259A, ICW4
OCW1    EQU    0FEH         ;8259A, OCW1
LINEA   EQU    43H         ;16550, registro de línea
LSB     EQU    40H         ;16550, divisor de Baudios LSB
MSB     EQU    41H         ;16550, divisor de Baudios MSB
FIFO    EQU    42H         ;16550, registro FIFO
ITR     EQU    41H         ;16550, registro de interrupciones

INIC    PROC   NEAR

;
;configuración del 16550
;
        MOV    AL,10001010B      ;habilita el divisor de velocidad en Baudios
        OUT    LINEA,AL

        MOV    AL,120            ;programa 9600 Baudios
        OUT    LSB,AL
        MOV    AL,0
        OUT    MSB,AL

        MOV    AL,00001010B      ;programa 7 bits de datos, paridad
        OUT    LINEA,AL          ;impar, 1 bit de paro

        MOV    AL,00000111B      ;habilita el transmisor y
        OUT    FIFO,AL          ;el receptor
;
;programación del 8259A
;
        MOV    AL,ICW1           ;programa la ICW1
        OUT    PIC1,AL
        MOV    AL,ICW2           ;programa la ICW2
        OUT    PIC2,AL
        MOV    AL,ICW4           ;programa la ICW4
        OUT    PIC2,AL
        MOV    AL,OCW1           ;programa la OCW1
        OUT    PIC2,AL
        STI                  ;habilita la terminal INTR
;
;habilitación de las interrupciones del 16550
;
        MOV    AL,5
        OUT    ITR,AL            ;habilita las interrupciones
        RET

INIC    ENDP

```

La primera porción del procedimiento (INIT) programa el UART 16550 para operar con siete bits de datos, paridad impar, un bit de paro y una velocidad de Baudios de 9600. El registro de control FIFO también habilita el transmisor y el receptor.

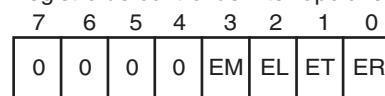
La segunda parte del procedimiento programa el 8259A con sus tres ICW y una OCW. El 8259A se configura de manera que funcione en los vectores de interrupción 80H-87H y que opere con EOI automático. La OCW habilita la interrupción para el UART 16550. La terminal INTR del microprocesador también se habilita mediante el uso de la instrucción STI.

La parte final del software habilita las interrupciones de error y al receptor del UART 16550, a través del registro de control de interrupciones. La interrupción del transmisor no se habilita sino hasta que haya datos disponibles para su transmisión. En la figura 12-22 podrá ver el contenido del registro de

FIGURA 12-22

El registro de control de interrupciones del 16550.

Registro de control de interrupciones



Habilita interrupción del receptor
0 = deshabilitada
1 = habilitada

Habilita interrupción del transmisor
0 = deshabilitada
1 = habilitada

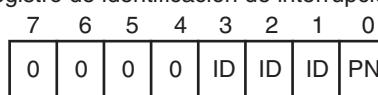
Habilita interrupción de línea
0 = deshabilitada
1 = habilitada

Habilita interrupción de módem
0 = deshabilitada
1 = habilitada

FIGURA 12-23

El registro de identificación de interrupciones del 16550.

Registro de identificación de interrupciones



Habilita interrupción del receptor
0 = deshabilitada
1 = habilitada

Habilita interrupción del transmisor
0 = deshabilitada
1 = habilitada

control de interrupciones del UART 16550. Observe que el registro de control puede habilitar o deshabilitar el receptor, el transmisor, el estado de línea (error) y las interrupciones del módem.

Manejo de la petición de interrupción del UART 16550. Como el 16550 sólo genera una petición de interrupción para varias interrupciones, el manejador de interrupciones debe sondear el 16550 para determinar qué tipo de interrupción se ha producido. Esto se logra mediante un análisis del registro de identificación de interrupciones (vea la figura 12-23). Observe que el registro de identificación de interrupciones (sólo lectura) comparte el mismo puerto de E/S que el registro de control FIFO (sólo escritura).

El registro de identificación de interrupciones indica si hay una interrupción pendiente, el tipo de interrupción y si están habilitadas las memorias FIFO del transmisor y del receptor. En la tabla 12-2 podrá consultar el contenido de los bits de control de interrupciones.

El procedimiento de servicio de interrupciones debe examinar el contenido del registro de identificación de interrupciones para determinar qué evento produjo la interrupción y pasar el control al procedimiento apropiado para el evento. El ejemplo 12-9 muestra la primera parte de un manejador de interrupciones que pasa el control a RECP para una interrupción de datos del receptor, a TRANS para una interrupción de datos del transmisor y a ERR para una interrupción de error de estado de línea. En este ejemplo no se evalúa el estado del módem.

EJEMPLO 12-9

```
;Manejador de interrupciones para el UART 16550 de la figura 12-21
INT80      PROC    FAR USES AX BX DI SI
            IN AL,42H          ;lee el ID de la interrupción
            .IF AL == 6
                        ;maneja interrupción de error del receptor
            .ELSEIF AL == 2
                        ;maneja interrupción de transmisor vacío
```

TABLA 12-2 Los bits de control de interrupciones del 16550.

<i>Bit 3</i>	<i>Bit 2</i>	<i>Bit 1</i>	<i>Bit 0</i>	<i>Prioridad</i>	<i>Tipo</i>	<i>Control de reinicio</i>
0	0	0	1	—	No hay interrupción.	—
0	1	1	0	1	Error del receptor (paridad, trama, saturación o interrupción).	Se reinicia mediante una lectura del registro.
0	1	0	0	2	Datos del receptor disponibles.	Se reinicia mediante la lectura de los datos.
1	1	0	0	2	Tiempo agotado de carácter, no se ha extraído nada del receptor FIFO durante al menos cuatro tiempos de carácter.	Se reinicia mediante la lectura de los datos.
0	0	1	0	3	Transmisor vacío.	Se reinicia mediante la escritura en el transmisor.
0	0	0	0	4	Estado del módem.	Se reinicia mediante la lectura del estado del módem.

```

JMP    TRAN ;ejemplo 12-13
ELSEIF AL == 4
;maneja interrupción de receptor listo
JMP    RECB ;ejemplo 12-11
ENDIF
IRET
INT80
ENDP

```

Para recibir datos del 16550 se requieren dos procedimientos. Uno lee el registro de datos del 16550 cada vez que la terminal INTR solicita una interrupción y lo almacena en la FIFO de la memoria. El otro procedimiento lee datos de la FIFO de la memoria desde el programa principal.

El ejemplo 12-10 muestra el procedimiento que se utiliza para leer datos de la FIFO de la memoria desde el programa principal. Este procedimiento supone que los apuntadores (IIN y IOUT) se inicializan en el diálogo de comienzo para el sistema (el cual no se muestra). Cuando el procedimiento LEE regresa, AL contiene un carácter leído de la FIFO de la memoria. Si la FIFO está vacía, el procedimiento regresa con el bit de bandera de acarreo puesto en 1 lógico. Si AL contiene un carácter válido, el bit de bandera de acarreo se borra al regresar de LEE.

Observe cómo se vuelve a utilizar la FIFO al cambiar la dirección desde la parte superior de la FIFO hacia el fondo, cada vez que su valor sea mayor al inicio de la FIFO más 16 K. Observe que las interrupciones se habilitan al final de este procedimiento, en caso de que el procedimiento de interrupción RECB las deshabilite mediante la condición “FIFO de memoria llena”.

EJEMPLO 12-10

```

;Un procedimiento que lee un carácter de la FIFO
;y lo regresa en AL. Si la FIFO está vacía, el procedimiento
;regresa con el bit de acarreo = 1.

```

```

LEEC    PROC    NEAR USES BX DX
        MOV     DI,IOUT          ;obtiene el apuntador
        MOV     BX,IIN
        .IF    BX == DI          ;si está vacío
              STC               ;activa el acarreo

```

```

.ELSE           ;si no está vacío
    MOV AL,ES:[DI]      ;obtiene datos
    INC DI              ;incrementa el apuntador
    .IF DI == OFFSET FIFO+16*1024
        MOV DI,OFFSET FIFO
    .ENDIF
    MOV IOUT,DI
    CLC
.ENDIF
PUSHF           ;habilita el receptor
IN AL,41H
OR AL,5
OUT 41H,AL
POPF
RET

LEEC    ENDP

```

El ejemplo 12-11 muestra el procedimiento de servicio de interrupciones RECB que se llama cada vez que el 16550 recibe un carácter para el microprocesador. En este ejemplo, la interrupción utiliza el número de tipo de vector 80H, el cual debe direccionar el manejador de interrupciones del ejemplo 12-9. Cada vez que se produce esta interrupción, el manejador de interrupciones accede al procedimiento RECB para leer un carácter del 16550. El procedimiento RECB almacena el carácter en la FIFO de la memoria. Si la FIFO está llena, el registro de control de interrupciones dentro del 16550 deshabilita la interrupción del receptor. Esto puede ocasionar la pérdida de datos, pero por lo menos no hará que la interrupción sobrescriba datos válidos que ya estén almacenados en la FIFO de la memoria. Cualquier condición de error detectada por el 8251A almacena un ? (3FH) en la FIFO de la memoria. Los errores se detectan mediante la porción correspondiente a ERR del manejador de interrupciones (el cual no se muestra).

EJEMPLO 12-11

;Porción correspondiente a RECB del manejador de interrupciones del ejemplo 12-9

```

RECB:
    MOV BX,IOUT          ;obtiene los apuntadores
    MOV DI,IIN
    MOV SI,DI
    INC SI
    .IF SI == OFFSET FIFO+16*1024
        MOV SI,OFFSET FIFO
    .ENDIF
    .IF SI == BX          ;si la FIFO está llena
        IN AL,41H          ;deshabilita el receptor
        AND AL,0FAH
        OUT 41H,AL
    .ENDIF
    IN AL,40H             ;lee los datos
    STOSB
    MOV IIN,SI
    MOV AL,20H             ;comando EOI del 8259A
    OUT 49H,AL
    IRET

```

Transmisión de datos al 16550. Los datos se transmiten al 16550 de una forma muy parecida a como se reciben, sólo que el procedimiento de servicio de interrupciones extrae los datos transmitidos de una segunda FIFO de memoria de 16 Kbytes.

El ejemplo 12-12 muestra el procedimiento que llena la FIFO de salida. Es similar al procedimiento que se muestra en el ejemplo 12-10, sólo que determina si la FIFO está llena en vez de vacía.

EJEMPLO 12-12

;Un procedimiento que coloca datos en la FIFO de memoria para transmitirlos mediante la interrupción del transmisor. AL = al

```

;carácter transmitido.
GUARDC PROC NEAR USES BX DI SI

    MOV SI,OIN           ;carga los apuntadores
    MOV BX,OOUT
    MOV DI,SI
    INC SI
    .IF SI == OFFSET OFIFO+16*1024
        MOV SI,OFFSET OFIFO
    .ENDIF
    IF BX == SI          ;si OFIFO está llena
        STC
    .ELSE
        STOSB
        MOV OIN,SI
        CLC
    .ENDIF
    PUSHF
    IN AL,41H            ;habilita el transmisor
    OR AL,1
    OUT 41H,AL
    RET

GUARDC ENDP

```

El ejemplo 12-13 muestra la subrutina de servicio de interrupciones para el transmisor del UART 16550. Este procedimiento es una continuación del manejador de interrupciones que se presentó en el ejemplo 12-9 y es similar al procedimiento RECB del ejemplo 12-11, sólo que determina si la FIFO está vacía en vez de llena. Observe que no incluimos un procedimiento de servicio de interrupciones para la interrupción por interrupción (break) ni para los errores.

EJEMPLO 12-13

```

;Servicio de interrupción para el transmisor del 16550

TRAN:
    MOV BX,OIN           ;carga los apuntadores
    MOV DI,OOUT
    .IF BX == DI          ;si está vacío
        IN AL,41H
        AND AL,OFDH        ;deshabilita el transmisor
        OUT 41H,AL
    .ELSE                  ;si no está vacío
        MOV AL,ES:[DI]
        OUT 40H,AL          ;envía datos
        INC DI
        .IF DI == OFFSET OFIFO+16*1024
            MOV DI,OFFSET OFIFO
        .ENDIF
        MOV OFIFO,DI
    .ENDIF
    MOV AL,20H            ;envía EOI al 8259A
    OUT 49H,AL
    IRET

```

El 16550 también contiene un registro residual, el cual es un registro de propósito general que puede utilizarse de cualquier forma que el programador considere necesaria. El 16550 también contiene un registro de control del módem y un registro de estado del módem. Estos registros permiten que el módem produzca interrupciones y que controle la operación del 16550. En la figura 12-24 podrá consultar el contenido del registro de estado del módem y del registro de control del módem.

El registro de control del módem utiliza las posiciones de bit 0-3 para controlar varias terminales en el 16550. La posición de bit 4 habilita la prueba de bucle interna para fines de prueba. El registro de estado del módem permite evaluar el estado de las terminales del módem; también permite verificar cambios en las terminales del módem o, en el caso de \overline{RI} , un borde de caída.

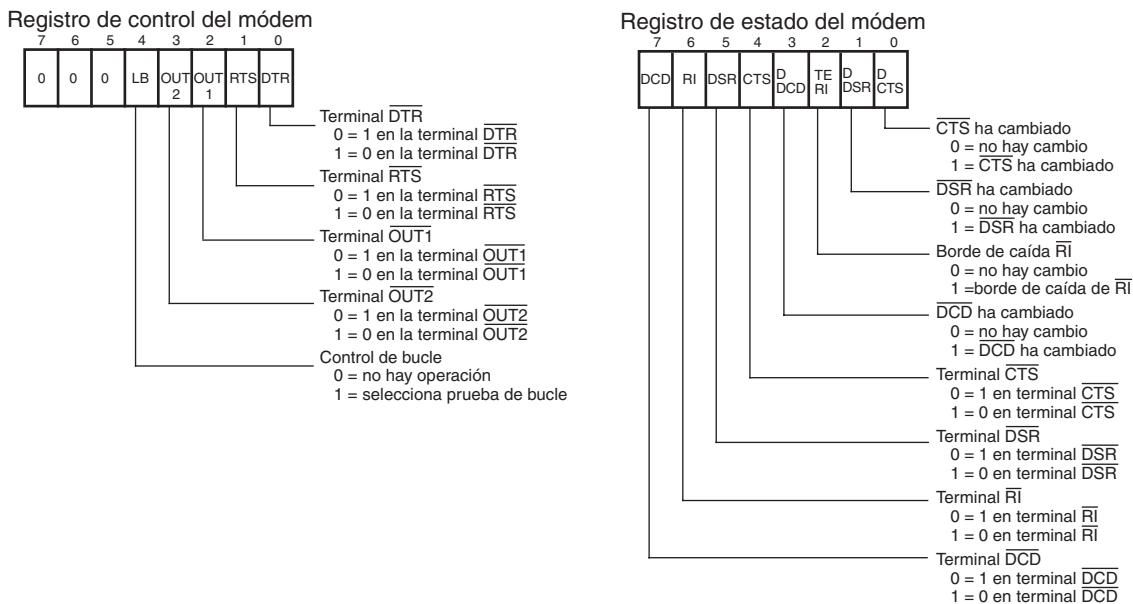


FIGURA 12-24 Los registros de control del módem y estado del módem del 16550.

La figura 12-25 muestra el UART 16550 conectado a una interfaz RS-232C que se utiliza a menudo para controlar un módem. En esta interfaz se incluyen los circuitos del controlador de línea y del receptor que se utilizan para convertir los niveles TTL en el 16550 a los niveles de la interfaz RS-232C. Observe que los niveles de la RS-232C, por lo general, son de +12 V para un 0 lógico y -12 V para un 1 lógico.

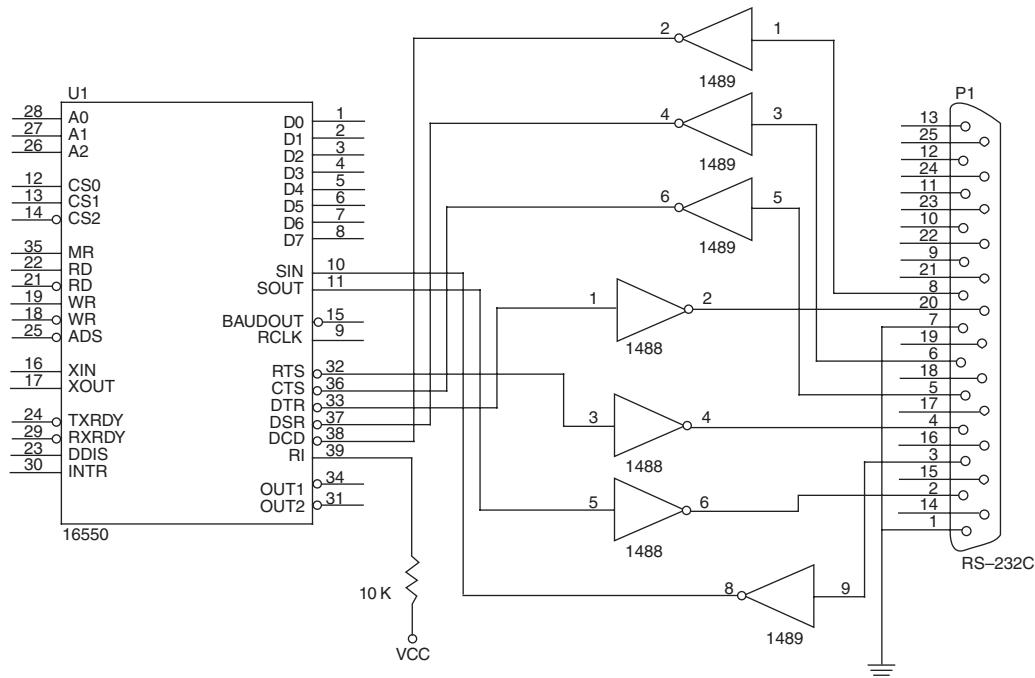


FIGURA 12-25 La interfaz RS-232C conectada con el 16550 mediante el uso de controladores de línea 1488 y receptores de línea 1489.

Para poder transmitir o recibir datos a través del módem se activa la terminal DTR (0 lógico) y después el UART espera a que la terminal DSR se vuelva 0 lógico desde el módem, lo cual indica que está listo. Una vez que se completa este protocolo de intercambio (handshake), el UART envía un 0 lógico al módem en la terminal RTS. Cuando el módem está listo, regresa la señal CTS (0 lógico) al UART. Ahora pueden comenzar las comunicaciones. La señal DCD del módem indica que éste ha detectado una portadora. También hay que evaluar la señal antes de que puedan comenzar las comunicaciones.

12-5

EJEMPLOS DE INTERRUPCIONES

En esta sección presentaremos un reloj en tiempo real y un teclado procesado por interrupciones como ejemplos de aplicaciones que utilizan interrupciones. Un reloj en tiempo real (RTC) lleva la hora en tiempo real; es decir, en horas y minutos. También se utiliza para retrasos de tiempo de precisión. El ejemplo que se muestra aquí lleva el tiempo en horas, minutos, segundos y 1/60 de segundo; para ello utiliza cuatro posiciones de memoria para almacenar la hora del día en BCD. El teclado procesado por interrupciones utiliza una interrupción periódica para explorar sus teclas.

Reloj en tiempo real

La figura 12-26 muestra un circuito simple que utiliza la línea de energía CA de 60 Hz para generar una señal periódica de petición de interrupción para la terminal de entrada de interrupción NMI. Aunque utilizamos una señal de la línea de energía CA en la que su frecuencia varía un poco de vez en cuando, es precisa durante un periodo de tiempo como lo exige la Comisión Federal de Comercio (FTC).

El circuito utiliza una señal de la línea de energía CA de 120 V, la cual está condicionada por un inversor de disparo Schmitt antes de aplicarse a la entrada de interrupción NMI. Debe asegurarse que la línea de energía esté conectada a la tierra del sistema en este diagrama esquemático. La conexión neutral (cable blanco) de la línea de energía es la terminal plana ancha. La terminal plana estrecha es el lado vivo (cable negro), o lado de 120 V CA de la línea.

El software para el reloj de tiempo real contiene un procedimiento de servicio de interrupciones que se llama 60 veces por segundo y un procedimiento que actualiza la cuenta almacenada en cuatro posiciones de memoria. El ejemplo 12-14 lista ambos procedimientos, junto con los cuatro bytes de memoria que se utilizan para almacenar la hora del día en BCD. Las posiciones de memoria para el TIEMPO se almacenan en alguna parte de la memoria del sistema, en la dirección de segmento (SEGMENTO) y en la dirección de desplazamiento TIEMPO, las cuales se cargan primero en el procedimiento TIEMPOP. La tabla de búsqueda (BUSCA) para los módulos o cada contador se almacena en el segmento de código, junto con el procedimiento.

EJEMPLO 12-14

```

TIEMPO  DB      ?          ;contador de 1/60 de seg (÷60)
        DB      ?          ;contador de segundos (÷60)
        DB      ?          ;contador de minutos (÷60)
        DB      ?          ;contador de horas (÷24)

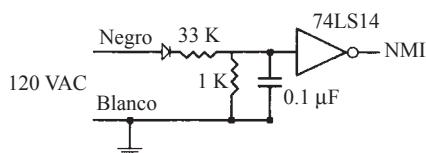
BUSCA   DB      60H, 60H, 60H, 24H

TIEMPOP PROC   FAR USES AX BX DS

        MOV  AX,SEGMENT           ;carga la dirección de segmento de TIEMPO
        MOV  DS,AX

        MOV  AX,SEGMENT
        MOV  DS,AX
    
```

FIGURA 12-26 Conversión de la línea de energía CA para una señal TTL de 60 Hz para la entrada NMI.



```

        MOV BX, 0           ; inicializa el apuntador
        .REPEAT             ; arranca el reloj
            MOV AL, DS:TIEMPO[BX]
            ADD AL, 1         ; incrementa la cuenta
            DAA               ; ajusta para BCD
            .IF AL == BYTE PTR CS:BUSCA[BX]
                MOV AL, 0
            .ENDIF
            MOV DS:TIEMPO[BX], AL
            INC BX
        .UNTIL AL != 0 || BX == 4
        IRET

TIEMPOP ENDP

```

Otra forma de manejar el tiempo es mediante el uso de un solo contador para almacenarlo en memoria y después determinar la hora actual mediante software. Por ejemplo, el tiempo puede almacenarse en un solo contador de 32 bits (hay 5,184,000 1/60 de seg en un día). En un contador tal como éste, una cuenta de 0 equivale a 12:00:00:00 AM y una cuenta de 5,183,999 equivale a 11:59:59 PM. El ejemplo 12-15 muestra el procedimiento de interrupción para este tipo de RTC, el cual requiere la menor cantidad de tiempo para ejecutarse.

EJEMPLO 12-15

```

TIEMPO      DD      ?           ; contador módulo 5,184,000
TIEMPOP     PROC   FAR USES EAX
    MOV AX, SEGMENT
    MOV DS, AX
    INC DS:TIEMPO
    .IF DS:TIEMPO == 5184000
        MOV DWORD PTR DS:TIEMPO, 0
    .ENDIF
    IRET
TIEMPOP ENDP

```

El software para convertir la cuenta del contador módulo 5,184,000 en horas, minutos y segundos aparece en el ejemplo 12-16. El procedimiento regresa con el número de horas (0-23) en BL, el número de minutos en BH y el número de segundos en AL. No se hizo el intento por extraer la cuenta de 1/60 de segundo.

EJEMPLO 12-16

```

;El tiempo se regresa como BL = horas, BH = minutos y AL = segundos
OBTIENET PROC NEAR USES ECX EDX
    MOV ECX, 216000          ; divide entre 216,000
    MOV EAX, TIEMPO
    SUB EDX, EDX             ; borra EDX
    DIV ECX                  ; obtiene las horas
    MOV BL, AL
    MOV EAX, EDX
    MOV ECX, 3600             ; divide entre 3600
    DIV ECX                  ; obtiene los minutos
    MOV BH, AL
    SUB EAX, EDX
    MOV ECX, 60                ; divide entre 60
    DIV ECX
    RET
OBTIENET ENDP

```

Suponga que se necesita un retraso de tiempo. Los retrasos de tiempo pueden lograrse mediante el uso del RTC del ejemplo 12-15 para cualquier cantidad, desde 1/60 de segundo hasta 24 horas. El ejemplo 12-17 muestra un procedimiento que utiliza el RTC para producir retrasos de tiempo del número de segundos que se pasan al procedimiento en el registro EAX. Esto puede ser desde 1 segundo hasta todo un día completo. Tiene una precisión de hasta 1/60 de segundo, que es la resolución del RTC.

EJEMPLO 12-17

```

SEG      PROC    NEAR USES EAX EDX
        MOV     EDX,60
        MUL     EDX           ;obtiene los segundos como cuenta de 1/60s
        ADD     EAX,TIEMPO   ;avanza el TIEMPO en EAX
        .IF    EAX >= 51840000
              SUB EAX,51840000
        .ENDIF
        .REPEAT          ;espera a que el TIEMPO se ponga al corriente
        .UNTIL   EAX == TIEMPO
        RET
SEG      ENDP

```

Teclado procesado por interrupciones

Este ejemplo explora las teclas de un teclado mediante una interrupción periódica. Cada vez que se produce la interrupción, el procedimiento de servicio de interrupciones comprueba una tecla o elimina los rebotes. Una vez que se detecta una tecla válida, el procedimiento de servicio de interrupciones almacena el código de tecla en una cola de teclado para que el sistema la lea después. La base para este sistema es una interrupción periódica que puede producirse mediante un temporizador, RTC u otro dispositivo en el sistema. La mayoría de los sistemas ya tienen una interrupción periódica para el reloj en tiempo real. En este ejemplo suponemos que la interrupción llama al procedimiento de servicio de interrupciones cada 10 ms o, si se utiliza el RTC con un reloj de 60 Hz, cada 16.7 ms.

La figura 12-27 muestra la interfaz entre el teclado y un 82C55. No muestra el temporizador ni cualquier otro circuito requerido para llamar a la interrupción una vez cada 10 ms o 16.7 ms. (En el software no se muestra la programación del 82C55.) El 82C55 debe programarse de manera que el puerto A sea un puerto de entrada, que el puerto B sea un puerto de salida y el software de inicialización debe almacenar un 00H en el puerto B. Esta interfaz utiliza una cantidad de memoria que se almacena en el segmento de código para una cola y unos cuantos bytes que llevan el registro de la exploración del teclado. El ejemplo 12-18 muestra el procedimiento de servicio de interrupciones para el teclado.

EJEMPLO 12-18

;Procedimiento de interrupción para el teclado de la figura 12-27

```

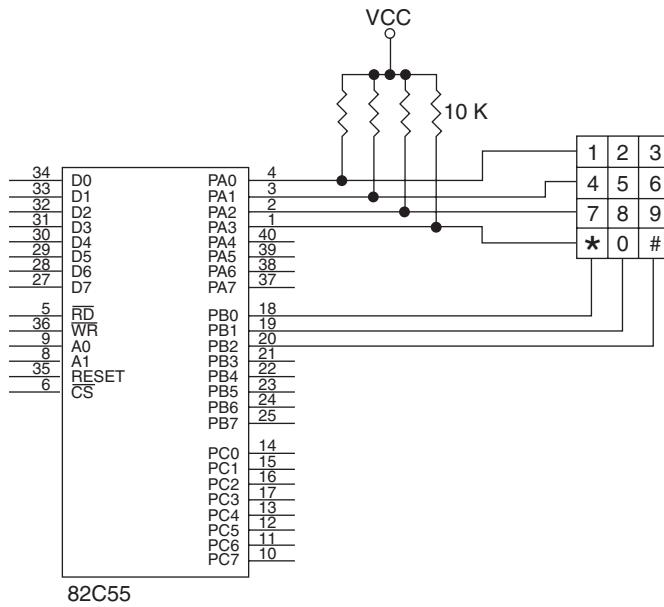
PUERTA  EQU    1000H
PUERTB  EQU    1001H

CNTER   DB     0           ;contador eliminador de rebotes
BER     DB     0           ;bandera eliminadora de rebotes
PNTR    DW     COLA        ;apuntador de entrada a la cola
OPNTR   DW     COLA        ;apuntador de salida a la cola
COLA    DB     16 DUP(?)   ;cola de 16 bytes

INTTEC  PROC   FAR USES AX BX DX
        MOV     DX,PUERTA          ;comprueba si hay una tecla
        IN      AL,DX
        OR      AL,0F0H
        .IF    AL != OFFH          ;si la tecla está oprimida
              INC CNTER          ;incrementa contador eliminador de rebotes
              .IF CNTER == 3         ;si la tecla está oprimida por > 20 ms

```

FIGURA 12-27 Interfaz entre un teclado estilo telefónico y el 82C55.



```

DEC CNTTER
.IF BER == 0
    MOV BER,1
    MOV BX,00FEH
.WHILE 1 ;busca la tecla
    MOV AL,BL
    MOV DX,PUERTB
    OUT DX,AL
    ROL BL,1
    MOV DX,PUERTA
    IN AL,DX
    OR AL,0FOH
.BREAK .IF AL != 0
    ADD BH,4
.ENDW
MOV BL,AL
MOV AL,0
MOV DX,PUERTB
OUT DX,AL
DEC BH
.REPEAT
    SHR BL,1
    INC BH
.UNTIL !CARRY?
MOV AL,BH
MOV BX,PNTR
MOV [BX],AL ;código de tecla a la cola
INC BX
.IF BX == OFFSET COLA+16
    MOV DX,OFFSET COLA
.ENDIF
MOV PNTR,BX
.ENDIF
.ENDIF
.ELSE ;si no hay tecla oprimida
    DEC CNTTER ;decrementa contador eliminador de rebotes
    .IF SIGN? ;si es menor que cero
        MOV CNTTER,0
        MOV BER,0
    .ENDIF
.ENDIF
IRET
INTTEC ENDP

```

La interrupción del teclado busca la tecla y almacena el código de tecla en la cola. El código que se almacena en la cola es un código puro que no indica el número de la tecla. Por ejemplo, el código para la tecla 1 es 00H, el código para la tecla 4 es 01H, y así sucesivamente. No hay provisión para un desbordamiento de la cola en este software. Podría agregarse pero, en la mayoría de los casos es difícil desbordar una cola de 16 bytes.

El ejemplo 12-19 muestra un procedimiento que extrae datos de la cola del teclado. Este procedimiento no se controla mediante interrupciones y se llama sólo cuando se necesita la información del teclado en un programa. El ejemplo 12-20 muestra el software que llama al procedimiento TECLA.

EJEMPLO 12-19

```
BUSCA    DB      1,4,7,10      ;tabla de búsqueda
        DB      2,5,8,0
        DB      3,6,9,11

TECLA    PROC    NEAR USES BX

        MOV BX,OPNTR
        .IF BX == PNTR          ;si la cola está vacía
            STC
        .ELSE
            MOV AL,[BX]          ;obtiene los datos de la cola
            INC BX
            .IF BX == OFFSET COLA+16
                MOV BX,OFFSET COLA
            .ENDIF
            MOV OPNTR,BX
            MOB BX,BUSCA
            XLAT
            CLC
        .ENDIF
        RET

TECLA    ENDP
```

EJEMPLO 12-20

```
.REPEAT
    CALL TECLA
.UNTIL !CARRY?
```

12-6

RESUMEN

1. Una interrupción es una llamada iniciada por hardware o software que interrumpe el programa que se esté ejecutando en cualquier punto dado en ese momento y llama a un procedimiento. Este procedimiento se llama mediante el manejador de interrupciones o un procedimiento de servicio de interrupciones.
2. Las interrupciones son útiles cuando es necesario dar servicio a un dispositivo de E/S sólo en ocasiones, con velocidades bajas de transferencia de datos.
3. El microprocesador tiene cinco instrucciones que se aplican a las interrupciones: BOUND, INT, INT 3, INTO e IRET. Las instrucciones INT e INT 3 hacen llamadas a procedimientos cuyas direcciones están almacenadas en el vector de interrupción cuyo tipo se indica mediante la instrucción. La instrucción BOUND es una interrupción condicional que utiliza el número de tipo de vector de interrupción 5. La instrucción INTO es una interrupción condicional que interrumpe un programa sólo si se activa la bandera de desbordamiento. Por último, la instrucción IRET se utiliza para regresar de los procedimientos de servicio de interrupciones.
4. El microprocesador tiene tres terminales que se aplican a su estructura de interrupciones de hardware: INTR, NMI e INTA. Las entradas de interrupción son INTR y NMI, las cuales se utilizan para solicitar interrupciones; INTA es una salida que se utiliza para aceptar la petición de interrupción INTR.

5. Se hace referencia a las interrupciones en modo real mediante una tabla de vectores que ocupa las posiciones de memoria 0000H-03FFH. Cada vector de interrupción es de cuatro bytes y contiene las direcciones de segmento y desplazamiento del procedimiento de servicio de interrupciones. En el modo protegido, las interrupciones hacen referencia a la tabla de descriptores de interrupción (IDT), la cual contiene 256 descriptores de interrupción. Cada descriptor de interrupción contiene un selector de segmento y una dirección de desplazamiento de 32 bits.
6. En la estructura de interrupciones del microprocesador se utilizan dos bits de bandera: atrapamiento (TF) y habilitación de interrupción (IF). El bit de bandera IF habilita la entrada de interrupción INTR y el bit de bandera TF hace que las interrupciones ocurran después de la ejecución de cada instrucción, siempre y cuando TF esté activo.
7. Las primeras 32 posiciones de vectores de interrupción están reservadas para uso de Intel; muchas de ellas están predefinidas en el microprocesador. Los últimos 224 vectores de interrupción son para uso del usuario y pueden realizar cualquier función que se desee.
8. Cada vez que se detecta una interrupción ocurren los siguientes eventos: (1) las banderas se meten a la pila, (2) se borran los bits de bandera IF y TF, (3) los registros IP y CS se meten a la pila y (4) el vector de interrupción se obtiene de la tabla de vectores de interrupción y se accede a la subrutina de servicio de interrupciones a través de la dirección del vector.
9. El rastreo o paso individual se logra mediante la activación del bit de bandera TF. Esto hace que se produzca una interrupción después de la ejecución de cada instrucción, para fines de depuración.
10. La entrada de interrupción no enmascarable (NMI) llama al procedimiento cuya dirección se almacena en el número de tipo de vector de interrupción 2. Esta entrada se dispara en los bordes positivos.
11. La terminal INTR no se decodifica en forma interna, como la terminal NMI. En vez de ello se utiliza INTA para aplicar el número de tipo de vector de interrupción a las conexiones D₀-D₇ del bus de datos durante el pulso INTA.
12. Los métodos para aplicar el número de tipo de vector de interrupción al bus de datos durante INTA varían mucho. Uno de esos métodos utiliza resistencias para aplicar el número de tipo de interrupción FFH al bus de datos, mientras que otro utiliza un búfer de tres estados para aplicar cualquier número de tipo de vector.
13. El controlador de interrupciones programable (PIC) 8259A agrega por lo menos ocho entradas de interrupción al microprocesador. Si se necesitan más interrupciones, este dispositivo puede conectarse en cascada para proporcionar hasta 64 entradas de interrupción.
14. La programación del 8259A es un proceso de dos etapas. Primero se envía una serie de palabras de comandos de inicialización (ICW) al 8259A; después se envía una serie de palabras de comandos de operación (OCW).
15. El 8259A contiene tres registros de estado: IMR (registro de máscara de interrupción), ISR (registro de servicio de entrada) e IRR (registro de petición de interrupción).
16. Un reloj en tiempo real se utiliza para llevar la hora en tiempo real. En la mayoría de los casos, el tiempo se almacena en forma binaria o BCD, en varias posiciones de memoria.

1. ¿Qué es lo que se interrumpe debido a una interrupción?
2. Defina el término *interrupción*.
3. ¿Qué es lo que se llama debido a una interrupción?
4. ¿Por qué las interrupciones liberan tiempo para el microprocesador?
5. Liste las terminales de interrupción en el microprocesador.
6. Liste las cinco instrucciones de interrupción para el microprocesador.
7. ¿Qué es un vector de interrupción?
8. ¿En qué parte de la memoria del microprocesador se encuentran los vectores de interrupción?
9. ¿Cuántos vectores de interrupción distintos se encuentran en la tabla de vectores de interrupción?
10. ¿Qué vectores de interrupción están reservados por Intel?
11. Explique cómo se produce una interrupción tipo 0.

12. ¿En dónde se encuentra la tabla de descriptores de interrupción para la operación en modo protegido?
13. ¿Qué información contiene cada descriptor de interrupción en modo protegido?
14. Describa las diferencias entre una interrupción en modo protegido y una en modo real.
15. Describa la operación de la instrucción BOUND.
16. Describa la operación de la instrucción INTO.
17. ¿Qué posiciones de memoria contiene el vector para una instrucción INT 44H?
18. Explique la operación de la instrucción IRET.
19. ¿Cuál es el propósito del número de tipo de vector de interrupción 7?
20. Liste los eventos que ocurren cuando se activa una interrupción.
21. Explique el propósito de la bandera de interrupción (IF).
22. Explique el propósito de la bandera de atrapamiento (TF).
23. ¿Cómo se borra y se activa IF?
24. ¿Cómo se borra y se activa TF?
25. ¿A través de qué número de tipo de vector se vectoriza la entrada de interrupción NMI en forma automática?
26. ¿Se activa la señal INTA para la terminal NMI?
27. La entrada INTR es sensible a _____.
28. La entrada NMI es sensible a _____.
29. Cuando la señal INTA se vuelve un 0 lógico, indica que el microprocesador está esperando que se coloque un número de _____ de interrupción en el bus de datos (D₀-D₇).
30. ¿Qué es una FIFO?
31. Desarrolle un circuito que coloque el número de tipo de interrupción 86H en el bus de datos, en respuesta a la entrada INTR.
32. Desarrolle un circuito que coloque el número de tipo de interrupción CCH en el bus de datos, en respuesta a la entrada INTR.
33. Explique por qué las resistencias elevadoras en D₀-D₇ hacen que el microprocesador responda con el número de tipo de vector de interrupción FFH para el pulso INTA.
34. ¿Qué es una cadena tipo margarita?
35. ¿Por qué deben sondarse los dispositivos que provocan interrupciones en un sistema de interrupciones en cadena tipo margarita?
36. ¿Qué es el 8259A?
37. ¿Cuántos controladores 8259A se requieren para tener 64 entradas de interrupción?
38. ¿Cuál es el propósito de las terminales IR₀-IR₇ del 8259A?
39. ¿Cuándo se utilizan las terminales CAS₂-CAS₀ en el 8259A?
40. ¿En dónde se conecta una terminal INT de esclavo en el 8259A maestro, en un sistema en cascada?
41. ¿Qué es una ICW?
42. ¿Qué es una OCW?
43. ¿Cuántas ICW se necesitan para programar el 8259A cuando opera como un solo controlador maestro en un sistema?
44. ¿En dónde se almacena el número de tipo de vector en el 8259A?
45. ¿En dónde se programa la sensibilidad de las terminales IR en el 8259A?
46. ¿Cuál es el propósito de ICW₁?
47. ¿Qué es un EOI no específico?
48. Explique el desplazamiento cíclico de prioridades en el 8259A.
49. ¿Cuál es el propósito de IRR en el 8259A?
50. ¿En cuáles vectores de interrupción se encuentra el 8259A maestro en la computadora personal?
51. ¿En cuáles vectores de interrupción se encuentra el 8259A esclavo en la computadora personal?

CAPÍTULO 13

Acceso directo a memoria y E/S controlada por DMA

INTRODUCCIÓN

En capítulos anteriores hablamos sobre la E/S básica y procesada por interrupciones. Ahora hablaremos sobre la última forma de E/S, conocida como acceso directo a memoria (DMA). La técnica de E/S por DMA proporciona un acceso directo a la memoria mientras el microprocesador está temporalmente deshabilitado. Esto permite la transferencia de datos entre memoria y el dispositivo de E/S a una velocidad que está limitada sólo por la velocidad de los componentes de memoria en el sistema o del controlador de DMA. La velocidad de transferencia de DMA puede alcanzar entre 33 y 150 Mbytes con los componentes de memoria RAM de alta velocidad que se utilizan en la actualidad.

Las transferencias por DMA se utilizan para muchos fines; lo más común es utilizar este tipo de transferencias en el refresco de DRAM, en la visualización de vídeo para refrescar la pantalla y en las operaciones de lectura y escritura del sistema de memoria en disco. La transferencia por DMA se utiliza también para realizar transferencias de memoria a memoria de alta velocidad.

En este capítulo también explicaremos la operación de los sistemas de memoria en disco y los sistemas de vídeo que por lo general se procesan por DMA. La memoria en disco está compuesta por el almacenamiento en disco flexible, fijo y óptico. Los sistemas de vídeo están compuestos por los monitores digitales y analógicos.

OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

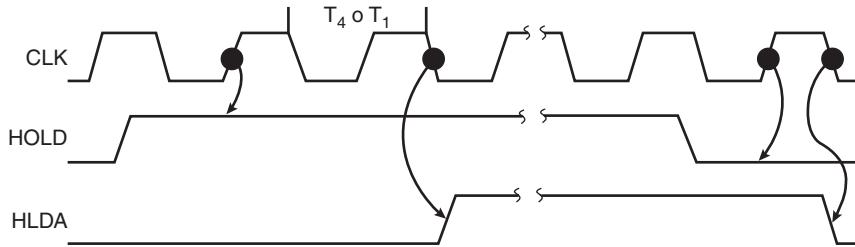
1. Describir una transferencia por DMA.
2. Explicar la operación de las señales de control HOLD y HLDA de acceso directo a memoria.
3. Explicar el funcionamiento del controlador DMA 8237 y su utilización en las transferencias por DMA.
4. Programar el 8237 para realizar transferencias por DMA.
5. Describir los estándares de disco que se utilizan en los sistemas de computadora personal.
6. Describir los diversos estándares de la interfaz de vídeo que se utilizan en la computadora personal.

13-1

OPERACIÓN BÁSICA DE DMA

Hay dos señales de control que se utilizan para solicitar y admitir una transferencia de acceso directo a memoria (DMA) en el sistema basado en microprocesador. La terminal HOLD es una entrada que se utiliza para solicitar una acción de DMA y la terminal HLDA es una salida que admite la acción de DMA. La figura 13-1 muestra la sincronización que se utiliza con frecuencia en estas dos terminales de control de DMA.

FIGURA 13-1
Sincronización de HOLD y HLDA para el microprocesador.



Cada vez que la entrada HOLD se coloca en el nivel de 1 lógico, se solicita una acción (retención) de DMA. El microprocesador responde en un lapso no mayor a unos cuantos ciclos y suspende la ejecución del programa; para ello coloca sus buses de direcciones, de datos y de control en sus estados de alta impedancia. Al entrar en este estado, es como si se hubiera removido el microprocesador de su zócalo. De esta forma, los dispositivos de E/S externos u otros microprocesadores pueden obtener acceso a los buses del sistema, para acceder a la memoria de manera directa.

Como lo indica el diagrama de sincronización, HOLD se muestrea a mitad de cualquier ciclo de reloj. Por ende, la retención puede llevarse a cabo en cualquier momento durante la operación de cualquier instrucción del conjunto de instrucciones del microprocesador. Tan pronto como el microprocesador reconoce la retención, deja de ejecutar software y entra a los ciclos de retención. La entrada HOLD tiene una prioridad más alta que las entradas de interrupción INTR o NMI. Las interrupciones se llevan a cabo al final de una instrucción, mientras que un HOLD se efectúa a la mitad de una instrucción. La única terminal del microprocesador que tiene una prioridad más alta que un HOLD es la terminal RESET. La entrada HOLD no puede estar activa durante un RESET, ya que de lo contrario no se garantiza el reinicio.

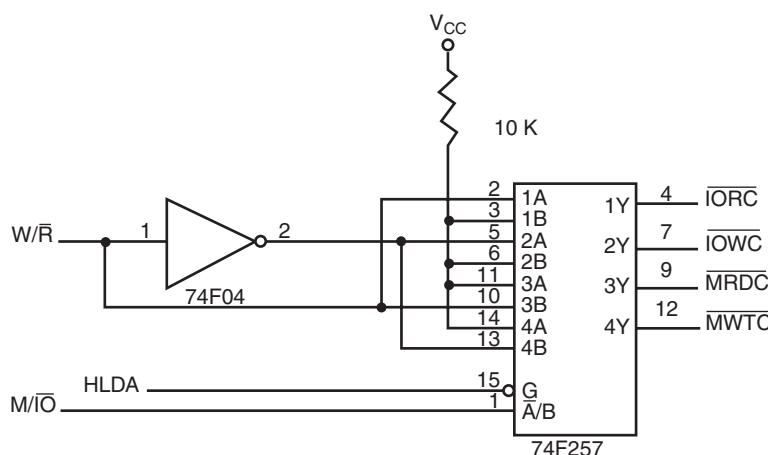
La señal HLDA se activa para indicar que el microprocesador ha colocado sus buses en su estado de alta impedancia, como puede ver en el diagrama de sincronización. Hay unos cuantos ciclos de reloj entre el momento en que cambia HOLD y hasta que cambia HLDA. La salida HLDA es una señal para el dispositivo externo solicitante, la cual le indica que el microprocesador ha renunciado al control de su espacio de memoria y de E/S. A la entrada HOLD se le conoce como entrada de petición de DMA y a la salida HLDA se le conoce como señal de concesión de DMA.

Definiciones básicas de DMA

Por lo general, los accesos directos a memoria se llevan a cabo entre un dispositivo de E/S y la memoria sin necesidad de usar el microprocesador. Una **lectura de DMA** transfiere datos desde la memoria hacia el dispositivo de E/S. Una **escritura de DMA** transfiere datos desde un dispositivo de E/S hacia la memoria. En ambas operaciones la memoria y la E/S se controlan de forma simultánea, razón por la cual el sistema contiene señales separadas para el control de memoria y el control de E/S. Esta estructura especial del bus de control del microprocesador permite las transferencias por DMA. Una lectura de DMA provoca la activación simultánea de las señales MRDC e IOWC para transferir datos desde la memoria hacia el dispositivo de E/S. Una escritura de DMA provoca la activación simultánea de las señales MWTC e IORC. Estas señales del bus de control están disponibles para todos los microprocesadores de la familia Intel, excepto el sistema 8086/8088. Estos microprocesadores requieren que se genere estas señales mediante un controlador del sistema o un circuito tal como el que se muestra en la figura 13-2. El controlador de DMA proporciona a la memoria su dirección y una señal del controlador (DACK) selecciona el dispositivo de E/S durante la transferencia por DMA.

La velocidad de transferencia de datos se determina en base a la velocidad del dispositivo de memoria o mediante un controlador de DMA que se utiliza a menudo para controlar las transferencias por DMA. Si la velocidad de la memoria es de 50 ns, las transferencias por DMA se llevan a cabo a velocidades de hasta 1/50 ns, o 20 Mbytes por segundo. Si el controlador de DMA en un sistema funciona a una velocidad máxima de 15 MHz y se utiliza memoria de 50 ns, la velocidad máxima de transferencia es de 15 MHz ya que el controlador de DMA es más lento que la memoria. En muchos casos, el controlador de DMA reduce la velocidad del sistema cuando se llevan a cabo las transferencias por DMA.

FIGURA 13-2 Un circuito que genera señales de control del sistema en un entorno DMA.



13-2

EL CONTROLADOR DE DMA 8237

Este controlador provee a la memoria y la E/S señales de control e información sobre direcciones de memoria durante la transferencia por DMA. El 8237 es en realidad un microprocesador de propósito especial, cuya función es la transferencia de datos de alta velocidad entre la memoria y la E/S. La figura 13-3 muestra el diagrama de terminales y de bloques del controlador de DMA programable 8237. Aunque tal vez este dispositivo no aparezca como un componente discreto en los sistemas modernos basados en microprocesador, aparece dentro de los conjuntos de chips controladores del sistema que se encuentran en la mayoría de los sistemas. Aunque no los describiremos aquí debido a su complejidad, el conjunto de chips (ISP 82875 o controlador integrado de periféricos del sistema) y su conjunto integral de dos controladores de DMA se programan de una manera casi idéntica (no soporta las transferencias de memoria a memoria) al 8237. El ISP también proporciona un par de controladores de interrupciones programables 8259A para el sistema.

El 8237 es un dispositivo de cuatro canales, compatible con los microprocesadores 8086/8088. Este dispositivo puede expandirse para incluir cualquier número de entradas de canal DMA, aunque cuatro canales son una cantidad adecuada para la mayoría de los sistemas pequeños. El 8237 es capaz de realizar transferencias por DMA a velocidades de hasta 1.6 Mbytes por segundo. Cada canal puede direccionar una sección completa de 64 Kbytes de memoria y puede transferir hasta 64 Kbytes mediante una sola programación.

Definiciones de las terminales

CLK	La entrada reloj se conecta a la señal de reloj del sistema, siempre y cuando ésta sea de 5 MHz o menor. En el sistema 8086/8088 el reloj debe invertirse para que el 8237 opere de manera correcta.
CS	Selección de chip habilita la programación del 8237. Por lo general, la terminal CS se conecta a la salida de un decodificador. Éste no utiliza la señal de control IO/M (M/I/O) del 8086/8088, ya que contiene las nuevas señales de control de memoria y E/S (MEMR , MEMW , IOR e IOW).
RESET	La terminal reinicio borra los registros de comandos, de estado, de petición y temporal. También borra el primer/último flip-flop y establece el registro de máscara. Esta entrada prepara el 8237 para deshabilitarlo hasta que se programe.
READY	Un 0 lógico en la entrada listo hace que el 8237 entre en estados de espera para los componentes de memoria más lentos.
HLDA	La terminal aceptación de retención indica al 8237 que el microprocesador ha renunciado al control de los buses de direcciones, de datos y de control.

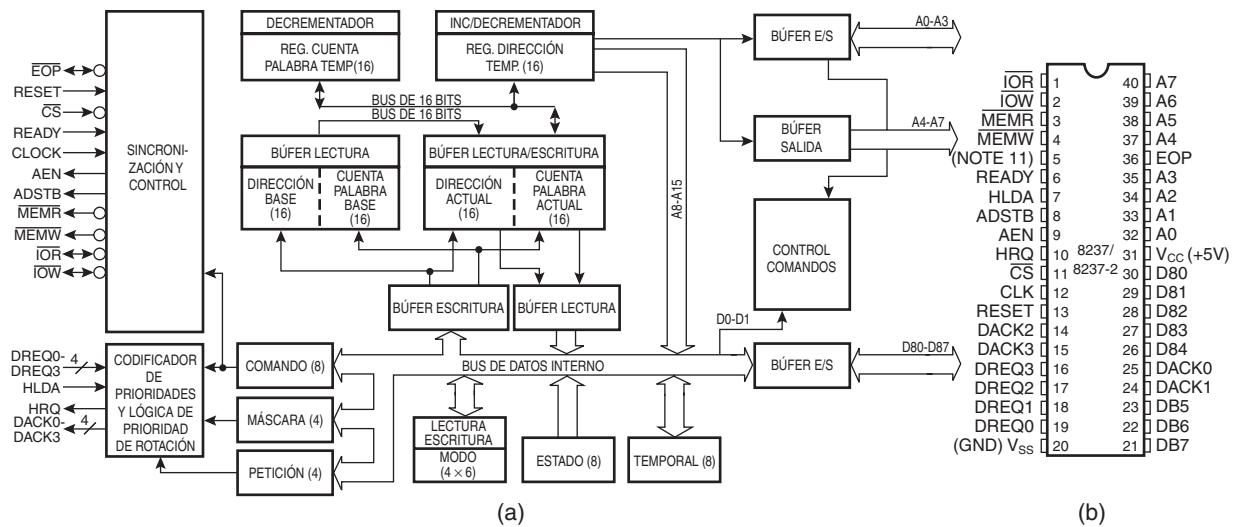


FIGURA 13-3 El controlador de DMA programable 8237A-5. (a) Diagrama de bloques y (b) diagrama de terminales.

DREQ₀-DREQ₃	Las entradas de petición de DMA se utilizan para solicitar una transferencia por DMA para cada uno de los cuatro canales de DMA. Como la polaridad de estas entradas es programable, pueden ser entradas activas en nivel alto o en nivel bajo.
DB₀-DB₇	Las terminales bus de datos se conectan a las conexiones del bus de datos del microprocesador y se utilizan durante la programación del controlador de DMA.
IOR	Lectura de E/S es una terminal bidireccional que se utiliza durante la programación y durante un ciclo de escritura de DMA.
IOW	Escritura de E/S es una terminal bidireccional que se utiliza durante la programación y durante un ciclo de lectura de DMA.
EOP	Fin del proceso es una señal bidireccional que se utiliza como entrada para terminar un proceso de DMA o como una salida para indicar el final de la transferencia por DMA. Esta entrada se utiliza a menudo para interrumpir una transferencia por DMA al final de un ciclo DMA.
A₀-A₃	Estas terminales de dirección son salidas que seleccionan un registro interno durante la programación y también proporcionan parte de la dirección de transferencia por DMA durante una acción de DMA.
HRQ	Retención de petición es una salida que se conecta a la entrada HOLD del microprocesador, para solicitar una transferencia por DMA.
DACK₀-DACK₃	Las salidas aceptación de canal de DMA aceptan una petición de canal de DMA. Estas salidas pueden programarse como señales activas en nivel alto o en bajo. Las salidas DACK se utilizan por lo general para seleccionar el dispositivo de E/S controlado por DMA durante la transferencia por DMA.
AEN	La señal habilitación de dirección habilita el enclavamiento de dirección DMA que se conecta a las terminales DB ₇ -DB ₀ del 8237. También se utiliza para deshabilitar cualquier búfer en el sistema que esté conectado al microprocesador.
ADSTB	Estrobo de dirección funciona como la señal ALE, sólo que el controlador de DMA la utiliza para enclavar los bits de dirección A ₁₅ -A ₁₈ durante la transferencia por DMA.
MEMR	Lectura de memoria es una salida que hace que la memoria lea datos durante un ciclo de lectura DMA.

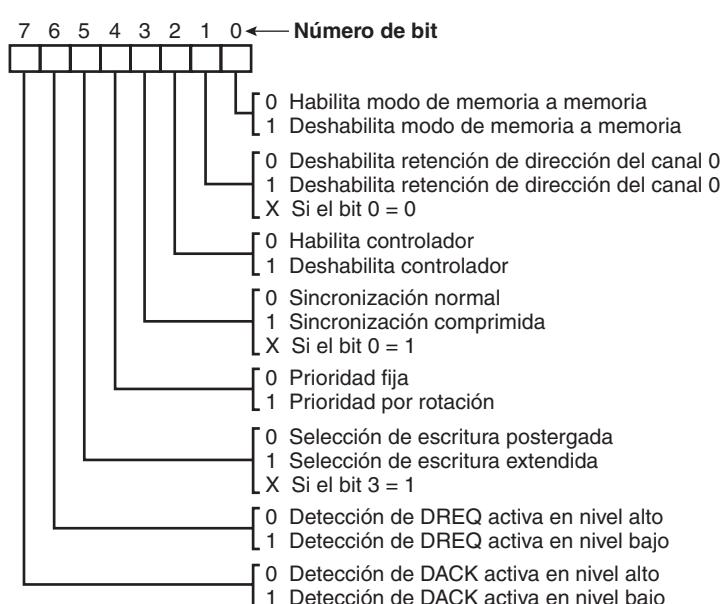
MEMW **Escritura en memoria** es una salida que hace que la memoria escriba datos durante un ciclo de escritura DMA.

Registros internos

- CAR** El **registro de dirección actual** se utiliza para almacenar la dirección de memoria de 16 bits que se va a utilizar para la transferencia por DMA. Cada canal tiene su propio registro de dirección actual para este fin. Cuando se transfiere un byte de datos durante una operación de DMA, el CAR se incrementa o se decrementa, dependiendo de cómo esté programado.
- CWCR** El **registro de cuenta de palabra actual** programa un canal para el número de bytes (hasta 64 K) que se van a transferir durante una acción de DMA. El número que se carga en este registro es uno menos que el número de bytes transferidos. Por ejemplo, si se carga un 10 en el CWCR, entonces se van a transferir 11 bytes durante la acción de DMA.
- BA y BWC** Los registros **dirección base (BA)** y **cuenta de palabra base (BWC)** se utilizan cuando se selecciona la inicialización automática para un canal. En el modo de inicialización automática, estos registros se utilizan para recargar los registros CAR y CWCR cuando se completa la acción de DMA. Esto permite utilizar la misma cuenta y dirección para transferir datos desde la misma área de memoria.
- CR** El **registro de comandos** programa la operación del controlador de DMA 8237. La figura 13-4 muestra la función del registro de comandos. Este registro utiliza la posición de bit 0 para seleccionar el modo de transferencia por DMA de memoria a memoria. Este tipo de transferencias utilizan el canal 0 de DMA para almacenar la dirección de origen y el canal 1 de DMA para almacenar la dirección de destino. (Esto es similar a la operación de una instrucción MOVSB.) Se lee un byte de la dirección a la que accede el canal 0 y se guarda dentro del 8237 en un registro de retención temporal. A continuación, el 8237 inicia un ciclo de escritura en memoria, en el que el contenido del registro de retención temporal se escribe en la dirección seleccionada por el canal 1 de DMA. El número de bytes a transferir se determina en base al registro de conteo del canal 1.

FIGURA 13-4

El registro de comandos del 8237A-5. (Cortesía de Intel Corporation.)



El bit de habilitación de retención de dirección del canal 0 (posición de bit 1) programa el canal 0 para las transferencias de memoria a memoria. Por ejemplo, si se debe llenar un área de memoria con datos, el canal 0 puede retenerte en la misma dirección mientras que el canal 1 cambia para la transferencia de memoria a memoria. Esta acción copia el contenido de la dirección a la que accede el canal 0 en un bloque de memoria utilizado por el canal 1.

El bit de habilitación/deshabilitación del controlador (posición de bit 2) conecta y desconecta el controlador por completo. El bit de sincronización normal y comprimida (posición de bit 3) determina si un ciclo de DMA contiene dos (comprimida) o cuatro (normal) periodos de reloj. La posición de bit 5 se utiliza en la sincronización normal para extender el pulso de escritura, de manera que aparezca un ciclo de reloj antes en la sincronización, para los dispositivos de E/S que requieren un pulso de escritura más ancho.

La posición de bit 4 selecciona la prioridad para las entradas DREQ de los cuatro canales de DMA. En el esquema de prioridad fija, el canal 0 tiene la prioridad más alta y el canal 3 tiene la más baja. En el esquema de prioridad por rotación, el canal que se haya atendido de manera más reciente asume la prioridad más baja. Por ejemplo, si el canal 2 acaba de tener acceso a una transferencia por DMA, asume la prioridad más baja y el canal 3 asume la posición de prioridad más alta. La prioridad por rotación es un intento por dar igualdad de prioridad a todos los canales.

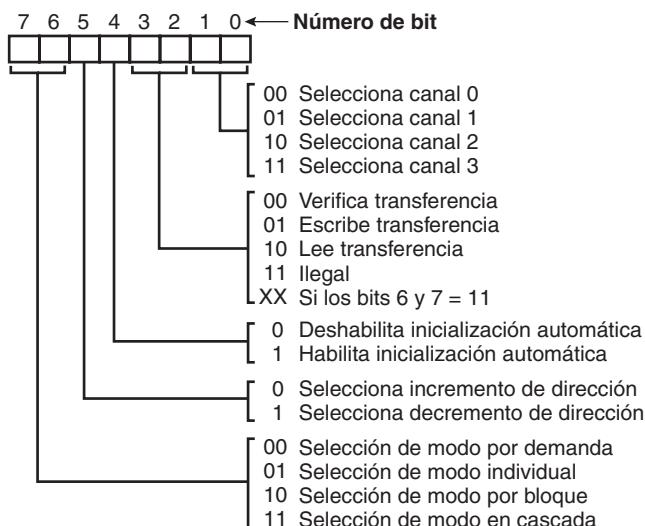
Los dos bits restantes (posiciones de bit 6 y 7) programan las polaridades de las entradas DREQ y las salidas DACK.

MR

El **registro de modo** programa el modo de operación para un canal. Cada canal tiene su propio registro de modo (vea la figura 13-5), según lo seleccionen las posiciones de bit 1 y 0. Los bits restantes del registro de modo seleccionan la operación, la inicialización automática, el incremento/decremento y el modo para el canal. Las operaciones de verificación generan las direcciones de DMA sin generar las señales de DMA de control de memoria y E/S.

Los modos de operación son: modo por demanda, modo individual, modo por bloque y modo en cascada. En el modo por demanda se transfieren datos hasta que se introduce un EOP externo o hasta que la entrada DREQ queda inactiva. En el modo individual se libera la retención (HOLD) después de transferir cada byte de datos. Si la terminal DREQ se mantiene activa, el 8237 solicita de nuevo una transferencia por

FIGURA 13-5 El registro de modo del 8273A-5. (Cortesía de Intel Corporation.)



DMA a través de la línea DRQ que va a la entrada HOLD del microprocesador. En el modo por bloque se transfiere de manera automática el número de bytes indicados por el registro de conteo para el canal. La señal DREQ no necesita mantenerse activa durante la transferencia en modo por bloque. El modo en cascada se utiliza cuando hay más de un 8237 en un sistema.

- BR** El **registro de petición de bus** se utiliza para solicitar una transferencia por DMA mediante software (vea la figura 13-6). Esto es muy útil en las transferencias de memoria a memoria, en donde no hay disponible una señal externa para comenzar la transferencia por DMA.
- MRSR** El modo de **activación/reinicio del registro de máscara** activa o borra la máscara de canal, como se muestra en la figura 13-7. Si se activa la máscara, el canal se deshabilita. Recuerde que la señal RESET activa todas las máscaras de los canales para deshabilitarlos.
- MSR** El **registro de máscara** (vea la figura 13-8) borra o activa todas las máscaras con un comando en vez de cada canal por separado, al igual que con el MRSR.
- SR** El **registro de estado** muestra el estado de cada canal de DMA (vea la figura 13-9). Los bits TC indican si el canal ha llegado a su cuenta terminal (si ha transferido todos sus bytes). Cada vez que se llega a la cuenta terminal, se termina la transferencia por DMA para la mayoría de los modos de operación. Los bits de petición indican si la entrada DREQ para un canal dado está activa.

FIGURA 13-6 El registro de petición del 8237A-5. (Cortesía de Intel Corporation.)

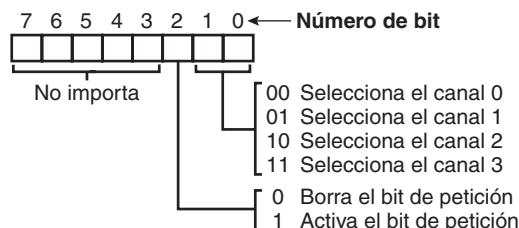


FIGURA 13-7 El modo de activación/reinicio del registro de máscara del 8237A-5. (Cortesía de Intel Corporation.)

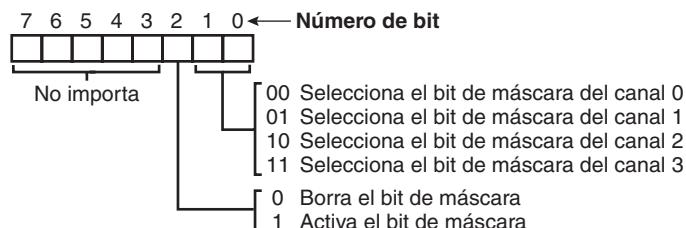


FIGURA 13-8 El registro de máscara del 8237A-5. (Cortesía de Intel Corporation.)

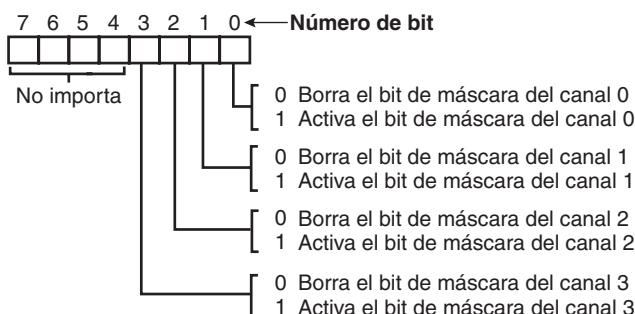
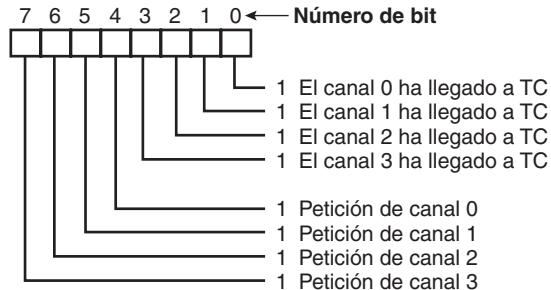


FIGURA 13-9 El registro de estado del 8237A-5. (Cortesía de Intel Corporation.)



Comandos de software

Hay tres comandos de software que se utilizan para controlar la operación del 8237. Estos comandos no tienen un patrón de bits binarios, como los diversos registros de control del 8237. Una salida simple para el número de puerto correcto habilita el comando de software. La figura 13-10 muestra las asignaciones de puertos de E/S que acceden a todos los registros y a los comandos de software.

En la siguiente lista se explican las funciones de los comandos de software:

1. **Borra el primer/último flip-flop:** borra el primer/último (P/U) flip-flop dentro del 8237. El P/U flip-flop selecciona cuál byte (de menor o mayor orden) se lee/escribe en la dirección actual y en los registros de conteo actuales. Si P/U = 0 se selecciona el byte de menor orden; si P/U = 1 se selecciona el byte de mayor orden. Cualquier lectura o escritura en la dirección o en el registro de conteo cambia de manera automática el estado del flip-flop.
2. **Borra maestro:** actúa de la misma forma que la señal RESET para el 8237. Al igual que con la señal RESET, este comando deshabilita todos los canales.
3. **Borra registro de máscara:** habilita los cuatro canales de DMA.

FIGURA 13-10
Asignaciones de puertos de comando y control del 8237A-5. (Cortesía de Intel Corporation.)

Señales						Operación
A3	A2	A1	A0	IOR	IOW	
1	0	0	0	0	1	Lee registro de estado
1	0	0	0	1	0	Escribe registro de comandos
1	0	0	1	0	1	Illegal
1	0	0	1	1	0	Escribe registro de comandos
1	0	1	0	0	1	Illegal
1	0	1	0	1	0	Escribe bit de registro de máscara individual
1	0	1	1	0	1	Illegal
1	0	1	1	1	0	Escribe registro de modo
1	1	0	0	0	1	Illegal
1	1	0	0	1	0	Borra Flip-flop apuntador de bytes
1	1	0	1	0	1	Lee registro temporal
1	1	0	1	1	0	Borra maestro
1	1	1	0	0	1	Illegal
1	1	1	0	1	0	Borra registro de máscara
1	1	1	1	0	1	Illegal
1	1	1	1	1	0	Escribe todos los bits del registro de máscara

Canal	Registro	Operación	Señales							Flip-flop interno	DB0-DB7 del bus de datos
			\bar{CS}	\bar{IOR}	\bar{IOW}	A3	A2	A1	A0		
0	Dirección base y actual	Escritura	0 0	1 0	0 0	0 0	0 0	0 0	0 0	0 1	A0-A7 A8-A15
	Dirección actual	Lectura	0 0	0 0	1 1	0 0	0 0	0 0	0 0	0 1	A0-A7 A8-A15
	Cuenta de palabras base y actual	Escritura	0 0	1 1	0 0	0 0	0 0	0 1	1 1	0 1	W0-W7 W8-W15
	Cuenta de palabras base	Lectura	0 0	0 0	1 1	0 0	0 0	0 0	1 1	0 1	W0-W7 W8-W15
1	Dirección base y actual	Escritura	0 0	1 1	0 0	0 0	0 1	1 0	0 0	0 1	A0-A7 A8-A15
	Dirección actual	Lectura	0 0	0 0	1 1	0 0	0 0	1 1	0 0	0 1	A0-A7 A8-A15
	Cuenta de palabras base y actual	Escritura	0 0	1 1	0 0	0 0	0 0	1 1	1 1	0 1	W0-W7 W8-W15
	Cuenta de palabras base	Lectura	0 0	0 0	1 1	0 0	0 0	1 1	1 1	0 1	W0-W7 W8-W15
2	Dirección base y actual	Escritura	0 0	1 1	0 0	0 0	1 1	0 0	0 0	0 1	A0-A7 A8-A15
	Dirección actual	Lectura	0 0	0 0	1 1	0 0	1 1	0 0	0 0	0 1	A0-A7 A8-A15
	Cuenta de palabras base y actual	Escritura	0 0	1 1	0 0	0 0	1 1	0 0	1 1	0 1	W0-W7 W8-W15
	Cuenta de palabras base	Lectura	0 0	0 0	1 1	0 0	1 1	0 0	1 1	0 1	W0-W7 W8-W15
3	Dirección base y actual	Escritura	0 0	1 1	0 0	0 0	1 1	1 1	0 0	0 1	A0-A7 A8-A15
	Dirección actual	Lectura	0 0	0 0	1 1	0 0	1 1	1 1	0 0	0 1	A0-A7 A8-A15
	Cuenta de palabras base y actual	Escritura	0 0	1 1	0 0	0 0	1 1	1 1	1 1	0 1	W0-W7 W8-W15
	Cuenta de palabras base	Lectura	0 0	0 0	1 1	0 0	1 1	1 1	1 1	0 1	W0-W7 W8-W15

FIGURA 13-11 Direcciones de los puertos de E/S de los canales de DMA del 8237A-5. (Cortesía de Intel Corporation.)

Programación de los registros de direcciones y de conteo

La figura 13-11 muestra las ubicaciones de los puertos de E/S para programar los registros de conteo y de direcciones para cada canal. Observe que el estado del P/U flip-flop determina si se programa el LSB o el MSB. Si se desconoce el estado del P/U flip-flop, la cuenta y la dirección podrían programarse de manera incorrecta. También es importante deshabilitar el canal de DMA antes de programar su dirección y su cuenta.

Se requieren cuatro pasos para programar el 8237: (1) el P/U flip-flop se borra mediante el uso del comando “borra P/U”; (2) se deshabilita el canal; (3) se programan el LSB y después el MSB de la dirección; y (4) se programan el LSB y el MSB de la cuenta. Una vez realizadas estas cuatro operaciones, el canal se programa y está listo para usarse. Se requiere una programación adicional para seleccionar el modo de operación antes de poder habilitar e iniciar el canal.

Conexión entre el 8237 y el microprocesador

La figura 13-12 muestra un sistema basado en el 80X86 que contiene el controlador de DMA 8237.

La salida de habilitación de dirección (AEN) del 8237 controla las terminales de salida de los enclavamientos y las salidas del 74LS257 (E). Durante la operación normal del 80X86 (AEN = 0), los enclavamientos A, C y el multiplexor (E) proporcionan los bits A₁₉-A₁₆ y A₇-A₀ del bus de direcciones. El multiplexor proporciona las señales de control del sistema, siempre y cuando el 80X86 tenga

el control del mismo. Durante una acción de DMA ($AEN = 1$) se deshabilitan los enclavamientos A y C junto con el multiplexor (E). Ahora los enclavamientos D y B proporcionan los bits de dirección $A_{19}-A_{16}$ y $A_{15}-A_8$. El 8237 proporciona directamente los bits A_7-A_0 , los cuales contienen una parte de la dirección de la transferencia por DMA. El controlador de DMA proporciona las señales de control MEMR, MEMW, IOR e IOW.

La salida de estrobo de dirección (ADSTB) del 8237 envía en un pulso de reloj a la dirección ($A_{15}-A_8$) con el enclavamiento D durante la acción de DMA, de manera que la dirección completa de transferencia por DMA esté disponible en el bus de direcciones. El enclavamiento B proporciona los bits $A_{19}-A_{16}$ del bus de direcciones. Este enclavamiento debe programarse con estos cuatro bits de dirección para poder habilitar el controlador para la transferencia por DMA. La operación de DMA del 8237 se limita a una transferencia de no más de 64 Kbytes dentro de la misma sección de 64 Kbytes de la memoria.

El decodificador (F) selecciona el 8237 para programar y el enclavamiento (B) de cuatro bits para los cuatro bits de dirección de mayor orden. El enclavamiento en una PC se llama registro de página de DMA (8 bits), el cual almacena los bits de dirección $A_{16}-A_{23}$ para una transferencia por DMA. También existe un registro de página superior, pero su dirección depende del chip. En la tabla 13-1 se muestran los números de puerto para los registros de página de DMA (éstos son para el ISP Intel). El decodificador en este sistema habilita el 8237 para las direcciones de los puertos de E/S XX60H-XX7FH, y el enclavamiento (B) de E/S para los puertos XX00H-XX1FH. La salida del decodificador se combina con la señal IOW para generar una señal de reloj que se active en nivel alto para el enclavamiento (B).

Durante la operación normal del 80X86 se deshabilitan el controlador de DMA y los circuitos integrados B y D. Durante una acción de DMA se deshabilitan los circuitos integrados A, C y E para que el 8237 pueda tomar el control del sistema a través de los buses de direcciones, de datos y de control.

En la computadora personal, los dos controladores de DMA se programan en los puertos de E/S 0000H-000FH para los canales 0-3 de DMA, y en los puertos 00C0H-00DFH para los canales 4-7 de DMA. El segundo controlador se programa sólo en direcciones pares, por lo que las direcciones base y actual del canal 4 se programan en el puerto de E/S 00COH y las cuentas base y actual del canal 4 se programan en el puerto 002CH. El registro de página, que almacena los bits de dirección $A_{23}-A_{16}$ de la dirección de DMA, se encuentra en los puertos de E/S 0087H (CH-0), 0083H (CH-1), 0081H (CH-2), 0082H (CH-3) (no hay canal 4), 008BH (CH-5), 0089H (CH-6) y 008AH (CH-7). El registro de página funciona como el enclavamiento de direcciones que describimos con los ejemplos en este libro.

Transferencia de memoria a memoria con el 8237

La transferencia de memoria a memoria es mucho más poderosa que cualquier instrucción, incluso más que la instrucción MOVSB de repetición automática. (Nota: la mayoría de los conjuntos de chips modernos no soportan la característica de memoria a memoria.) Aunque las tablas de instrucciones MOVSB repetidas muestran que el 8088 requiere 4.2 μ s por byte, el 8237 sólo requiere 2.0 μ s por byte, que es casi dos veces más rápido que una transferencia de datos por software. Esto no se aplica si en el sistema se utiliza un 80386, 80486 o un microprocesador del Pentium al Pentium 4.

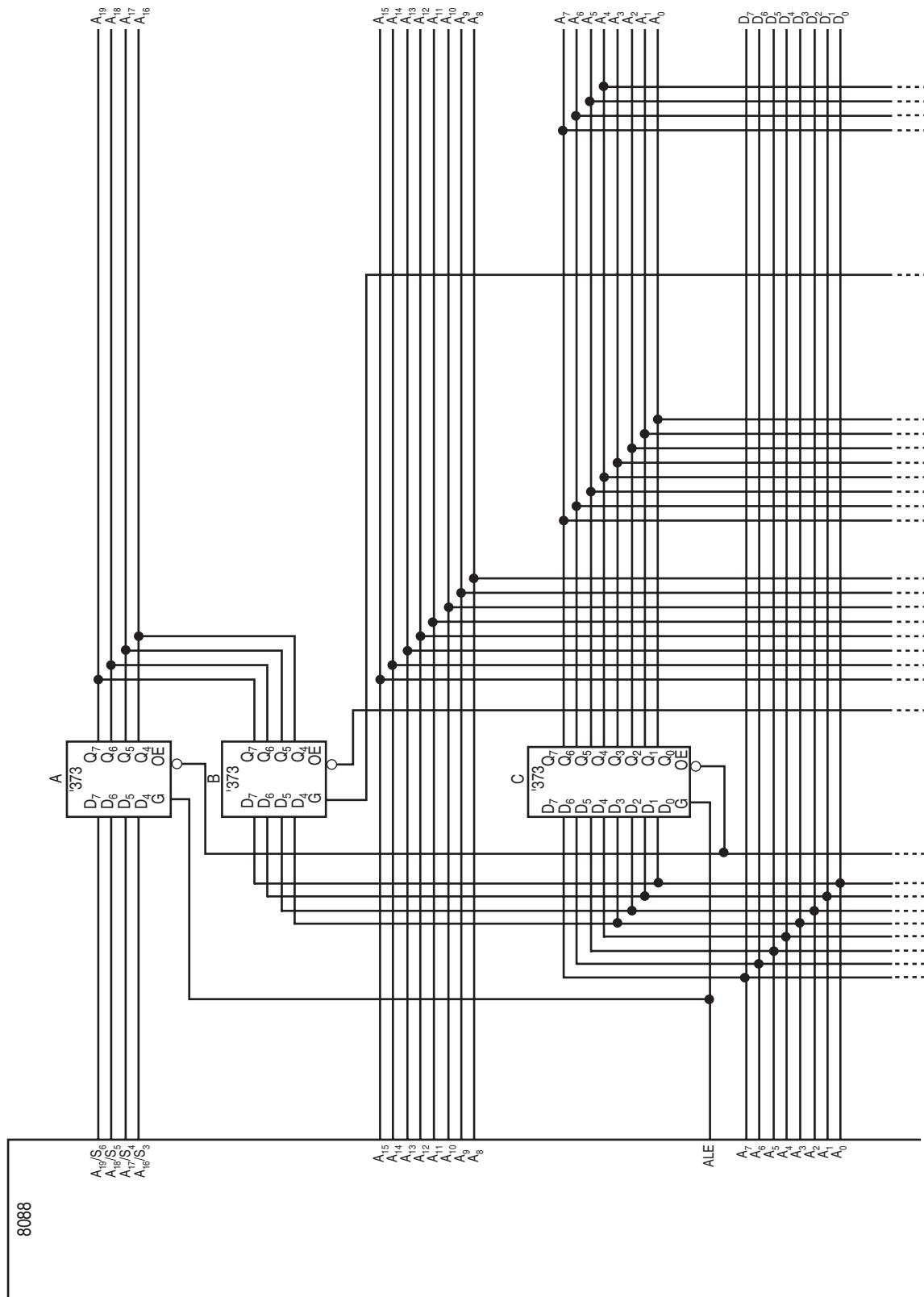
Ejemplo de transferencia por DMA de memoria a memoria. Suponga que se debe transferir el contenido de las posiciones de memoria 10000H-13FFFH hacia las posiciones de memoria 14000H-17FFFH. Esto se logra mediante una instrucción repetida de movimiento de cadenas o, si se desea una velocidad de transferencia mucho mayor, mediante el controlador de DMA.

El ejemplo 13-1 muestra el software requerido para inicializar el 8237 y para programar el enclavamiento B de la figura 13-12 para esta transferencia por DMA. Este software está escrito para una aplicación integrada. Para que pueda funcionar en la PC (si su conjunto de chips soporta esta característica), debe utilizar las direcciones de puertos que se muestran en la tabla 13-1 para los registros de página.

EJEMPLO 13-1

;Un procedimiento que transfiere un bloque de datos mediante el controlador
;de DMA 8237A de la figura 13-12. Ésta es una transferencia de
;memoria a memoria.

FIGURA 13-12 Sistema de DMA completo en modo mínimo del 8088.



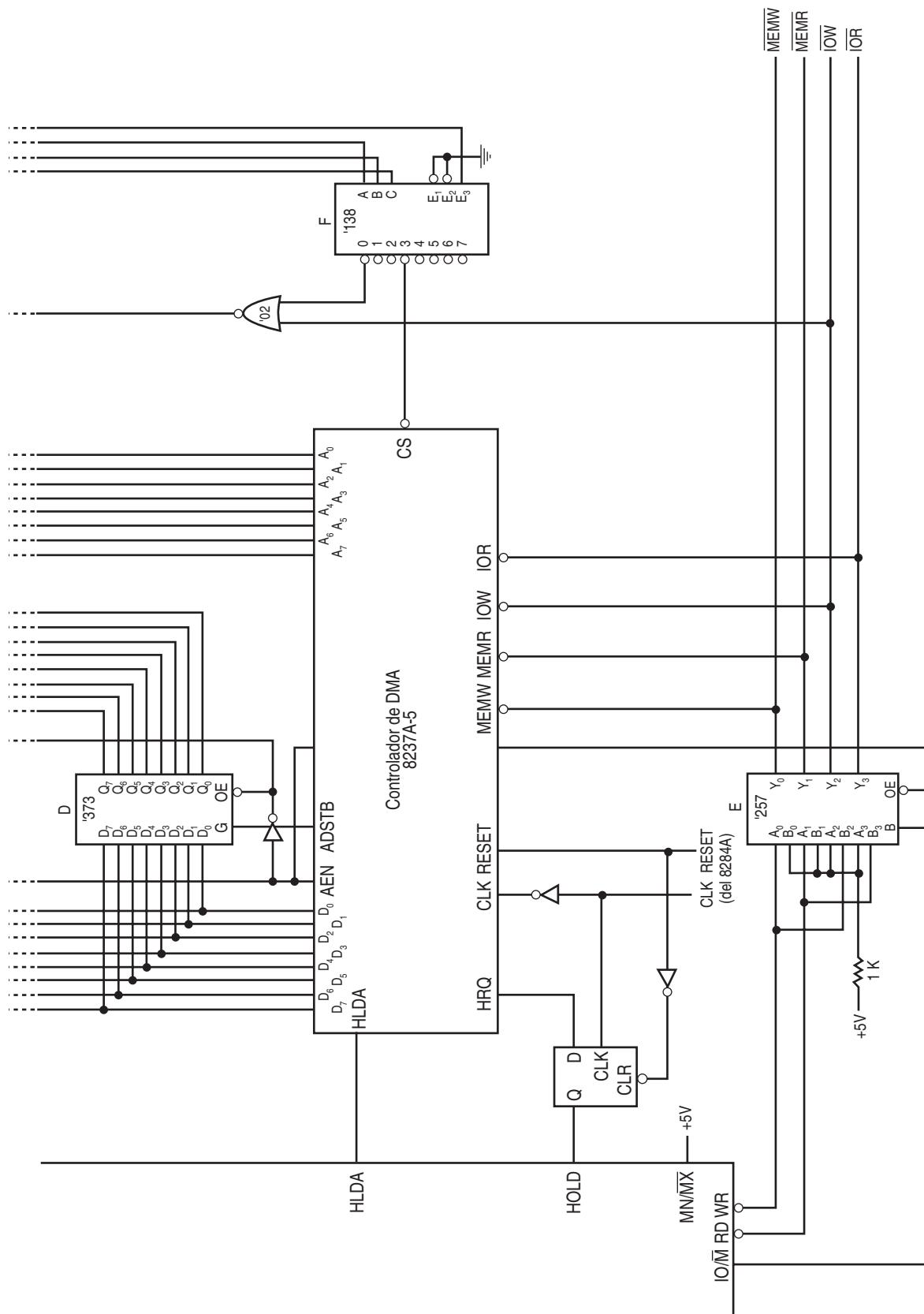


TABLA 13-1 Puertos del registro de página de DMA.

<i>Canal</i>	<i>Puerto para A₁₆–A₂₃</i>	<i>Puerto para A₂₄–A₃₁</i>
0	87H	487H
1	83H	483H
2	81H	481H
3	82H	482H
4	8FH	48FH
5	8BH	48BH
6	89H	489H
7	8AH	486H

```
;Parámetros de llamada:
;      SI = dirección de origen
;      DI = dirección de destino
;      CX = cuenta
;      ES = segmento de origen y de destino

ENCLAB    EQU    10H
BORRAF   EQU    7CH
CHOA     EQU    70H
CH1A     EQU    72H
CH1C     EQU    73H
MODO     EQU    7BH
CMNDO    EQU    78H
MASCS    EQY    7FH
REQ      EQU    79H
ESTADO   EQU    78H

TRANS     PROC    NEAR USES AX

        MOV     AX,ES           ;programa el enclavamiento B
        MOV     AL,AH
        SHR     AL,4
        OUT    ENCLAB,AL
        OUT    BORRAF,AL         ;borra P/U flip-flop

        MOV     AX,ES           ;programa dirección de origen
        SHL     AX,4
        ADD     AX,SI
        OUT    CHOA,AL
        MOV     AL,AH
        OUT    CHOA

        MOV     AX,ES           ;programa dirección de destino
        SHL     AX,4
        ADD     AX,DI
        OUT    CH1A,AL
        MOV     AL,AH
        OUT    CH1A,AL

        MOV     AX,CX           ;programa cuenta
        DEC     AX
        OUT    CH1C,AL
        MOV     AL,AH
        OUT    CH1C,AL

        MOV     AL,88H          ;programa el modo
        OUT    MODO,AL
        MOV     AL,85H
        OUT    MODO,AL

        MOV     AL,1             ;habilita transferencia por bloque
        OUT    CMNDO,AL
```

```

MOV    AL, 0EH           ;desenmascara el canal 0
OUT    MASCS, AL

MOV    AL, 4              ;inicia DMA
OUT    REQ, AL

.REPEAT
    IN AL, ESTADO
.UNTIL AL & 1
RET

TRANS ENDP

```

Para programar el controlador de DMA se requieren unos cuantos pasos, como se muestra en el ejemplo 13-1. El dígito de más a la izquierda de la dirección de cinco dígitos se envía al enclavamiento B. A continuación, los canales se programan una vez que se borra el P/U flip-flop. Observe que utilizamos el canal 0 como origen y el canal 1 como destino para una transferencia de memoria a memoria. Después se programa la cuenta con un valor igual a uno menos que el número de bytes a transferir. A continuación se programa el registro de modo de cada canal, el registro de comandos selecciona un movimiento por bloque, se habilita el canal 0 y se inicia una petición de DMA por software. Antes de regresar del procedimiento se evalúa el registro de estado para una cuenta terminal. Recuerde que la bandera de cuenta terminal indica que la transferencia de DMA está completa. La TC también deshabilita el canal, con lo que se previenen transferencias adicionales.

Ejemplo de llenado de memoria mediante el uso del 8237. Para poder llenar un área de memoria con los mismos datos, se programa el registro de origen del canal 0 para que apunte a la misma dirección durante toda la transferencia. Esto se logra mediante el modo de retención del canal 0. El controlador copia el contenido de esta posición individual de memoria a un bloque completo de memoria direccionado por el canal 1. Esto tiene muchas aplicaciones útiles.

Por ejemplo, suponga que se debe borrar una pantalla de vídeo de DOS. Esta operación puede realizarse mediante el uso del controlador de DMA con el modo de retención del canal 0 y una transferencia de memoria a memoria. Si la pantalla de vídeo contiene 80 columnas y 25 líneas, cuenta con 2000 posiciones de visualización que deben ajustarse a 20H (un espacio en ASCII) para borrar la pantalla.

El ejemplo 13-2 muestra un procedimiento que borra un área de memoria direccionada por ES:DI. El registro CX transfiere el número de bytes que se van a borrar mediante el procedimiento BORRA. Observe que este procedimiento es casi idéntico al ejemplo 13-1, sólo que el registro de comandos se programa de manera que se retenga la dirección del canal 0. La dirección de origen se programa de manera que sea la misma dirección que ES:DI, y después el destino se programa como una posición más allá de ES:DI. Observe también que este programa está diseñado para funcionar con el hardware de la figura 13-12, por lo que no funcionará en la computadora personal a menos que usted tenga el mismo hardware.

EJEMPLO 13-2

;Un procedimiento que borra la pantalla de vídeo en modo DOS mediante el uso del
;controlador de DMA, como se indica en la figura 13-12.

;Secuencia de llamada:
; DI = dirección de desplazamiento del área a borrar
; ES = dirección de segmento del área a borrar
; CX = número de bytes a borrar

ENCLAB	EQU	10H
BORRAF	EQU	7CH
CH0A	EQU	70H
CH1A	EQU	72H
CH1C	EQU	73H
MODO	EQU	7BH
CMNDO	EQU	78H
MASCS	EQU	7FH
REQ	EQU	79H

```

ESTADO EQU 78H
CERO EQU 0

BORRA PROC NEAR USES AX

    MOV AX,ES ;programa el enclavamiento B
    MOV AL,AH
    SHR AL,4
    OUT ENCLAB,AL
    OUT BORRAF,AL ;borra el P/U flip-flop

    MOV AL,CERO ;almacena cero en el primer byte
    MOV ES:[DI],AL

    MOV AX,ES ;programa dirección de origen
    SHL AX,4
    ADD AX,SI
    OUT CHOA,AL
    MOV AL,AH
    OUT CHOA

    MOV AX,ES ;programa dirección de destino
    SHL AX,4
    ADD AX,DI
    OUT CH1A,AL
    MOV AL,AH
    OUT CH1A,AL

    MOV AX,CX ;programa la cuenta
    DEC AX
    OUT CH1C,AL
    MOV AL,AH
    OUT CH1C,AL

    MOV AL,88H ;programa el modo
    OUT MODO,AL
    MOV AL,85H
    OUT MODO,AL

    MOV AL,03H ;habilita transferencia de retención por bloque
    OUT CMNDO,AL

    MOV ALOEH ;habilita el canal 0
    OUT MASCS,AL

    MOV AL,4 ;inicia DMA
    OUT REQ,AL

.REPEAT
    IN AL,ESTADO
.UNTIL AL & 1
RET

BORRA ENDP

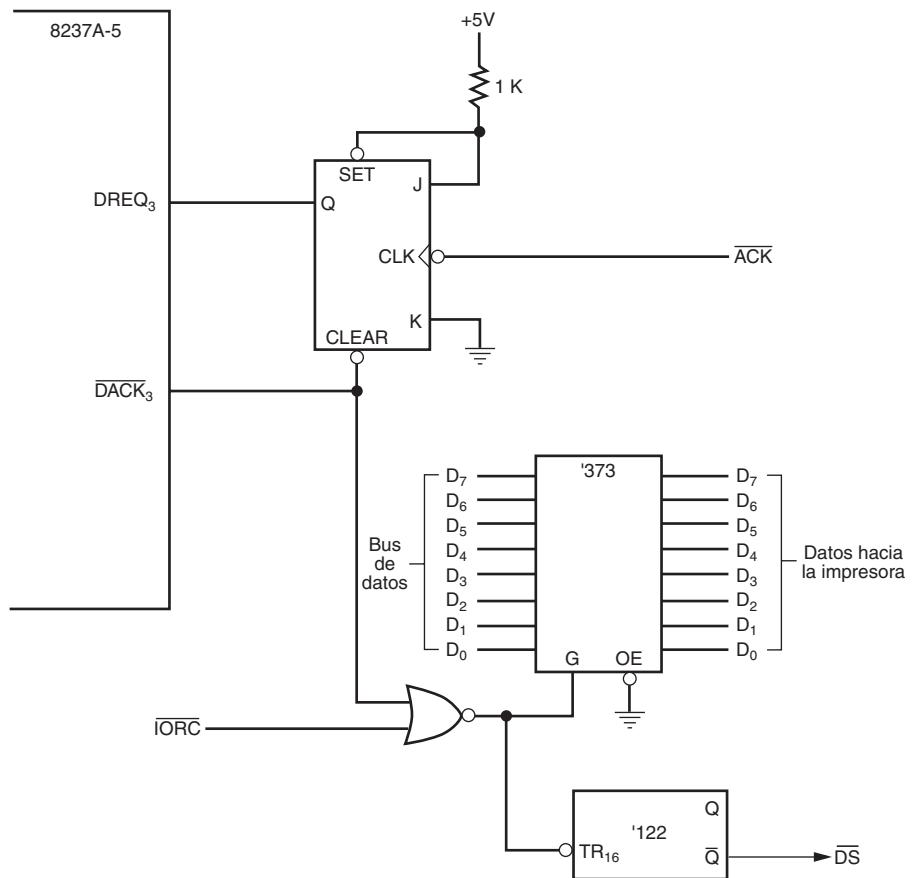
```

Interfaz de impresora procesada por DMA

La figura 13-13 muestra el hardware que se agrega a la figura 13-12 para una interfaz de impresora controlada por DMA. Se agregan muy pocos circuitos adicionales para esta interfaz con una impresora en paralelo tipo Centronics. El enclavamiento se utiliza para capturar los datos que se envían a la impresora durante la transferencia por DMA. El pulso de escritura que se pasa hacia el enclavamiento durante la acción de DMA también genera la señal de estrobo de datos (\overline{DS}) para la impresora, a través del dispositivo “single-shot”. La señal ACK regresa de la impresora cada vez que está lista para más datos. En este circuito, ACK se utiliza para solicitar una acción de DMA a través de un flip-flop.

Observe que el dispositivo de E/S no se selecciona mediante la decodificación de la dirección en el bus de direcciones. Durante la transferencia por DMA, el bus de direcciones contiene la dirección de

FIGURA 13-13 Interfaz de impresora procesada por DMA.



memoria y no puede contener la dirección del puerto de E/S. En lugar de la dirección de puerto de E/S, la salida $\overline{\text{DACK}_3}$ del 8237 selecciona el enclavamiento al aplicar el pulso de escritura a través de una compuerta OR.

El software que controla esta interfaz es simple, ya que sólo se programan la dirección de los datos y el número de caracteres a imprimir. Una vez programado, se habilita el canal y la acción de DMA transfiere un byte a la vez a la interfaz de impresora cada vez que la interfaz recibe la señal $\overline{\text{ACK}}$ de la impresora.

En el ejemplo 13-3 se muestra el procedimiento que imprime datos del segmento de datos actual. Este procedimiento programa el 8237, pero en realidad no imprime nada. La impresión se realiza mediante el controlador de DMA y la interfaz de impresora.

EJEMPLO 13-3

```
;Un procedimiento que imprime datos mediante la interfaz de
;impresora de la figura 13-13
```

```
;Secuencia de llamada:
;      BX = dirección de desplazamiento de los datos de la impresora
;      DS = dirección de segmento de los datos de la impresora
;      CX = número de bytes a imprimir
```

```
ENCLAB EQU 10H
BORRAF EQU 7CH
CH3A EQU 76H
CH1C EQU 77H
MODO EQU 7BH
```

```

CMNDO    EQU     78H
MASCS    EQY     7FH
REQ      EQU     79H

IMPRIME  PROC    NEAR USES AX CX BX

    MOV     EAX,0
    MOV     AX,DS           ;programa el enclavamiento B
    SHR     EAX,4
    PUSH    AX
    SHR     EAX,16
    OUT    ENCLAB,AL

    POP     AX           ;programa la dirección
    OUT    CH3A,AL
    MOV     AL,AH
    OUT    CH3A,AL

    MOV     AX,CX          ;programa la cuenta
    DEC     AX
    OUT    CH3C,AL
    MOV     AL,AH
    OUT    CH3C,AL

    MOV     AL,0BH          ;programa el modo
    OUT    MODO,AL

    MOV     AL,00H          ;habilita la transferencia en modo por bloque
    OUT    CMND,AL

    MOV     AL,7             ;habilita el canal 3
    OUT    MASCS,AL
    RET

IMPRIME ENDP

```

Se requiere un segundo procedimiento para determinar si se ha completado la acción de DMA. El ejemplo 13-4 muestra el segundo procedimiento que evalúa el controlador de DMA para ver si la transferencia por DMA está completa. El procedimiento PRUEBAP se llama antes de programar el controlador de DMA para ver si se completó la transferencia anterior.

EJEMPLO 13-4

```

;Un procedimiento que evalúa si se completó la acción de DMA

ESTADO  EQU     78H

PRUEBAP PROC    NEAR USES AX

    .REPEAT
        IN    AL,ESTADO
    .UNTIL AL & 8
    RET

PRUEBAP ENDP

```

Se puede utilizar un doble búfer con los datos impresos si se carga primero el búfer 1 con los datos a imprimir. Después se hace una llamada al procedimiento IMPRIME para comenzar a imprimir el búfer 1. Como se requiere muy poco tiempo para programar el controlador de DMA, puede llenarse un segundo búfer (búfer 2) con los nuevos datos para la impresora mientras se imprime el primer búfer (búfer 1) a través de la interfaz de impresora y el controlador de DMA. Este proceso se repite hasta que se imprimen todos los datos.

13-3

OPERACIÓN CON BUS COMPARTIDO

Los complejos sistemas computacionales de la actualidad tienen tantas tareas por realizar, que algunos sistemas utilizan más de un microprocesador para realizar su trabajo. A esto se le llama un sistema de **multiprocesamiento**. Algunas veces también se le conoce como un sistema **distribuido**. Un sistema que realiza más de una tarea se llama sistema **multitareas**. En sistemas que contienen más de un microprocesador es necesario desarrollar y emplear algún método de control. En un entorno multitarea, multiprocesamiento y distribuido, cada microprocesador tiene acceso a dos buses: (1) el **bus local** y (2) el **bus remoto o compartido**.

En esta sección hablaremos sobre la operación de bus compartido para los microprocesadores 8086 y 8088, mediante el uso del árbitro 8289. El 80286 utiliza el árbitro de bus 82289 y los microprocesadores 80386/80486 utilizan el árbitro de bus 82389. Los microprocesadores del Pentium al Pentium 4 soportan un entorno multiusuario de manera directa, como veremos en los capítulos 17, 18 y 19. Estos sistemas son mucho más complejos y difíciles de ilustrar en este punto del libro, pero su terminología y operación es en esencia la misma que para el 8086/8088.

El bus local se conecta a la memoria y a los dispositivos de E/S a los que se accede directamente mediante un solo microprocesador, sin ningún protocolo o reglas de acceso especiales. El bus remoto (compartido) contiene memoria y dispositivos de E/S a los que puede acceder cualquier microprocesador en el sistema. La figura 13-14 ilustra esta idea con unos cuantos microprocesadores. La computadora personal también está configurada de la misma forma que el sistema de la figura 13-14. El maestro de bus es el microprocesador principal en la computadora personal. En esta ilustración, el bus compartido es lo que conocemos como bus local en la computadora personal. El bus ISA se opera como esclavo para el microprocesador de la computadora personal, al igual que cualquier otro dispositivo conectado al bus compartido. El bus PCI puede operar como esclavo o como maestro.

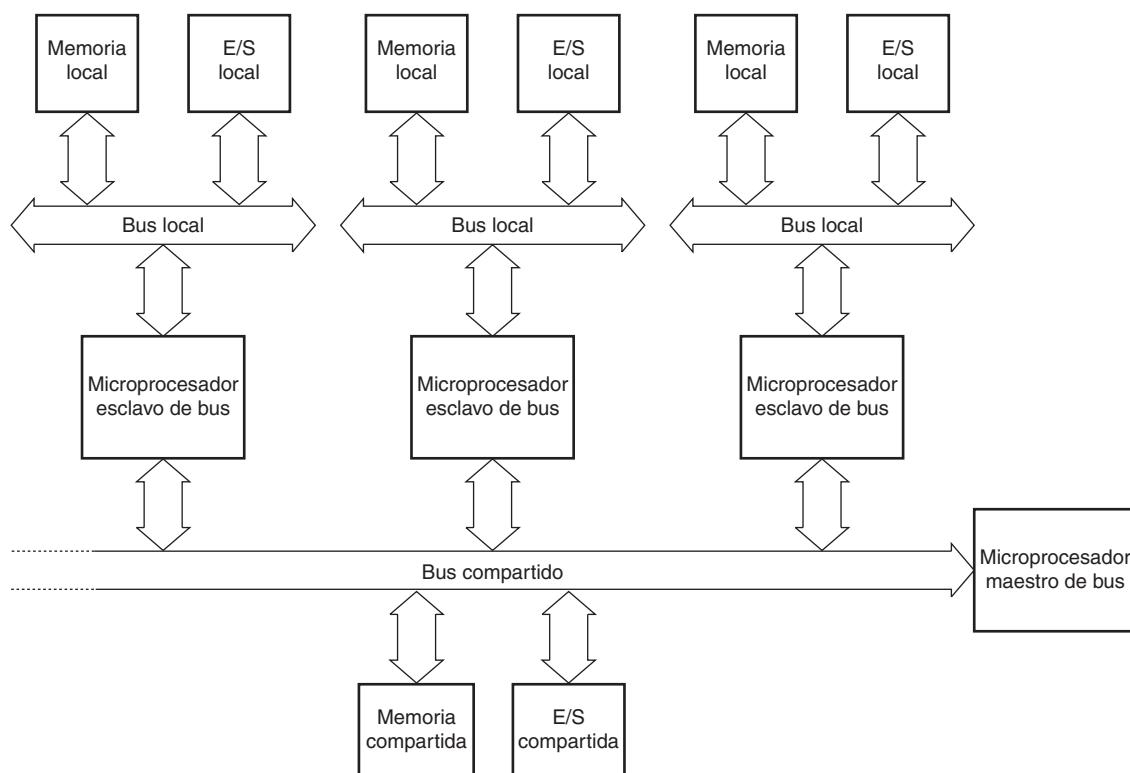


FIGURA 13-14 Un diagrama de bloques en el que se ilustran los buses compartido y local.

Tipos de buses definidos

El bus local es el bus residente para el microprocesador. El bus local contiene la memoria y los dispositivos de E/S residentes o locales. Todos los microprocesadores que hemos estudiado hasta ahora en este libro se consideran como sistemas de bus local. El microprocesador que está conectado en forma directa a la memoria local y a la E/S local es el que tiene acceso a ellos.

Un bus compartido está conectado a todos los microprocesadores en el sistema. El bus compartido se utiliza para intercambiar datos entre los distintos microprocesadores en el sistema. Un bus compartido puede contener memoria y dispositivos de E/S a los que accedan todos los microprocesadores en el sistema. El acceso al bus compartido se controla mediante cierta forma de árbito que sólo permite a un microprocesador el acceso al espacio del bus compartido del sistema. Como dijimos antes, el bus compartido en un sistema con varios microprocesadores viene siendo el bus local en la computadora personal, ya que es local para el microprocesador en este tipo de sistema computacional.

La figura 13-15 muestra un microprocesador 8088 conectado como maestro de bus remoto. El término **maestro de bus** se aplica a cualquier dispositivo (un microprocesador, por ejemplo) que puede controlar un bus que contenga memoria y E/S. El controlador de DMA 8237 que presentamos en secciones anteriores de este capítulo es un ejemplo de maestro de bus remoto. El controlador de DMA obtiene acceso a la memoria del sistema y al espacio de E/S para provocar una transferencia de datos. De igual forma, un maestro de bus remoto obtiene acceso al bus compartido para el mismo fin. La diferencia es que el microprocesador maestro de bus remoto puede ejecutar software variable, mientras que el controlador de DMA sólo puede transferir datos.

El acceso al bus compartido se obtiene mediante el uso de la terminal HOLD en el microprocesador para el controlador de DMA. El acceso al bus compartido para el maestro de bus remoto se obtiene mediante un **árbito de bus**, cuya función es resolver la prioridad entre los maestros de bus y sólo permite que un dispositivo a la vez tenga acceso al bus compartido.

Observe en la figura 13-15 que el microprocesador 8088 tiene una interfaz tanto para el bus residente local como para el bus compartido. Esta configuración permite al 8088 el acceso a la memoria y los dispositivos de E/S locales o, mediante el árbito de bus y los búferes, al bus compartido. La tarea asignada al microprocesador podría ser la comunicación de datos. Tal vez, después de recolectar un bloque de datos de la interfaz de comunicaciones, podría pasar esos datos al bus compartido y a la memoria compartida para que otros microprocesadores conectados al sistema puedan acceder a esos datos. Esto permite que muchos microprocesadores compartan datos comunes. De la misma forma, pueden asignarse varias tareas a múltiples microprocesadores en el sistema, con lo cual el rendimiento se mejora de una manera drástica.

El árbito de bus

Para poder comprender la figura 13-15 por completo, debemos conocer el funcionamiento del árbito de bus. El árbito de bus 8289 controla la interfaz entre un maestro de bus y un bus compartido. Aunque el 8289 no es el único árbito de bus está diseñado para funcionar con los microprocesadores 8086/8088; es por esta razón que lo veremos en esta sección. Cada maestro de bus o microprocesador requiere un árbito para la interfaz con el bus compartido; a esta interfaz Intel la llama Multibus e IBM la llama Microcanal.

El bus compartido se utiliza sólo para pasar información de un microprocesador a otro, en cualquier otro caso los maestros de bus funcionan en sus propios modos de bus local mediante el uso de sus propios programas, memoria y espacio de E/S locales. Por lo general, a los microprocesadores que se conectan en este tipo de sistema se les conoce como procesadores **paralelos o distribuidos**, ya que pueden ejecutar software y realizar tareas en paralelo.

Arquitectura del 8289. La figura 13-16 muestra el diagrama de terminales y de bloques del árbito de bus 8289. El lado izquierdo del diagrama de bloques describe las conexiones que van al microprocesador. El lado derecho denota la conexión del 8289 con el bus compartido (remoto), o Multibus.

Para controlar el bus compartido, el 8289 hace que la entrada READY que va al microprocesador se vuelva un 0 lógico (no está listo) si se niega el acceso al bus compartido. El **bloqueo** ocurre cada vez que otro microprocesador está utilizando el bus compartido. Como resultado, el microprocesador que solicita el acceso queda bloqueado por el 0 lógico que se aplica a su entrada READY. Cuando la

entrada READY es un 0 lógico, el microprocesador y su software esperan hasta que el árbitro otorgue el acceso al bus compartido. De esta forma, sólo un microprocesador a la vez puede obtener acceso al bus compartido. No se requieren instrucciones especiales para el arbitraje de bus con el 8289, ya que se realiza sólo mediante el hardware.

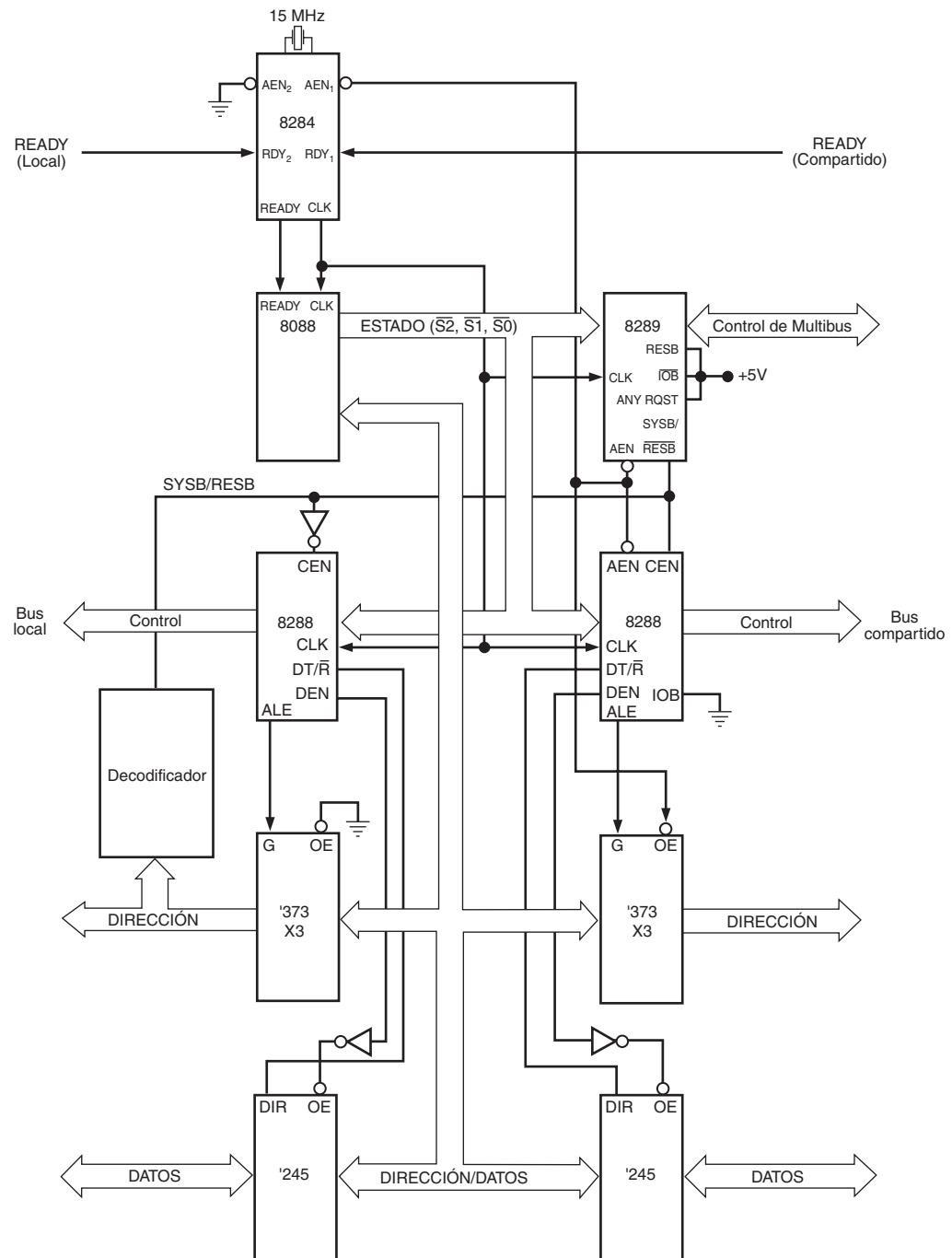
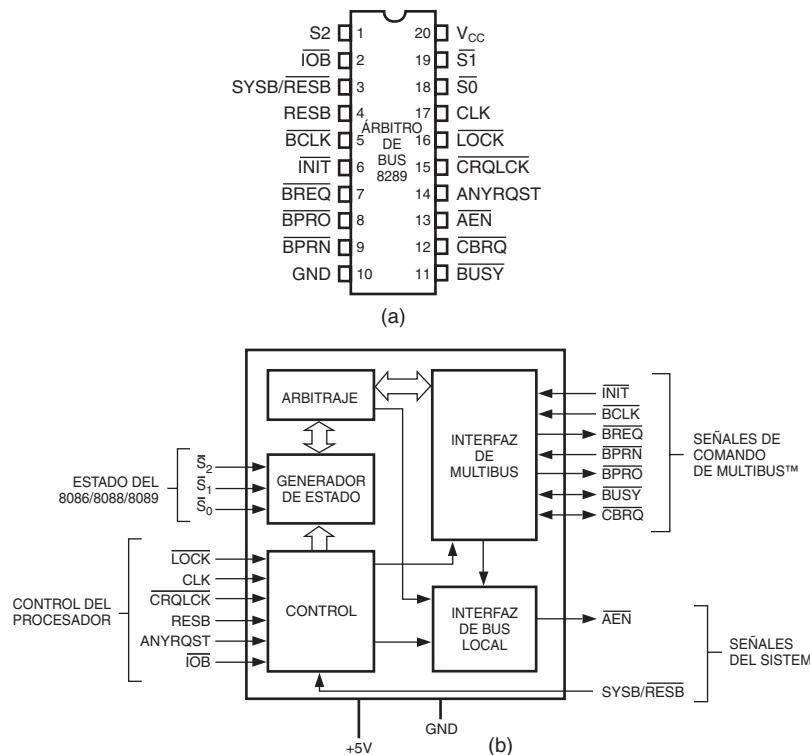


FIGURA 13-15 La operación del 8088 en modo remoto para ilustrar las conexiones de los buses local y compartido.

FIGURA 13-16

Diagrama de terminales y de bloques del 8289.
(Cortesía de Intel Corporation.)



Definiciones de terminales

- AEN** La salida **habilita dirección** hace que los controladores de bus en un sistema cambien a su estado de alta impedancia de tres estados.
- ANYRQST** La entrada **cualquier petición** es una opción de puenteo que evita que un microprocesador de menor prioridad obtenga el acceso al bus compartido. Si se conecta a un 0 lógico se produce el arbitraje normal y un microprocesador de menor prioridad puede obtener acceso al bus compartido si CBRQ también es un 0 lógico.
- BCLK** La entrada **reloj del bus** sincroniza todos los maestros del bus compartido.
- BPRN** La **entrada de prioridad del bus** permite al 8289 adquirir el bus compartido en el siguiente borde de caída de la señal BCLK.
- BPRO** La **salida de prioridad del bus** es una señal que se utiliza para resolver la prioridad en un sistema que contiene varios maestros de bus.
- BREQ** La salida **petición de bus** se utiliza para solicitar acceso al bus compartido.
- BUSY** La entrada/salida **ocupado** indica, como salida, que un 8289 ha adquirido el bus compartido. Como entrada BUSY se utiliza para detectar que otro 8289 ha adquirido el bus compartido.
- CBRQ** La entrada/salida **petición de bus común** se utiliza cuando un microprocesador de menor prioridad está solicitando el uso del bus compartido. Como salida, CBRQ se vuelve 0 lógico cada vez que el 8289 solicita el bus compartido y permanece en nivel bajo hasta que el 8289 obtiene acceso al bus compartido.
- CLK** La entrada **reloj** se genera mediante el generador de reloj 8284A y proporciona la fuente de sincronización interna para el 8289.
- CRQLCK** La entrada **bloque de petición común** evita que el 8289 entregue el bus compartido a cualquiera de los dispositivos 8289 en el sistema. Esta señal funciona en conjunto con la terminal CBRQ.

INIT	La entrada inicialización reinicia el 8289 y por lo general va conectada a la señal RE-SET del sistema.
IOB	La entrada bus de E/S selecciona si el 8289 opera en un sistema de bus compartido (si se selecciona mediante RESB) con E/S ($\overline{IOB} = 0$) o con memoria y E/S ($\overline{IOB} = 1$).
LOCK	La entrada bloqueo evita que el 8289 permita que cualquier otro microprocesador obtenga acceso al bus compartido. Una instrucción del 8086/8088 que contenga un prefijo LOCK evitará que otros microprocesadores accedan al bus compartido.
RESB	La entrada bus residente es una conexión de puenteo que permite al 8289 operar en sistemas que tienen un sistema de bus compartido o de bus residente. Si RESP es un 1 lógico, el 8289 se configura como maestro de bus compartido. Si RESB es un 0 lógico, el 8289 se configura como maestro de bus local. Cuando se configura como maestro de bus compartido, el acceso se solicita a través de la terminal de entrada SYSB/RESB.
S₀, S₁ y S₂	Las entradas estado inician peticiones y entregas del bus compartido. Estas terminales se conectan a las terminales de estado del controlador 8288 del bus del sistema.
SYSB/RESB	La entrada bus del sistema/bus residente selecciona el sistema de bus compartido cuando se coloca en un 1 lógico, o el bus local residente cuando se coloca en un 0 lógico.

Operación general del 8289. Como lo demuestran las descripciones de las terminales, el 8289 puede operarse en tres modos básicos: (1) modo de bus periférico de E/S, (2) modo de bus residente y (3) modo de bus individual. En la tabla 13-2 podrá ver las conexiones requeridas para operar el 8289 en estos modos. En el modo de bus periférico de E/S todos los dispositivos en el bus local se tratan como E/S, incluyendo la memoria, y pueden utilizarse con todas las instrucciones. Todas las referencias a memoria acceden al bus compartido y toda la E/S accede al bus local residente. El modo de bus residente permite accesos a memoria y a la E/S tanto en el bus local como en el bus compartido. Por último, el modo de bus individual sirve de interfaz entre un microprocesador y un bus compartido, sólo que el microprocesador no tiene memoria local ni E/S local. En muchos sistemas, un microprocesador se configura como el maestro de bus compartido (modo de bus individual) para controlar el bus compartido y convertirse en el maestro de bus compartido. Este maestro de bus compartido controla el sistema a través de la memoria y la E/S compartidas. Los microprocesadores adicionales se conectan al bus compartido como maestros de bus residente o de bus periférico de E/S. Por lo general, estos maestros de bus adicionales realizan tareas independientes, las cuales se reportan al maestro de bus compartido a través del bus compartido.

Sistema que ilustra las conexiones de bus individual y de bus residente. La operación en modo de bus individual actúa como interfaz entre un microprocesador y un bus compartido que contiene recursos de E/S y de memoria, los cuales se comparten entre otros microprocesadores. La figura 13-17 muestra tres microprocesadores 8088, cada uno de los cuales está conectado a un bus compartido. Dos de los tres microprocesadores operan en el modo de bus residente, mientras que el tercero opera en el modo de bus individual. El microprocesador A de la figura 13-17 opera en el modo de bus individual y no tiene bus local. Este microprocesador accede sólo a la memoria y el espacio de E/S compartidos. Por lo general, al microprocesador A se le conoce como maestro de bus del sistema debido a que es responsable

TABLA 13-2 Modos de operación del 8289.

Modo	Conexiones de terminales
Bus individual	$\overline{IOB} = 1$ y $\overline{RESB} = 0$
Bus residente	$\overline{IOB} = 1$ y $\overline{RESB} = 1$
Bus de E/S	$\overline{IOB} = 0$ y $\overline{RESB} = 0$
Bus de E/S y bus residente	$\overline{IOB} = 0$ y $\overline{RESB} = 1$

de coordinar las tareas de E/S y la memoria principal. Los otros dos microprocesadores (B y C) se conectan en el modo de bus residente, el cual les permite acceder tanto al bus compartido como a sus propios buses locales. Estos microprocesadores de bus residente se utilizan para realizar tareas que son independientes del maestro de bus del sistema. De hecho, la única vez que se interrumpe al maestro de bus del sistema de la realización de sus tareas es cuando uno de los dos microprocesadores de bus residente necesita transferir datos entre sí mismo y el bus compartido. Esta conexión permite a los tres microprocesadores realizar tareas en forma simultánea, además de que los datos pueden compartirse entre los microprocesadores cuando es necesario.

En la figura 13-17, el maestro de bus (A) permite al usuario operar con una terminal de vídeo que permite la ejecución de programas y en general controla el sistema. El microprocesador B se encarga

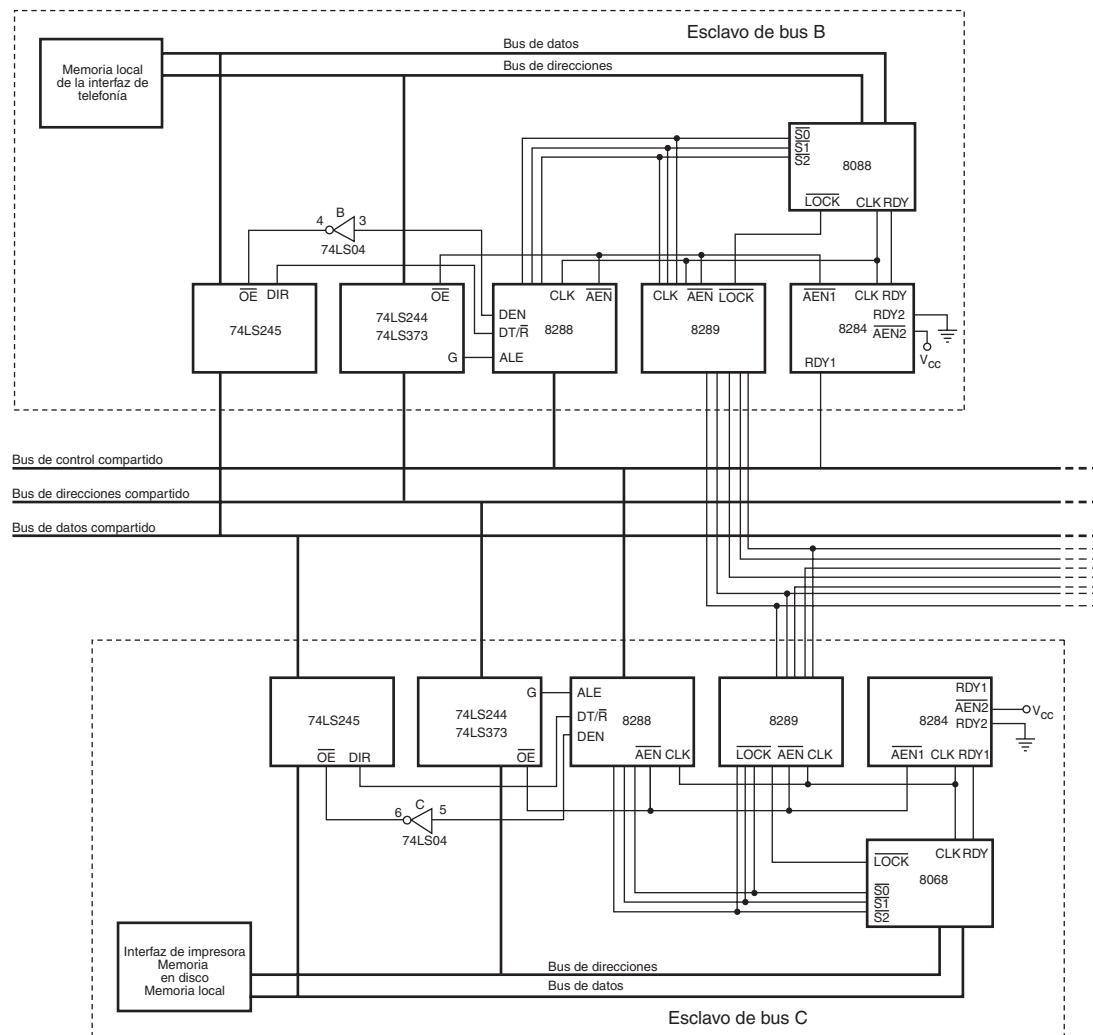


FIGURA 13-17 Tres microprocesadores 8088 que comparten un sistema de bus común. El microprocesador A es el maestro de bus que controla la memoria compartida y la terminal CRT. El microprocesador B es un esclavo de bus que controla su interfaz de telefonía local y la memoria. El microprocesador C también es un esclavo que controla una impresora, el sistema de memoria en disco y la memoria local.

de todas las comunicaciones telefónicas y pasa esta información a la memoria compartida en bloques. Esto significa que el microprocesador B espera a que cada carácter se transmita o se reciba y controla el protocolo utilizado para las transferencias. Por ejemplo, suponga que se transmite un bloque de datos de 1 Kbyte a través de la interfaz telefónica, a una velocidad de 100 caracteres por segundo. Esto significa que la transferencia requiere 10 segundos. En vez de retener el maestro de bus durante 10 segundos, con paciencia el microprocesador B realiza la transferencia de datos desde su propia memoria local y la interfaz de comunicaciones local. Esto libera al maestro de bus para que realice otras tareas. El único momento en que el microprocesador B interrumpe al maestro de bus es para transferir datos entre la memoria compartida y su sistema de memoria local. Esta transferencia de datos entre el microprocesador B y el maestro de bus requiere sólo de unos cuantos cientos de microsegundos.

El microprocesador C se utiliza como cola de impresión. Su única tarea es imprimir datos en la impresora. Cada vez que el maestro de bus requiere de datos de salida impresos, transfiere la tarea al microprocesador C. Éste accede a la memoria compartida, captura los datos a imprimir y los almacena en su propia memoria local. Después los datos se imprimen desde la memoria local, liberando al maestro de bus para que realice otras tareas. Esto permite al sistema ejecutar un programa con el maestro de bus, transferir datos a través de la interfaz de comunicaciones con el microprocesador B e imprimir información en la impresora con el microprocesador C. Todas estas tareas se ejecutan de manera simultánea. No hay un límite en cuanto al número de microprocesadores que se pueden conectar a un sistema, o en cuanto al número de tareas que se realizan de manera simultánea mediante el uso de esta técnica. El único límite es el que se introduce debido al diseño del sistema y la ingenuidad del diseñador. La empresa Lawrence Livermore Labs en California tiene un sistema que contiene 4096 microprocesadores Pentium.

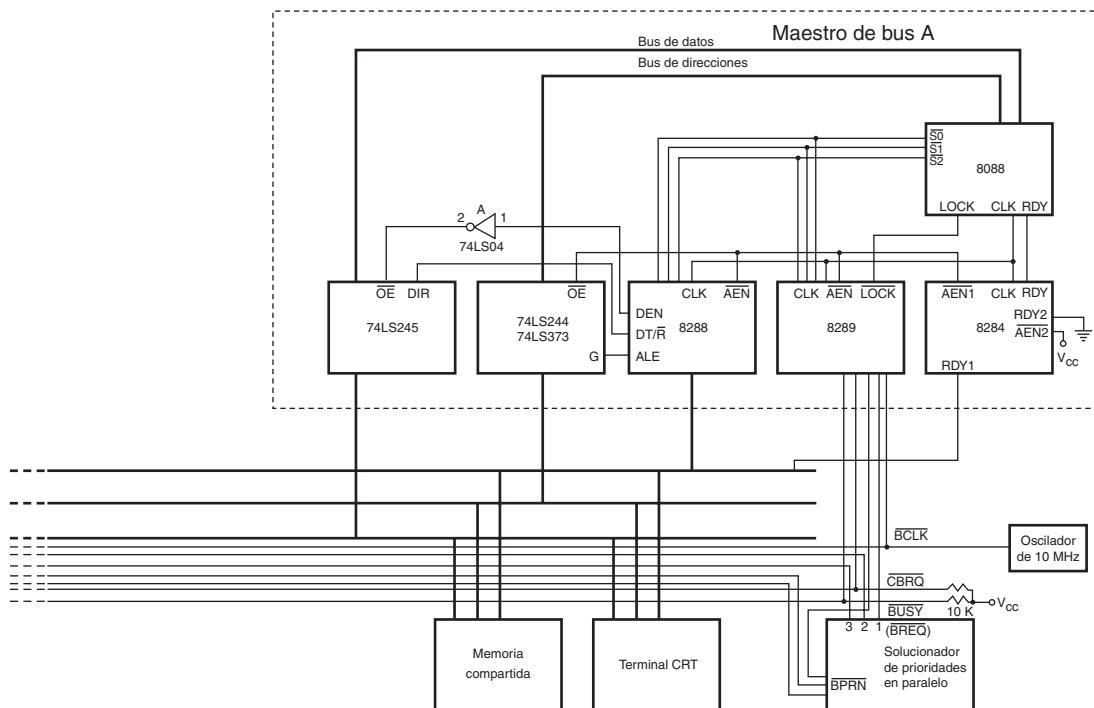


FIGURA 13-17 (continuación)

13-4

SISTEMAS DE MEMORIA EN DISCO

La memoria en disco se utiliza para almacenar datos a largo plazo. Hay muchos tipos de sistemas de almacenamiento en disco disponibles actualmente, los cuales utilizan medios magnéticos con la excepción de la memoria en disco óptico, la cual almacena datos en un disco de plástico. La memoria en disco óptico puede ser un **CD-ROM** (disco compacto/memoria de sólo lectura) que se puede leer pero no escribir, o un **WORM** (se escribe una vez/puede leerse muchas veces) que la mayor parte del tiempo se lee, pero puede escribirse en él una vez mediante un rayo láser. También se está dando a conocer la memoria en disco óptico, en el que se puede leer y escribir muchas veces, pero aún hay un límite en cuanto al número de operaciones de escritura permitidas. La tecnología de disco óptico más reciente se conoce como **DVD** (**disco versátil digital**). En esta sección del capítulo veremos una introducción a los sistemas de memoria en disco, de manera que podamos utilizarlos con sistemas computacionales. También proporcionaremos detalles sobre su operación.

Memoria en disco flexible

Una de las formas más comunes y básicas de memoria en disco es el floppy, o disco flexible. Hoy en día el disco flexible está comenzando a desaparecer y tal vez llegue a hacerlo por completo en muy poco tiempo, gracias a la unidad de almacenamiento portátil USB. Los medios de grabación magnéticos del disco flexible están disponibles en tres tamaños: **8" estándar**, **5½" minifloppy** y **3½" microfloppy**. Actualmente, la versión estándar de 8" y el minifloppy de 5½" han desaparecido por completo, dejando el camino libre a los discos tipo microfloppy y, hasta hace poco, a las unidades de almacenamiento portátil USB. El disco de 8" es demasiado grande, difícil de manejar y de almacenar. Para resolver este problema, la industria desarrolló el disco minifloppy de 5½". En la actualidad, el disco microfloppy ha reemplazado al minifloppy en los sistemas más recientes debido a su reducido tamaño, su facilidad de almacenamiento y su resistencia. Por ello, los sistemas siguen comercializándose con las unidades de disco microfloppy.

Todos los discos, incluyendo las unidades de almacenamiento portátil USB, tienen varias cosas en común. Todos están organizados de manera que los datos se almacenen en pistas. Una **pista** es un anillo concéntrico de datos que se almacenan en la superficie de un disco. La figura 13-18 ilustra la

FIGURA 13-18 El formato de un disco minifloppy de 5½".

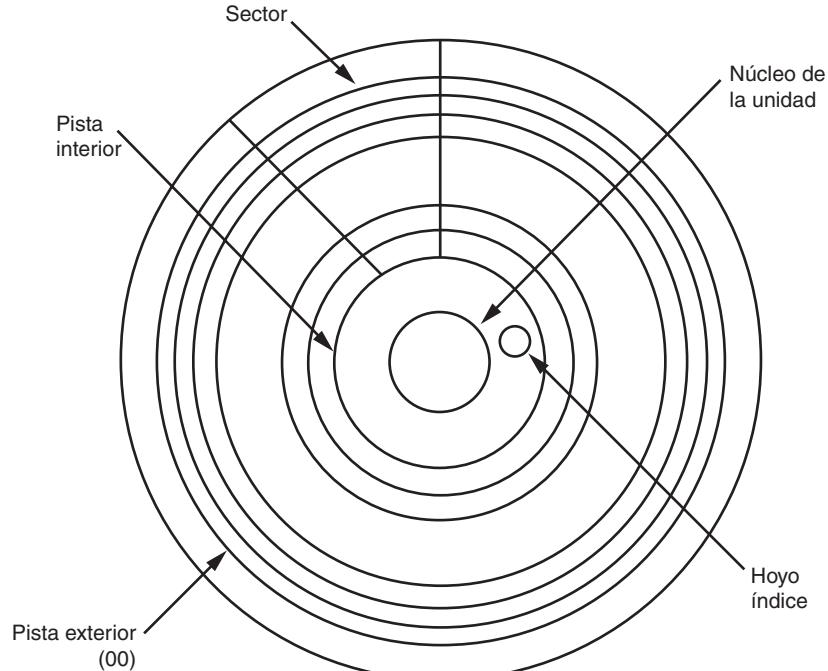
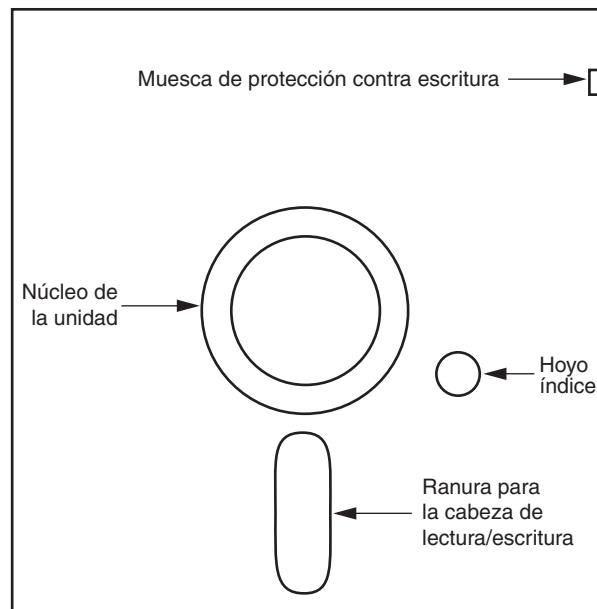


FIGURA 13-19 El disco minifloppy de 5¼".



superficie de un disco minifloppy de 5¼", en donde se muestra una pista dividida en sectores. Un **sector** es una subdivisión común de una pista, el cual está diseñado para almacenar una cantidad razonable de datos. En muchos sistemas, un sector almacena 512 o 1024 bytes de datos. El tamaño de un sector puede variar desde 128 bytes hasta la longitud de toda una pista completa.

En la figura 13-18 podrá observar que hay un hoyo a través del disco, identificado como un **hoyo índice**. Este hoyo índice está diseñado de manera que el sistema electrónico que lee el disco pueda encontrar el comienzo de una pista y su primer sector (00). Las pistas están numeradas desde la pista 00, la más externa, y el valor va aumentando hacia el centro del disco, hasta llegar a la pista más interna. Por lo general, los sectores se enumeran desde el sector 00, que está en la pista más externa, hasta el valor que se requiera para llegar a la pista más interna y a su último sector.

El disco minifloppy de 5¼". Hoy en día, el disco flexible de 5 ¼" es muy difícil de encontrar y se utiliza sólo con sistemas microcomputacionales antiguos. La figura 13-19 muestra una ilustración de este disco minifloppy. El disco flexible gira a 300 RPM dentro de su cubierta de plástico semirrígido. El mecanismo de la cabeza en una unidad de disco flexible hace contacto físico con la superficie del disco, lo cual con el tiempo produce desgaste y daños al disco.

La mayoría de los discos minifloppy son de doble lado. Esto significa que los datos se escriben en las superficies superior e inferior del disco. Un conjunto de pistas, al cual se le conoce como **cilindro**, consiste en una pista superior y una pista inferior. Por ejemplo, el cilindro 00 consiste en las pistas superior e inferior más externas.

Los datos en disco flexible se almacenan en el formato de doble densidad, el cual utiliza una técnica de grabación conocida como MFM (**modulación de frecuencia modificada**) para almacenar la información. En general, los discos de doble lado, doble densidad (DSDD) se organizan con 40 pistas de datos en cada lado del disco. Por lo común, la pista de un disco de doble densidad se divide en nueve sectores, en donde cada sector contiene 512 bytes de información. Esto significa que la capacidad total de un disco de doble lado, doble densidad es de 40 pistas por lado × 2 lados × 9 sectores por pista × 512 bytes por sector, o de 368,640 (360 K) bytes de información.

Los discos minifloppy de **alta densidad** (HD) también son comunes. Un disco minifloppy de alta densidad contiene 80 pistas de información por cada lado, con ocho sectores por pista. Cada sector contiene 1024 bytes de información. Esto proporciona al disco minifloppy de 5¼" y alta densidad una capacidad total de 80 pistas por lado × 2 lados × 15 sectores por pista × 512 bytes por sector, o 1,228,800 (un valor aproximado de 1.2 M) bytes de información.

La técnica de grabación magnética que se utiliza para almacenar datos en la superficie del disco se conoce como grabación **sin retorno a cero** (NRZ). Con este tipo de grabación, el flujo magnético

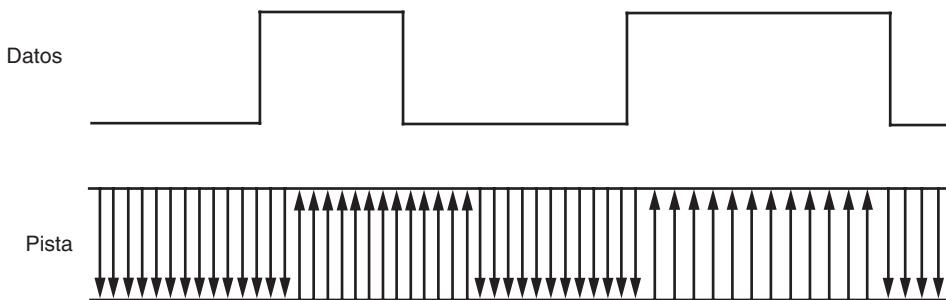


FIGURA 13-20 La técnica de grabación sin retorno a cero (NRZ).

que se coloca en la superficie del disco nunca regresa a cero. La figura 13-20 muestra la información almacenada en una porción de una pista. También muestra cómo el campo magnético codifica los datos. Observe que las flechas se utilizan en esta ilustración para mostrar la polaridad del campo magnético almacenado en la superficie del disco.

La razón principal por la que se eligió esta forma de codificación magnética es que borra de manera automática la información antigua cuando se graba nueva información. Si se utilizara otra técnica, se requeriría una cabeza separada para borrar información. La alineación mecánica de una cabeza de borrado separada y una cabeza de lectura/escritura separada es casi imposible. La densidad del flujo magnético de la señal NRZ es tan intensa que satura (magnetiza) por completo la superficie del disco, borrando todos los datos anteriores. También asegura que la información no se vea afectada por el ruido, debido a que la amplitud del campo magnético no contiene información. Ésta se almacena en la colocación de los cambios del campo magnético.

En los sistemas de disco flexible modernos, los datos se almacenan en forma de MFM (modulación de frecuencia modificada). La técnica de grabación MFM almacena datos en la forma que se muestra en la figura 13-21. Observe que el tiempo de cada bit es de 2.0 μ s en un disco de doble densidad. Esto significa que los datos se graban a la velocidad de 500,000 bits por segundo. El tiempo de cada bit de 2.0 μ s se divide en dos partes: una parte está designada para almacenar un pulso de reloj y la otra guarda un pulso de datos. Si hay un pulso de reloj presente, es de 1.0 μ s, al igual que un pulso de datos. Los pulsos de reloj y de datos nunca están presentes al mismo tiempo en un periodo de un bit. (Tenga en cuenta que las unidades de disco de alta densidad reducen estos tiempos a la mitad, de manera que el tiempo de un bit es de 1.0 μ s y un pulso de reloj o de datos es de 0.5 μ s. Esto también duplica la velocidad de transferencia a 1 millón de bits por segundo, o 1 Mbps.)

Si hay un pulso de datos presente, el tiempo del bit representa un 1 lógico. Si no hay datos ni reloj presente, el tiempo del bit representa un 0 lógico. Si hay un pulso de reloj presente sin pulso de datos, el tiempo del bit también representa un 0 lógico. Las reglas a seguir cuando se almacenan datos mediante el uso de MFM son:

1. Un pulso de datos siempre se almacena para un 1 lógico.
2. No se almacenan datos ni reloj para el primer 0 lógico en una cadena de 0 lógicos.
3. El segundo 0 lógico y todos los que le sigan en una fila contendrán un pulso de reloj, pero sin pulso de datos.

La razón por la cual se inserta un pulso de reloj como el segundo cero (y los subsiguientes) en una fila es para mantener la sincronización a medida que se leen datos del disco. La electrónica que se

FIGURA 13-21 Uso de la modulación de frecuencia modificada (MFM) con la memoria en disco.

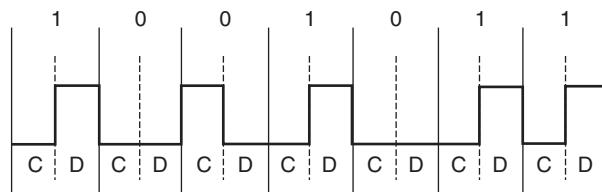
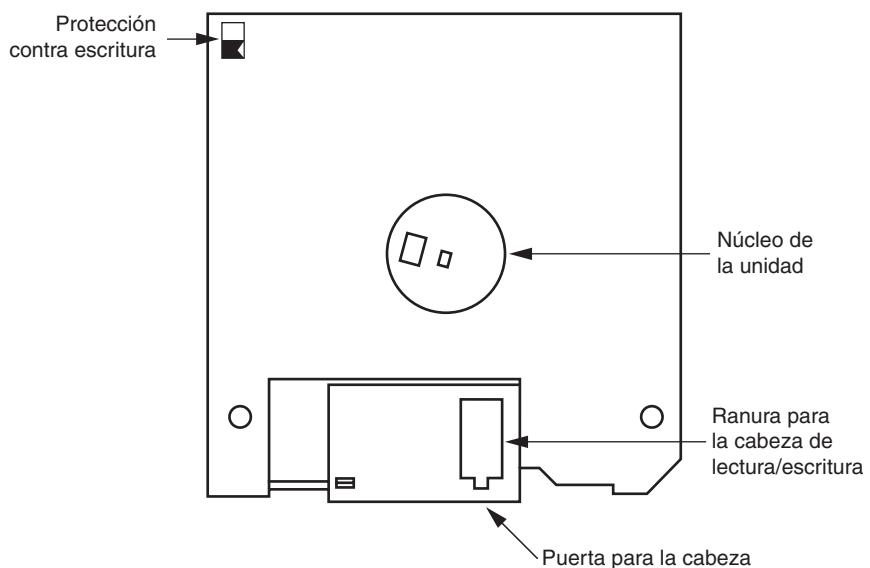


FIGURA 13-22 El disco microfloppy de 3½".



utiliza para recapturar los datos de la unidad de disco utiliza un ciclo de bloqueo de fase para generar un reloj y una ventana de datos. El ciclo de bloqueo de fase necesita un reloj o datos para mantener la operación sincronizada.

El disco microfloppy de 3½". Uno de los tamaños de disco más populares es el disco microfloppy de 3½". Hasta hace poco, este tamaño de disco flexible ha comenzado a ser reemplazado por la unidad de almacenamiento portátil USB como el medio portátil dominante. El disco microfloppy es una versión mucho más mejorada del disco minifloppy que describimos en la sección anterior. La figura 13-22 muestra el disco microfloppy de 3½".

Los diseñadores de discos observaron varias desventajas del minifloppy (que es una versión a escala del floppy estándar de 8") poco después de empezar a comercializarlo. Tal vez uno de los problemas más grandes con el minifloppy sea que viene empaquetado en una cubierta de plástico rígido que no se dobla con facilidad. Esto proporciona un grado mucho mayor de protección para el disco dentro de la cubierta.

Otro problema con el minifloppy es la ranura para la cabeza, la cual expone de manera continua la superficie del disco a los contaminantes. Este problema también se corrigió en el microfloppy ya que está construido con una puerta deslizante operada por resorte. La puerta para la cabeza permanece cerrada hasta que el disco se inserta en la unidad. Una vez dentro de la unidad, el mecanismo abre la puerta y expone la superficie del disco a las cabezas de lectura/escritura. Esto proporciona un buen grado de protección para la superficie del disco microfloppy.

Otra de las mejoras en el disco microfloppy es el mecanismo de protección de plástico deslizable. En el disco minifloppy se colocaba un pedazo de cinta sobre una muesca al lado de la cubierta para evitar la escritura. Esta cinta de plástico se despegaba con facilidad dentro de las unidades de disco, provocando problemas. En el microfloppy, un plástico deslizable integrado sustituye al mecanismo de protección de escritura de cinta. Para proteger el disco microfloppy contra la escritura, el plástico se desliza para abrir el hoyo que pasa a través de la cubierta del disco. Esto permite que la luz llegue a un sensor que inhibe la escritura.

Otra mejora más es el reemplazo del hoyo índice con un mecanismo de control distinto. El mecanismo de la unidad en el disco minifloppy permite sujetar el disco en cualquier punto. Esto requiere un hoyo índice para que la electrónica pueda encontrar el comienzo de una pista. El hoyo índice es otro generador de problemas, ya que recolecta suciedad y polvo. El microfloppy tiene un mecanismo de control tipo llave, de manera que sólo puede introducirse de una forma en la unidad de disco. El hoyo índice ya no es necesario debido a este mecanismo tipo llave. Gracias al mecanismo de la puerta deslizable para la cabeza y debido al hecho de que no existe un hoyo índice, el disco microfloppy no tiene un lugar en el que pueda acumularse polvo o suciedad.

Hay dos tipos de discos microfloppy disponibles en casi cualquier parte: el disco de doble lado, doble densidad (DSDD) y el disco de alta densidad (HD). El disco microfloppy de doble lado, doble densidad tiene 80 pistas por lado y cada pista contiene nueve sectores. Cada sector contiene 512 bytes de información. Esto permite almacenar $80 \text{ pistas} \times 2 \text{ lados} \times 9 \text{ sectores} \times 512 \text{ bytes por sector}$, o 737,280 (720 K) bytes de datos en un disco flexible de doble densidad, doble lado.

El disco microfloppy de alta densidad, de doble lado almacena aún más información. La versión de alta densidad tiene 80 pistas por lado, pero el número de sectores se duplica a 18 por pista. Este formato aún utiliza 512 bytes por sector, como el formato de doble densidad. El número total de bytes en un disco microfloppy de alta densidad, doble lado es de $80 \text{ pistas} \times 2 \text{ lados} \times 18 \text{ sectores por pista} \times 512 \text{ bytes por sector}$, o 1,474,560 (1.44 M) bytes de información.

Unidades de almacenamiento portátil USB

Estas unidades, también conocidas como unidades Flash, sustituyen a las unidades de disco flexible y utilizan memoria Flash para almacenar datos. Un controlador, que forma parte de Windows (excepto en Windows 98), trata a la unidad de almacenamiento portátil como un disco flexible con pistas y sectores, aún y cuando en realidad no contiene pistas ni sectores. Cuando se conecta una unidad de almacenamiento portátil al bus USB, el sistema operativo la reconoce y permite la transferencia de datos entre ésta y la computadora.

Las unidades de almacenamiento portátiles más recientes utilizan la especificación de bus USB 2.0 para transferir datos a una velocidad mucho mayor que la especificación USB 1.1 anterior. Las velocidades de transferencia para la especificación USB 1.1 son una velocidad de lectura de 750 KBps y una velocidad de escritura de 450 KBps. Las unidades USB 2.0 tienen una velocidad de transferencia aproximada de 48 MBps. En la actualidad, la unidad de almacenamiento portátil está disponible en tamaños de hasta 1 Gbyte y tiene un ciclo de borrado de hasta 1,000,000 de veces. El precio es bastante razonable si se le compara con el disco flexible.

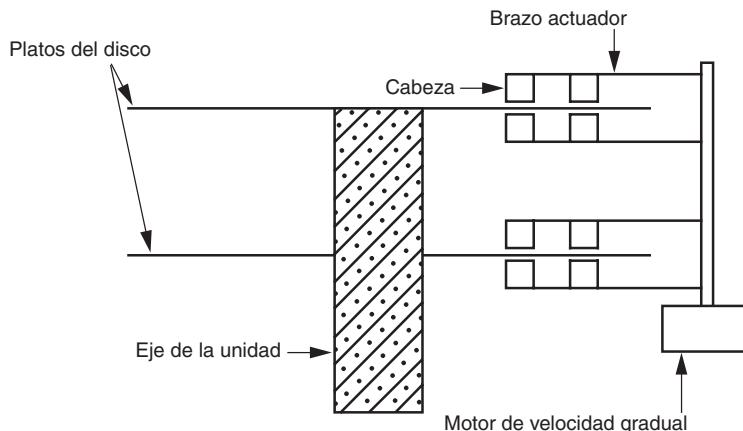
Memoria en disco duro

La memoria en disco más grande está disponible en forma de **unidad de disco duro**. Con frecuencia, a esta unidad se le conoce como **disco fijo** ya que no es removible como el disco flexible. También se le llama **disco rígido**. El término **unidad Winchester** se utilizaba antes para describir a un disco duro, pero en la actualidad es poco común. La memoria en disco duro tiene una capacidad mucho mayor que la memoria en disco flexible. La memoria en disco duro está disponible en tamaños que están muy cerca de llegar a 1T (tera) byte de datos. Actualmente los tamaños comunes de bajo costo (menos de \$1 USD por gigabyte) son de 20 Gbytes a 300 Gbytes.

Existen varias diferencias entre la memoria en disco flexible y la memoria en disco duro. Esta última utiliza una cabeza suspendida para almacenar y leer datos de la superficie del disco. Una cabeza suspendida, que es muy pequeña y ligera, no toca la superficie del disco. Vuela por encima de la superficie en una pelcula de aire que se lleva con la superficie del disco a medida que este gira. Las velocidades típicas de rotación de un disco duro están entre 3000 y 15,000 RPM, lo cual es muchas veces más rápido que el disco flexible. Esta alta velocidad de rotación permite a la cabeza volar (así como vuela un aeroplano) justo por encima de la superficie del disco. Ésta es una característica importante, ya que no hay desgaste en la superficie del disco duro, como en el disco flexible.

No obstante, pueden surgir problemas debido a las cabezas voladoras. Uno de ellos es cuando la cabeza se estrella. Si se interrumpe de manera repentina la energía o si el disco duro se golpea, la cabeza puede estrellarse en la superficie del disco, lo cual puede dañar la superficie o la cabeza. Para ayudar a evitar los estrellamientos, algunos fabricantes de discos duros incluyen un sistema que estaciona la cabeza de manera automática cuando se interrumpe la energía. Este tipo de unidad de disco duro cuenta con cabezas que se estacionan de manera automática. Cuando las cabezas se estacionan, se mueven hacia una zona de aterrizaje seguro (una pista no utilizada) cuando se desconecta la energía. Algunas unidades no estacionan sus cabezas de manera automática; por lo general requieren un programa que estaciona las cabezas en la pista más interna antes de desconectar la energía. La pista más interna es un área segura de aterrizaje ya que es la última pista que se llena en la unidad de disco duro. En este tipo de unidad, el operador es responsable de estacionar las cabezas.

FIGURA 13-23 Una unidad de disco duro que utiliza cuatro cabezas por plato.



Otra diferencia entre una unidad de disco flexible y una unidad de disco duro es el número de cabezas y superficies de disco. Una unidad de disco flexible tiene dos cabezas, una para la superficie superior y otra para la inferior. La unidad de disco duro puede tener hasta ocho superficies de disco (cuatro platos) y hasta dos cabezas por superficie. Cada vez que se obtiene un nuevo cilindro al desplazar el conjunto de cabezas, hay 16 pistas nuevas disponibles debajo de ellas. En la figura 13-23 podrá ver la ilustración de un sistema de disco duro.

Las cabezas se desplazan de pista en pista mediante el uso de un motor de velocidad gradual o de una bobina. El motor de velocidad gradual es lento y ruidoso, mientras que el mecanismo de la bobina es silencioso y rápido. Para desplazar el conjunto de cabezas se requiere un paso por cada cilindro, en un sistema que utiliza un motor de velocidad gradual para posicionar las cabezas. En un sistema que utiliza una bobina, las cabezas pueden desplazarse muchos cilindros con un movimiento de barrido. Esto hace que la unidad de disco sea más rápida al buscar nuevos cilindros.

Otra ventaja del sistema de bobina es que un servomecánismo puede supervisar la amplitud de la señal que llega de la cabeza de lectura, para así realizar pequeños ajustes en la posición de las cabezas. Esto no es posible con un motor de velocidad gradual, el cual se basa sólo en mecanismos para posicionar la cabeza. A menudo, los mecanismos de posicionamiento de cabezas mediante motores de velocidad gradual pueden desalinearse con el uso, mientras que el mecanismo de bobina corrige cualquier desalineamiento.

Por lo general, las unidades de disco duro almacenan la información en sectores de 512 bytes. Los datos se direccionan en clústeres de ocho o más sectores, los cuales contienen 4096 bytes (o más) en la mayoría de las unidades de disco duro. Estas unidades utilizan la técnica MFM o RLL para almacenar información. La técnica MFM se describió con las unidades de disco flexible. Aquí describiremos la técnica de longitud de recorrido limitada (RLL).

Una típica unidad de disco duro MFM antigua utiliza 18 sectores por pista para que puedan almacenarse 18 Kbytes de datos por pista. Si una unidad de disco duro tiene una capacidad de 40 Mbytes, contiene un valor aproximado de 2280 pistas. Si la unidad de disco tiene dos cabezas, significa que contiene 1140 cilindros; si contiene cuatro cabezas entonces tiene 570 cilindros. Estas especificaciones varían de una unidad de disco a otra.

TABLA 13-3 Codificación RLL 2,7 estándar.

Flujo de datos de entrada	Salida RLL
000	000100
10	0100
010	100100
0010	00100100
11	1000
011	001000
0011	00001000

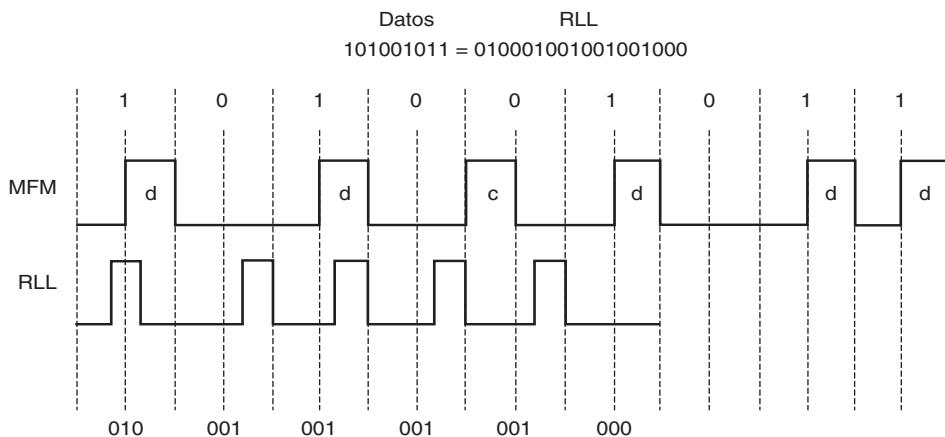


FIGURA 13-24 Comparación entre MFM y RLL mediante el uso de los datos 101001011.

Almacenamiento RLL. Las unidades de disco con **longitud de recorrido limitada** (RLL) utilizan un método distinto al de MFM para codificar los datos. El término RLL significa que el recorrido de ceros (los ceros en una fila) está limitado. Un esquema de codificación RLL común en uso hoy en día es RLL 2,7. Esto significa que el recorrido de ceros siempre está entre dos y siete. La tabla 13-3 muestra la codificación que se utiliza con el formato RLL estándar.

Los datos se codifican primero mediante el uso de la tabla 13-3 antes de enviarse a los componentes electrónicos de la unidad para almacenarlos en la superficie del disco. Debido a esta técnica de codificación, es posible lograr un aumento del 50% en el almacenamiento de datos en una unidad de disco, si se le compara con el formato MFM. La principal diferencia es que la unidad RLL a menudo contiene 27 pistas en vez de las 18 que se encuentran en la unidad MFM. (Algunas unidades RLL también utilizan 35 sectores por pista.)

En la mayoría de los casos, la codificación RLL no requiere modificaciones en los componentes electrónicos de la unidad o en la superficie del disco. La única diferencia es una ligera reducción en la anchura del pulso al usar RLL, para lo cual se pueden requerir partículas de óxido un poco más finas en la superficie del disco. Los fabricantes de discos prueban la superficie del disco y califican la unidad de disco como unidad certificada para MFM o certificada para RLL. Aparte de la calificación, no hay diferencia en la construcción de la unidad de disco ni en el material magnético que cubre la superficie de los discos.

La figura 13-24 muestra una comparación de datos MFM y datos RLL. Observe que la cantidad de tiempo (espacio) requerido para almacenar datos RLL se reduce en comparación con el formato MFM. Aquí, los datos 101001011 se codifican en ambos formatos (MFM y RLL) para poder comparar estos dos estándares. Observe que se ha reducido la anchura de la señal RLL para que puedan acomodarse tres pulsos en el mismo espacio que un pulso de reloj y un pulso de datos para MFM. Un disco MFM de 40 Mbytes puede almacenar 60 Mbytes de datos codificados mediante RLL. Además de poder almacenar más información, pueden realizarse operaciones de escritura y de lectura en la unidad RLL a una velocidad más alta.

Actualmente, todas las unidades de disco duro utilizan codificación RLL. Además, hay una variedad de interfaces de disco duro en uso. La más antigua es la interfaz ST-506, la cual utiliza datos MFM o RLL. Un sistema de disco que utiliza esta interfaz se conoce también como sistema de disco MFM o RLL. También hay estándares más recientes en uso hoy en día: ESDI, SCSI e IDE. Todos estos nuevos estándares utilizan LL, aún y cuando, por lo general, no lo consideran importante. La principal diferencia es la interfaz entre la computadora y la unidad de disco. El sistema IDE se está convirtiendo en la interfaz de memoria en disco duro estándar.

El sistema de **interfaz mejorada de discos pequeños** (ESDI), que ha desaparecido, es capaz de transferir datos entre sí mismo y la computadora a velocidades cercanas a 10 Mbytes por segundo. En comparación, una interfaz ST-506 puede alcanzar una velocidad de transferencia de 860 Kbytes por segundo.

La **interfaz de sistemas computacionales pequeños** (SCSI) también se encuentra en uso, ya que permite conectar hasta siete discos distintos u otras interfaces a la computadora, a través del mismo controlador de interfaz. La interfaz SCSI se encuentra en algunas computadoras tipo PC y también en el sistema Apple Macintosh. Una versión mejorada, SCSI-II, ha empezado a aparecer en algunos sistemas. En el futuro, tal vez esta interfaz sea remplazada por IDE en la mayoría de las aplicaciones.

Hoy en día, uno de los sistemas más comunes es el sistema de **electrónica de unidades integrada** (IDE), el cual incorpora el controlador de disco en la unidad de disco y la conecta al sistema anfitrión a través de un pequeño cable de interfaz. Esto permite conectar muchas unidades de disco a un sistema sin tener que preocuparse por los conflictos del bus o del controlador. Las unidades IDE se encuentran en los sistemas IBM PS-2 más recientes y en muchos clones. Incluso hasta los sistemas computacionales Apple están comenzando a incluir unidades IDE en vez de las unidades SCSI de las computadoras Apple anteriores. La interfaz IDE también es capaz de controlar otros dispositivos de E/S además del disco duro. Por lo general, esta interfaz también contiene una memoria caché de 256 K a 8 Mbytes para los datos de los discos. La caché agiliza las transferencias de disco. Los tiempos de acceso comunes para una unidad IDE son a menudo menores de 8 ms, mientras que el tiempo de acceso para un disco flexible es de casi 200 ms.

Algunas veces a la interfaz IDE se le llama ATA, un acrónimo para **conector de AT**. El sistema más reciente es la interfaz ATA serial, o SATA. Esta interfaz transfiere datos en serie a velocidades de 150 MBps, una velocidad mayor a cualquier interfaz IDE. Puede hacer esto debido a que el nivel de 1 lógico ya no es de 5.0 V. En la interfaz SATA, el nivel de 1 lógico es de 0.5 V, lo cual permite la transferencia de datos a velocidades más altas ya que se requiere menos tiempo para que la señal se eleve a 0.5 V de lo que se necesita para que se eleve a 5.0 V. Dentro de poco, las velocidades de esta interfaz podrán llegar a los 600 MBps.

Memoria en disco óptico

Por lo general, este tipo de memoria (vea la figura 13-25) se encuentra disponible en dos formas: el CD-ROM (disco compacto/memoria de sólo lectura) y el WORM (se escribe una vez/se lee muchas veces). El CD-ROM es el disco óptico de menor costo, pero sufre de la falta de velocidad. Los tiempos de acceso

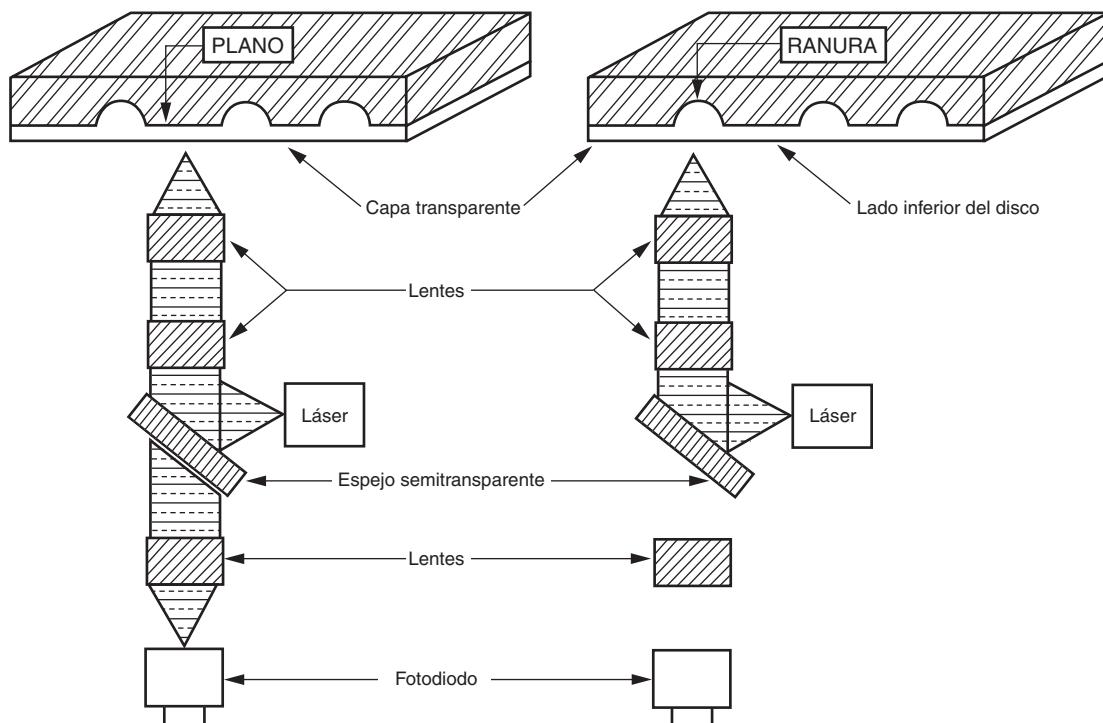


FIGURA 13-25 El sistema de memoria en CD-ROM óptico.

típicos para un CD-ROM son de 300 ms o más, casi el mismo valor que para un disco flexible. (Tenga en cuenta que hay dispositivos de CD-ROM más lentos, los cuales deben evitarse.) La memoria magnética en disco duro puede tener tiempos de acceso de hasta 11 ms. Un CD-ROM almacena 660 Mbytes de datos, o una combinación de datos y pasajes musicales. A medida que se desarrollen los sistemas y se vuelvan más activos en relación con lo visual, el uso de la unidad de CD-ROM se hará más común.

La unidad WORM tiene una aplicación mucho más comercial que el CD-ROM. El problema es que su aplicación es muy especializada debido a su naturaleza. Como los datos sólo se pueden escribir una vez, la principal aplicación es en la industria bancaria, en la industria de los seguros y en otras organizaciones de almacenamiento de datos masivos. El uso común del WORM es para formar un rastro de auditoría de transacciones que se envían mediante una cola al WORM y se recuperan sólo durante una auditoría. Al WORM se le podría llamar dispositivo de archivado.

Muchos sistemas de memoria en disco tipo WORM y de disco óptico de lectura/escritura se conectan al microprocesador mediante el uso de los estándares SCSI o ESDI que se utilizan con la memoria en disco duro. La diferencia es que las unidades de disco óptico actuales no son más rápidas que la mayoría de los discos flexibles. Algunas unidades de CD-ROM se conectan al microprocesador a través de interfaces propietarias que no son compatibles con otras unidades de disco.

La principal ventaja del disco óptico es su durabilidad. Como se utiliza un rayo láser de estado sólido para leer los datos del disco, y como el punto de enfoque es por debajo de una cubierta protectora de plástico, la superficie del disco puede leerse correctamente aunque contenga pequeños rayones y partículas de polvo. Esta característica permite que un disco óptico requiera de menos cuidado en comparación con un disco flexible. Casi la única forma de destruir datos en un disco óptico es si se rompe o se hacen rayones profundos en él.

La unidad de CD-ROM de lectura/escritura ya está disponible, y su costo está disminuyendo de manera considerable. En el futuro cercano empezaremos a ver cómo el CD-ROM de lectura/escritura reemplazará a las unidades de disco flexible. La principal ventaja es la gran capacidad de almacenamiento disponible en el CD-ROM de lectura/escritura. Muy pronto, el formato cambiará para que pueda haber muchos Gbytes de datos disponibles. El nuevo CD-ROM versátil de lectura/escritura, conocido como DVD, apareció en el mercado a finales de 1996 o a principios de 1997. El DVD funciona de igual forma que el CD-ROM, sólo que la densidad de bits es mucho más alta. El CD-ROM almacena 660 Mbytes de datos, mientras que el DVD actual almacena de 4.7 Gbytes a 9.4 Gbytes, dependiendo del estándar actual. Muy pronto el DVD sustituirá al formato de CD-ROM.

13-5

PANTALLAS DE VÍDEO

Las pantallas de vídeo modernas son dispositivos OEM (**fabricante de equipo original**) que por lo general se compran y se incorporan en un sistema. En la actualidad existen muchos tipos distintos de pantallas de vídeo disponibles en versiones a color o monocromáticas.

Las versiones monocromáticas, por lo general, muestran la información mediante el uso de pantallas en color ámbar, verde o blanco papel. Las pantallas en color blanco papel fueron alguna vez populares para muchas aplicaciones. Las aplicaciones más comunes de este tipo son el diseño gráfico (desktop publishing) y el dibujo asistido por computadora (CAD).

Las pantallas a color son más diversas y han sustituido por completo a la pantalla en blanco y negro. Hay sistemas de pantalla a color que aceptan la información como una señal de vídeo compuesto, en forma muy parecida a la televisión, como señales de niveles de voltaje TTL (0 o 5 V) y como señales análogas (0-0.7 V). Las pantallas de vídeo compuesto están desapareciendo, ya que la resolución disponible es demasiado baja. Hoy en día, muchas aplicaciones requieren gráficos de alta resolución que no pueden mostrarse en una pantalla de vídeo compuesto tal como un receptor de televisión en el hogar. Las primeras pantallas de vídeo compuesto se encontraban en los sistemas computacionales Commodore 64, Apple 2 y similares.

Señales de vídeo

La figura 13-26 muestra la señal que se envía a una pantalla de vídeo compuesto. Esta señal está compuesta de varias partes requeridas para este tipo de pantalla. Las señales que se ilustran representan

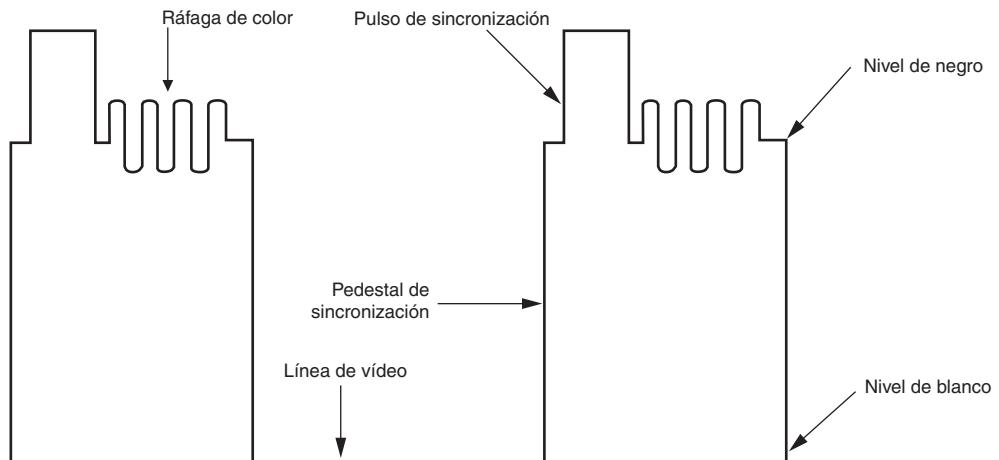


FIGURA 13-26 La señal de vídeo compuesto.

las señales que se envían a un monitor de vídeo compuesto a color. Observe que estas señales no sólo incluyen vídeo, sino también pulsos de sincronización, pedestales de sincronización y una ráfaga de color. Aquí no se muestra una señal de audio, ya que por lo general no existe. En vez de incluirlo con la señal de vídeo compuesto, el audio se desarrolla en la computadora y se produce mediante una bocina dentro del gabinete de la computadora. También puede desarrollarse mediante un sistema de sonido y reproducirse en sonido estereofónico en bocinas externas. Las principales desventajas de la pantalla de vídeo compuesto son la resolución y las limitaciones de color. Las señales de vídeo compuesto se diseñaron para emular las señales de vídeo de televisión, para que un receptor de televisión doméstico pudiera funcionar como monitor de vídeo.

La mayoría de los sistemas de vídeo modernos utilizan señales de vídeo directas que se generan mediante señales de sincronización separadas. En un sistema de vídeo directo, la información de vídeo se pasa al monitor a través de un cable que utiliza líneas separadas para el vídeo y también pulsos de sincronización. Recuerde que estas señales se combinaban en una señal de vídeo compuesto.

Un monitor monocromático (de un color) utiliza un cable para el vídeo, uno para la sincronización horizontal y uno para la sincronización vertical. A menudo, estos son los únicos cables de señales que se encuentran. Un monitor de vídeo a color utiliza tres señales de vídeo. Una señal representa el color rojo, otra el verde y la tercera el azul. Por lo general, a estos monitores se les conoce como monitores RGB debido a los colores primarios de la luz para el vídeo: rojo (R), verde (G) y azul (B).

El monitor RGB TTL

El monitor RGB está disponible como monitor analógico o TTL. El monitor RGB utiliza señales de nivel TTL (0 o 5 V) como entradas de vídeo y una cuarta línea llamada intensidad para permitir un cambio en la intensidad. La pantalla TTL de vídeo RGB puede mostrar un total de 16 colores distintos. El monitor RGB TTL se utiliza en el sistema CGA (**adaptador de gráficos a color**) que se encuentra en los sistemas computacionales antiguos.

La tabla 13-14 lista estos 16 colores y también las señales TTL presentes para generarlos. Ocho de los 16 colores se generan con intensidad alta y los otros ocho con intensidad baja. Los tres colores de vídeo son rojo, verde y azul. Éstos son los colores primarios de la luz. Los colores secundarios son cyan, magenta y amarillo. El cyan es una combinación de las señales de color azul y verde; su apariencia es de color azul-verdoso. Magenta es una combinación de las señales de vídeo azul y rojo; su apariencia es de color púrpura.

Los colores amarillo (alta intensidad) y café (baja intensidad) son una combinación de las señales de vídeo de color rojo y verde. Si se desean colores adicionales, por lo general no se utiliza el vídeo TTL. Hay un esquema que se desarrolló mediante el uso de señales de vídeo TTL de color bajo y medio,

TABLA 13-4 Los 16 colores de una pantalla TTL.

Intensidad	Rojo	Verde	Azul	Color
0	0	0	0	Negro
0	0	0	1	Azul
0	0	1	0	Verde
0	0	1	1	Cyan
0	1	0	0	Rojo
0	1	0	1	Magenta
0	1	1	0	Café
0	1	1	1	Blanco
1	0	0	0	Gris
1	0	0	1	Azul brillante
1	0	1	0	Verde brillante
1	0	1	1	Cyan brillante
1	1	0	0	Rojo brillante
1	1	0	1	Magenta brillante
1	1	1	0	Amarillo
1	1	1	1	Blanco brillante

con lo que se obtenían 32 colores, pero no tuvo mucha aplicación y nunca se encontró un uso amplio en el campo.

La figura 13-27 muestra el conector más común que se encuentra en el monitor RGB TTL o en un monitor monocromático TTL. El conector que se muestra es un conector de 9 terminales. Dos de las conexiones se utilizan para tierra, tres para vídeo, dos para señales de sincronización o retorno y una para la intensidad. Observe que la terminal 7 se identifica como vídeo normal. Ésta es la terminal que se utiliza en un monitor monocromático para la luminosidad, o señal de brillo. Los monitores TTL monocromáticos utilizan el mismo conector de 9 terminales que los monitores TTL RGB.

El monitor RGB analógico

Para poder mostrar más de 16 colores se requiere una pantalla de video analógico. A este tipo de pantallas se les conoce como monitores RGB analógicos. Estos monitores tienen todavía tres señales de entrada de video, pero no tienen la entrada de intensidad. Como las señales de video son analógicas en vez de señales TTL de dos niveles, se encuentran en cualquier nivel de voltaje entre 0.0 V y 0.7 V, lo cual permite mostrar un número infinito de colores. Esto se debe a que se puede generar un número infinito de niveles de voltaje entre los valores mínimo y máximo. En la práctica se genera un número finito de niveles. Los números más comunes son 256 K, 16 M o 24 M de colores, dependiendo del estándar.

La figura 13-28 muestra el conector que se utiliza para un monitor RGB analógico o monocromático analógico. Observe que el conector tiene 15 terminales y soporta pantallas analógicas RGB y monocromáticas.

FIGURA 13-27 El conector de 9 terminales de un monitor TTL.

DB9	Pin	Función
5 9 4 8 3 7 2 6 1	1	Tierra
	2	Tierra
	3	Vídeo rojo
	4	Vídeo verde
	5	Vídeo azul
	6	Intensidad
	7	Vídeo normal
	8	Retorno horizontal
	9	Retorno vertical

DB15	Pin	Función
	1	Vídeo rojo
8 5 7 4 6 3 5 2 4 0 2 9 1	2	Vídeo verde (vídeo monocromático)
	3	Vídeo azul
	4	Tierra
	5	Tierra
	6	Tierra roja
	7	Tierra verde (tierra monocromático)
	8	Tierra azul
	9	Se bloquea como llave
	10	Tierra
	11	Detección de color (tierra en un monitor a color)
	12	Detección monocromático (tierra en un monitor monocromático)
	13	Retorno horizontal
	14	Retorno vertical
	15	Tierra

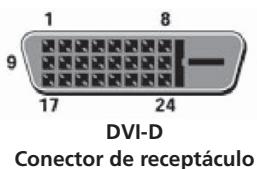
FIGURA 13-28 El conector de 15 terminales de un monitor analógico.

La manera en que se visualizan los datos en un monitor RGB analógico depende del estándar de interfaz utilizado con el monitor. La terminal 9 es una llave, lo cual significa que no existe un hoyo en el conector hembra para esta terminal.

Otro tipo de conector para el monitor RGB analógico que se está haciendo común es el conector DVI-D (interfaz visual digital). La -D es de digital y es la interfaz más común de este tipo. La figura 13-29 muestra el conector hembra en los nuevos monitores y tarjetas de vídeo.

La mayoría de las pantallas analógicas utilizan un convertidor de digital a analógico (DAC) para generar cada voltaje de vídeo de color. Un estándar común utiliza un DAC de ocho bits para cada señal de vídeo, para generar 256 niveles de voltaje distintos entre 0 V y 0.7 V. Existen 256 niveles de vídeo rojo distintos, 256 niveles de vídeo verde distintos y 256 niveles de vídeo azul distintos. Esto permite mostrar $256 \times 256 \times 256$, o 16,777,216 (16 M) colores.

La figura 13-30 muestra el circuito de generación de vídeo que se emplea en muchos estándares de vídeo comunes, como el poco popular EGA (**adaptador de gráficos mejorado**) y el VGA (**adaptador de gráficos variable**), según su uso en una IBM PC. Este circuito se utiliza para generar vídeo



ASIGNACIONES DE TERMINALES DEL CONECTOR SÓLO DIGITAL					
Terminal	Asignación de señal	Terminal	Asignación de señal	Terminal	Asignación de señal
1	Datos2-	9	Datos1-	17	Datos0-
2	Datos2+	10	Datos1+	18	Datos0+
3	Datos2/4 Protector	11	Datos1/3 Protector	19	Datos0/5 Protector
4	Datos4-	12	Datos3-	20	Datos5-
5	Datos4+	13	Datos3+	21	Datos5+
6	Reloj DDC	14	Energía +5V	22	Protección reloj
7	Datos DDC	15	Tierra (para +5V)	23	Reloj+
8	Sin conexión	16	Detección de conexión viva	24	Reloj-

FIGURA 13-29 La interfaz DVI-D en muchos monitores y tarjetas de vídeo recientes.

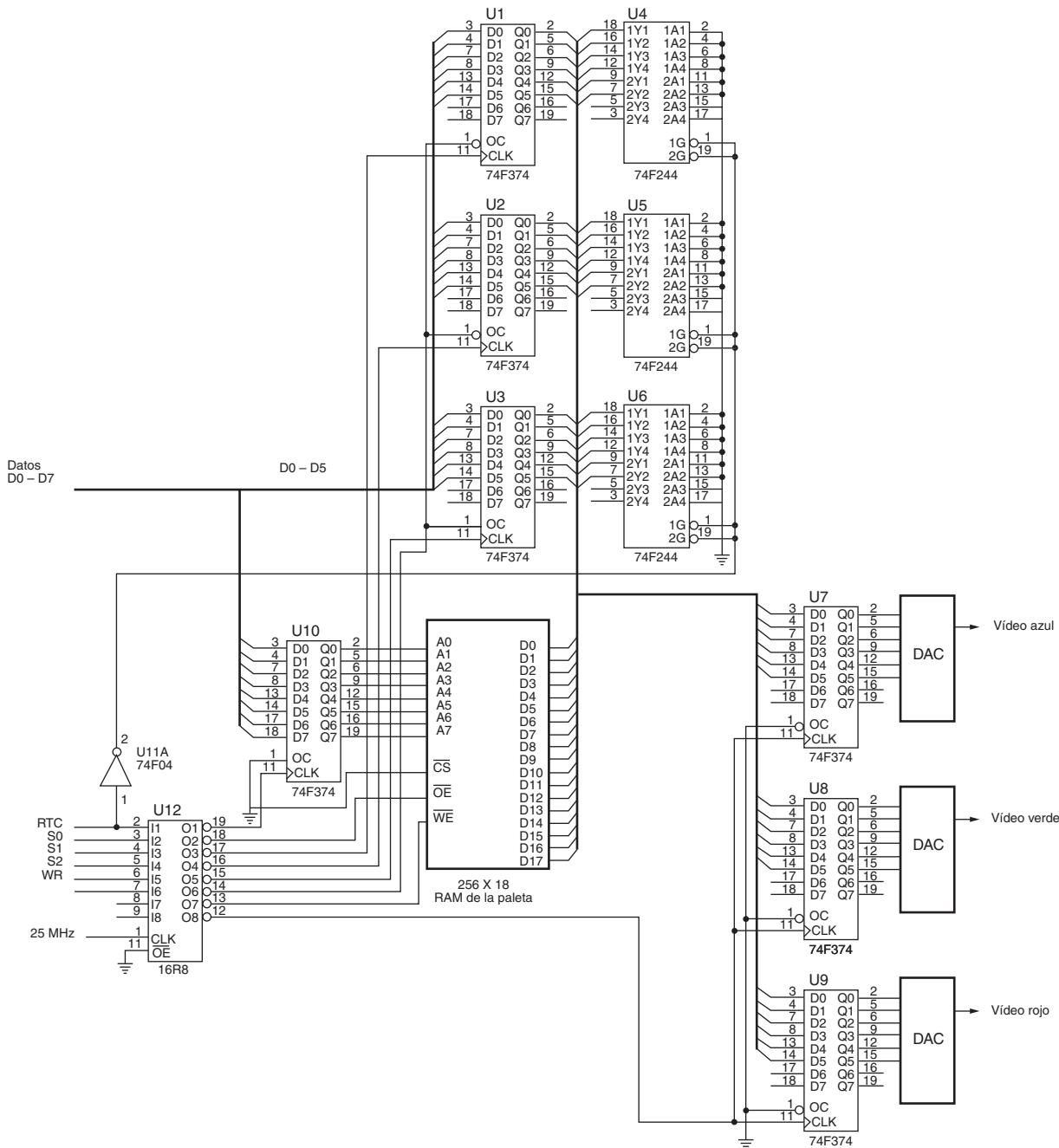


FIGURA 13-30 Generación de señales de vídeo VGA.

VGA. Observe que cada color se genera con un código digital de 18 bits. Seis de los 18 bits se utilizan para generar cada voltaje de color de vídeo cuando se aplican a las entradas de un DAC de ocho bits.

En este circuito se utiliza una SRAM de paleta de alta velocidad (con un tiempo de acceso menor de 40 ns) para almacenar 256 códigos de 18 bits distintos, los cuales representan 256 tonos distintos. Este código de 18 bits se aplica a los convertidores de digital a analógico. La entrada de dirección para la SRAM selecciona uno de los 256 colores que se almacenan como códigos binarios de 18 bits. Este sistema permite mostrar 256 colores a la vez, de una variedad de 256 K colores posibles. Para poder seleccionar cualquiera de los 256 colores, se utiliza un código de 8 bits que se almacena en la RAM de la

pantalla de vídeo de la computadora para especificar el color de un elemento de imagen. Si se utilizan más colores en un sistema, el código debe ser más amplio. Por ejemplo, un sistema que muestra 1024 colores a partir de 256 K colores requiere un código de 10 bits para direccionar la SRAM que contiene 1024 posiciones, cada una de las cuales contiene un código de color de 18 bits. Algunos sistemas más recientes utilizan una SRAM de paleta más grande para almacenar hasta 64 K de códigos de color distintos.

Cada vez que se coloca un color en la pantalla de vídeo, y si es que RTC es un 0 lógico, el sistema envía el código de 8 bits que representa un color a las conexiones D₀-D₇. Después el PLD genera un pulso de reloj para U₁₀, que enclava el código de color. Después de 40 ns (un ciclo de reloj de 25 MHz) el PLD genera un pulso de reloj para los enclavamientos del DAC (U₇, U₈ y U₉). La SRAM de paleta requiere esta cantidad de tiempo para buscar el contenido de 18 bits de la posición de memoria seleccionada por U₁₀. Una vez que se enclava el código de color (18 bits) en U₇-U₉, los tres DACS lo convierten en tres voltajes de vídeo para el monitor. Este proceso se repite para cada elemento de imagen (píxel) de 40 ns de amplitud que se muestre. El píxel tiene 40 ns de amplitud debido a que se utiliza un reloj de 25 MHz en este sistema. Puede obtenerse una resolución más alta si se utiliza una frecuencia de reloj más alta con el sistema.

Si se deben modificar los códigos de color (18 bits) que se almacenan en la SRAM, esto siempre se realiza durante el retorno cuando RTC es un 1 lógico. Esto evita que cualquier ruido de vídeo perturbe la imagen que se muestra en el monitor.

Para poder cambiar un color, el sistema utiliza las entradas S₀, S₁ y S₂ del PLD para seleccionar U₁, U₂, U₃ y U₁₀. Primero se envía la dirección del color que se va a modificar al enclavamiento U₁₀, el cual direcciona una posición en la SRAM de paleta. Después, cada uno de los nuevos colores de vídeo se cargan en U₁, U₂ y U₃. Por último, el PLD genera un pulso de escritura para la entrada \overline{WE} que va a la SRAM, para escribir el nuevo código de color en la SRAM de paleta.

En una pantalla de 640 × 480, el retorno ocurre 70.1 veces por segundo en dirección vertical y 31,500 veces por segundo en dirección horizontal. Durante el retorno, el voltaje de la señal de vídeo que se envía a la pantalla debe ser 0 V, para que se muestre el color negro durante el retorno. Este retorno se utiliza para desplazar el rayo de electrones hacia la esquina superior izquierda para el retorno vertical, y hacia el margen izquierdo de la pantalla para el retorno horizontal.

El circuito que se ilustra habilita los búferes U₄-U₆ para que cada uno de ellos aplique un 000000 al enclavamiento del DAC para el retorno. Los enclavamientos del DAC capturan este código y generan 0 V para que cada una de las señales de color de vídeo pongan la pantalla en blanco. Por definición, 0 V se considera el nivel de color negro para el vídeo y 0.7 V se considera una intensidad completa en una señal de color de vídeo.

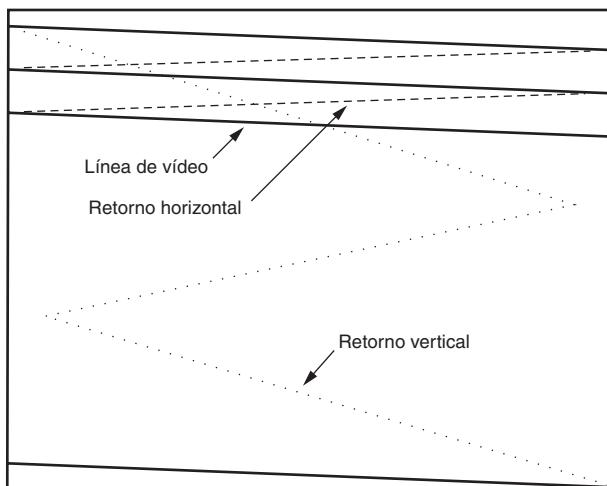
La resolución de la pantalla, por ejemplo 640 × 480, determina la cantidad de memoria requerida para la tarjeta de interfaz de vídeo. Si se utiliza esta resolución con una pantalla de 256 colores (ocho bits por píxel), entonces se requieren 640 × 480 bytes de memoria (307,200) para almacenar todos los pixeles para la pantalla. Es posible obtener pantallas de mayor resolución, pero como podría imaginar, se requiere aún más memoria. Una pantalla de 640 × 480 tiene 480 líneas de trama de vídeo y 640 pixeles por línea. Una **línea de trama** es la línea horizontal de información de vídeo que se muestra en el monitor. Un píxel es la subdivisión más pequeña de esta línea horizontal.

La figura 13-31 muestra la pantalla de vídeo, junto con las líneas de vídeo y el retorno. Hemos exagerado mucho el declive de cada una de las líneas de vídeo en esta ilustración, al igual que el espacio entre las líneas. Aquí se muestra el retorno en las direcciones vertical y horizontal. En el caso de una pantalla VGA, el retorno vertical ocurre 70.1 veces por segundo y el retorno horizontal ocurre 31,500 veces por segundo; ambos valores son exactos.

Para poder generar 640 pixeles a lo largo de una línea, se requieren 40 ns × 640, o 25.6 μ s. Un tiempo horizontal de 31,500 Hz permite un tiempo de línea horizontal de 1/31,500, o 31.746 μ s. La diferencia entre estos dos tiempos es el tiempo de retorno que se permite al monitor. (La Apple Macintosh tiene un tiempo de línea horizontal de 28.57 μ s.)

Como la velocidad de repetición del retorno vertical es de 70.1 Hz, el número de líneas que se generan se determina en base a la división del tiempo vertical entre el tiempo horizontal. En el caso de una pantalla VGA (una pantalla de 640 × 400), son 449.358 líneas. Sólo se utilizan 400 de esas líneas para mostrar información; el resto se pierde durante el retorno. Como se pierden 49.358 líneas durante el retorno, el tiempo que dura éste es de 49.358 × 31.766 μ s, o 1568 μ s. Durante esta cantidad “extensa”

FIGURA 13-31 Una pantalla de vídeo en la que se ilustran las líneas de trama y el retorno.



de tiempo es cuando la SRAM de paleta de colores se modifica, o cuando se actualiza el sistema de memoria de pantalla para una nueva pantalla de vídeo.

En la computadora Apple Macintosh (640×480), el número de líneas que se generan es de 525. Del número total de líneas, se pierden 45 durante el retorno vertical.

Otras resoluciones de pantalla son 800×600 y 1024×768 . La pantalla SVGA de 800×600 (súper VGA) es ideal para un monitor a color de 14", mientras que la pantalla EVGA o XVGA de 1024×768 (VGA extendido) es ideal para un monitor de 21" o 25", como los que se utilizan en sistemas CAD. Estas resoluciones parecen ser sólo otro conjunto de números, pero hay que tener en cuenta que un receptor de televisión doméstico común tiene una resolución aproximada de 400×300 . La pantalla de alta resolución disponible en los sistemas computacionales es mucho más clara que la de una televisión doméstica. Una resolución de 1024×768 se aproxima a la que tienen las películas de 35 mm. La única desventaja de la pantalla de vídeo en una computadora es el número de colores que se muestran a la vez, pero a medida que pase el tiempo, con seguridad esto mejorará. Los colores adicionales permiten que la imagen aparezca en forma más real, debido a los sutiles matices que se requieren para una imagen verdadera de alta calidad.

Si un sistema de pantalla opera con un tiempo vertical de 60 Hz y un tiempo horizontal de 15,600 Hz, el número de líneas que se generan es de $15,600/60$, o 260 líneas. El número de líneas que se pueden utilizar en este sistema es un valor aproximado a 240, en donde 20 se pierden durante el retorno vertical. Queda claro que el número de líneas de exploración puede ajustarse si se modifican las velocidades de exploración vertical y horizontal. La velocidad de exploración vertical debe ser mayor o igual a 50 Hz; de lo contrario se producirá un parpadeo. La velocidad vertical no debe ser mayor de 75 Hz, ya que pueden producirse problemas con la bobina de deflexión vertical. El rayo de electrones en un monitor se posiciona en base a un campo magnético eléctrico generado por bobinas en un yugo que rodea el cuello del tubo de imagen. Como el campo magnético se genera mediante bobinas, la frecuencia de la señal que se aplica a la bobina está limitada.

La velocidad de exploración horizontal también se limita en base al diseño físico de las bobinas en el yugo. Debido a esto, es común que la frecuencia que se aplica a las bobinas horizontales se encuentre dentro de un rango estrecho. Por lo general, este rango es de 30,000 Hz a 37,000 Hz o de 15,000 Hz a 17,000 Hz. Algunos de los monitores más recientes se denominan monitores "multisync" debido a que la bobina de deflexión está encintada para que pueda controlarse con distintas frecuencias de deflexión. Algunas veces las dos bobinas (vertical y horizontal) están encintadas para obtener distintas velocidades de exploración vertical y horizontal.

Las pantallas de alta resolución utilizan exploración entrelazada o no entrelazada. El sistema de exploración entrelazada se utiliza en todos los estándares, excepto el más alto. En el sistema entrelazado, para mostrar la imagen de vídeo se dibuja la mitad de la imagen primero, con todas las líneas de exploración impares, y después se dibuja la otra mitad con las líneas de exploración pares. Es obvio que este sistema es más complejo y sólo es más eficiente debido a que las frecuencias de exploración

se reducen en un 50%. Por ejemplo, un sistema de vídeo que utiliza 60 Hz para la frecuencia de exploración vertical y 15,720 Hz para la frecuencia horizontal genera 262 ($15,720/60$) líneas de vídeo a una velocidad de 60 cuadros completos por segundo. Si se hace una ligera modificación a la frecuencia horizontal para que sea de 15,750 Hz se generan 262.5 ($15,750/60$) líneas, por lo cual se requieren dos barridos para dibujar una imagen completa de 525 líneas de vídeo. Observe cómo sólo un pequeño cambio en la frecuencia horizontal hace que se duplique el número de líneas de trama.

13-6**RESUMEN**

1. La entrada HOLD se utiliza para solicitar una acción de DMA y la salida HLDA indica que se está llevando a cabo una retención. Cuando se coloca un 1 lógico en la entrada HOLD, el microprocesador (1) deja de ejecutar el programa; (2) coloca sus buses de dirección, de datos y de control en su estado de alta impedancia; y (3) indica que se está llevando a cabo una retención al colocar un 1 lógico en la terminal HLDA.
2. Una operación de lectura de DMA transfiere datos desde una posición de memoria hacia un dispositivo de E/S externo. Una operación de escritura de DMA transfiere datos desde un dispositivo de E/S hacia la memoria. También hay una transferencia de memoria a memoria que permite transferir datos entre dos posiciones de memoria mediante el uso de técnicas de DMA.
3. El controlador 8237 de acceso directo a memoria (DMA) es un dispositivo de cuatro canales que puede expandirse para incluir un canal adicional de DMA.
4. La memoria en disco está representada por el almacenamiento en discos flexibles, los cuales son discos microfloppy de $3\frac{1}{2}$ ". Los discos pueden ser dispositivos de almacenamiento de doble lado, doble densidad (DSDD) o de alta densidad (HD). El disco DSDD de $3\frac{1}{2}$ " almacena 720 Kbytes de datos y el disco HD de $3\frac{1}{2}$ " almacena 1.44 Mbytes de datos.
5. Los datos de la memoria en disco flexible se almacenan mediante el uso de la grabación NRZ (sin retorno a cero). Este método satura el disco con una polaridad de energía magnética para un 1 lógico y la polaridad opuesta para un 0 lógico. En cualquier caso, el campo magnético nunca regresa a 0. Esta técnica elimina la necesidad de una cabeza de borrado separada.
6. Los datos se graban en los discos mediante el uso de los esquemas codificación de modulación de frecuencia modificada (MFM) o longitud de recorrido limitada (RLL). El esquema MFM graba un pulso de datos para un 1 lógico, sin datos ni reloj para el primer 0 lógico de una cadena de ceros, y un pulso de reloj para el segundo y los subsiguientes 0 en una cadena de ceros. El esquema RLL codifica los datos para que pueda empaquetarse un 50% más de información en la misma área del disco. La mayoría de los sistemas de memoria en disco modernos utilizan el esquema de codificación RLL.
7. Los monitores de vídeo pueden ser TTL o analógicos. El monitor TTL utiliza dos niveles discretos de voltaje: 0 V y 5.0 V. El monitor analógico utiliza un número infinito de niveles de voltaje entre 0.0 V y 0.7 V. El monitor analógico puede mostrar un número infinito de niveles de vídeo, mientras que el monitor TTL está limitado a sólo dos niveles de vídeo.
8. El monitor TTL a color muestra 16 colores distintos. Esto se logra mediante tres señales de vídeo (rojo, verde y azul) y una entrada de intensidad. El monitor analógico a color puede mostrar un número infinito de colores a través de sus tres entradas de vídeo. En la práctica, la forma más común del sistema de pantalla analógico a color (VGA) puede mostrar 16 M de colores distintos.
9. Los estándares de vídeo de la actualidad son VGA (640×480), SVGA (800×600) y EVGA o XEVA (1024×768). En los tres casos, la información de vídeo puede contener 16 M de colores.

13-7**PREGUNTAS Y PROBLEMAS**

1. ¿Qué terminales del microprocesador se utilizan para solicitar y aceptar una transferencia por DMA?
2. Explique lo que ocurre cada vez que se coloca un 1 lógico en la terminal de entrada HOLD.

3. Una lectura de DMA transfiere datos desde _____ hacia _____.
4. Una escritura de DMA transfiere datos desde _____ hasta _____.
5. ¿A través de qué señales del bus selecciona el controlador de DMA el dispositivo de E/S que se va a utilizar durante una transferencia por DMA?
6. ¿Cuál es la terminal que utiliza el controlador de DMA para seleccionar el dispositivo de E/S que se va a utilizar durante una transferencia por DMA?
7. ¿Qué es una transferencia por DMA de memoria a memoria?
8. Describa el efecto en el microprocesador y el controlador de DMA cuando las terminales HOLD y HLDA están en el nivel de 1 lógico.
9. Describa el efecto en el microprocesador y el controlador de DMA cuando las terminales HOLD y HLDA están en el nivel de 0 lógico.
10. El controlador de DMA 8237 es un controlador de DMA de _____ canales.
11. Si el controlador de DMA 8237 se decodifica en los puertos de E/S 2000H-200FH, ¿qué puertos se utilizan para programar el canal 1?
12. ¿Cuál registro del controlador de DMA 8237 se programa para inicializar el controlador?
13. ¿Cuántos bytes puede transferir el controlador de DMA 8237?
14. Escriba una secuencia de instrucciones para transferir datos desde las posiciones de memoria 21000H-210FFH hacia las posiciones 20000H-200FFH mediante el uso del canal 2 del controlador de DMA 8237. Debe inicializar el 8237 y utilizar el enclavamiento que se describe en la sección 12-1 para guardar las terminales A₁₉-A₁₆.
15. Escriba una secuencia de instrucciones para transferir datos desde memoria hacia un dispositivo de E/S externo, mediante el uso del canal 3 del 8237. El área de memoria a transferir se encuentra en las posiciones 2000H-20FFFH.
16. ¿Qué es una unidad de almacenamiento portátil USB?
17. El disco de 3½" se conoce como disco _____ floppy.
18. Los datos se graban en anillos concéntricos en la superficie del disco, que se conocen como _____.
19. Una pista se divide en secciones de datos, que se conocen como _____.
20. En un disco de doble lado, las pistas superior e inferior se conocen en conjunto como _____.
21. ¿Por qué se utiliza el método de grabación NRZ en un sistema de memoria en disco?
22. Dibuje el diagrama de sincronización que se genera para escribir los datos 1001010000 mediante el uso de la codificación MFM.
23. Dibuje el diagrama de sincronización que se genera para escribir los datos 1001010000 mediante el uso de la codificación RLL.
24. ¿Qué es una cabeza voladora?
25. ¿Por qué deben estacionarse las cabezas en un disco duro?
26. ¿Cuál es la diferencia entre un mecanismo de posicionamiento de cabeza mediante bobina y un mecanismo de posicionamiento de cabeza mediante un motor de velocidad gradual?
27. ¿Qué es un WORM?
28. ¿Qué es un CD-ROM?
29. ¿Cuántos datos pueden almacenarse en un DVD común?
30. ¿Cuál es la diferencia entre un monitor TTL y un monitor analógico?
31. ¿Cuáles son los tres colores primarios de la luz?
32. ¿Cuáles son los tres colores secundarios de la luz?
33. ¿Qué es un píxel?
34. Una pantalla de vídeo con una resolución de 800 × 600 contiene _____ líneas, en donde cada línea se divide en _____ píxeles.
35. Explique cómo un monitor RGB TTL puede mostrar 16 colores distintos.
36. ¿Qué es un conector DVI?
37. Explique cómo un monitor RGB analógico puede mostrar un número infinito de colores.
38. Si un sistema de vídeo RGB analógico utiliza DACs de 8 bits, puede generar _____ colores distintos.
39. Si un sistema de vídeo utiliza una frecuencia vertical de 60 Hz y una frecuencia horizontal de 32,400 Hz, ¿cuántas líneas de trama se generan?

CAPÍTULO 14

El coprocesador aritmético, las tecnologías MMX y SIMD

INTRODUCCIÓN

La familia Intel de coprocesadores aritméticos está compuesta por los coprocesadores 8087, 80287, 80387SX, 80387DX y 80487SX, que se utiliza con el microprocesador 80486SX. Los microprocesadores del 80486DX al Pentium 4 contienen coprocesadores aritméticos integrados. Hay que tomar en cuenta que algunos de los microprocesadores 80486 clonados (de IBM y Cyrix) no contenían coprocesadores aritméticos. Los conjuntos de instrucciones y la programación para todos los dispositivos son casi idénticos; la principal diferencia es que cada coprocesador fue diseñado para funcionar con un microprocesador Intel distinto. En este capítulo hablaremos con detalle sobre la familia completa de coprocesadores aritméticos. Como el coprocesador es una parte de los microprocesadores del 80486DX al Pentium 4, y éstos son muy comunes, muchos programas requieren, o cuando menos se benefician, de un coprocesador.

La familia de coprocesadores 80X87 multiplica, divide, suma, resta, encuentra la raíz cuadrada y calcula la tangente parcial, el arco tangente parcial y logaritmos. Los tipos de datos son enteros con signos integrados de 16, 32 y 64 bits; datos BCD de 18 dígitos, y números de punto flotante de 32, 64 y 80 bits. Las operaciones que realiza la familia 80X87 se ejecutan, por lo general, muchas veces más rápido que las operaciones equivalentes escritas con los programas más eficientes que utilizan el conjunto de instrucciones normal del microprocesador. Con el coprocesador Pentium mejorado, las operaciones se ejecutan aproximadamente cinco veces más rápido que mediante el microprocesador 80486 con una frecuencia de reloj similar. El Pentium a menudo puede ejecutar una instrucción de coprocesador y dos instrucciones de números enteros de manera simultánea. Los coprocesadores Pentium Pro a Pentium 4 son similares en rendimiento al coprocesador Pentium, sólo que traen agregadas dos instrucciones: FCMOV y FCOMI.

Las extensiones multimedia (MMX) para los microprocesadores del Pentium al Pentium 4 son instrucciones que comparten el conjunto de registros del coprocesador aritmético. La extensión MMX es un procesador interno especial diseñado para ejecutar instrucciones de números enteros a alta velocidad para dispositivos multimedia externos. Por tal razón, en este capítulo analizaremos el conjunto de instrucciones MMX y sus especificaciones. Las extensiones SIMD (una sola instrucción, varios datos), que se conocen como SSE (extensiones SIMD de flujo), son similares a las instrucciones MMX, sólo que funcionan con números de punto flotante en vez de enteros, y no utilizan el espacio de registros del coprocesador, como las instrucciones MMX.

OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

1. Realizar conversiones entre datos decimales y enteros con signo, BCD, y datos de punto flotante con el coprocesador aritmético y las tecnologías MMX y SSE.

2. Explicar la operación del coprocesador aritmético 80X87 y las unidades MMX y SSE.
3. Explicar la operación y los modos de direccionamiento de cada coprocesador aritmético, así como de cada instrucción MMX y SSE.
4. Desarrollar programas que resuelvan problemas aritméticos complejos mediante el uso del coprocesador aritmético y las instrucciones MMX y SSE.

14-1**FORMATOS DE LOS DATOS PARA EL COPROCESADOR ARITMÉTICO**

En esta sección del libro presentaremos los tipos de datos que se utilizan en todos los miembros de la familia de coprocesadores aritméticos. (En la tabla 14-1 verá un listado de todos los microprocesadores Intel y sus coprocesadores complementarios.) Estos tipos de datos son: enteros con signo, BCD y punto flotante. Cada uno tiene un uso específico en un sistema, en tanto que muchos sistemas requieren los tres tipos de datos. Tenga en cuenta que la programación en lenguaje ensamblador con el coprocesador se limita a menudo a modificar la codificación generada por un lenguaje de alto nivel tal como C/C++. Para realizar dicha modificación, se requiere el conjunto de instrucciones y ciertos conceptos básicos de programación, los cuales veremos en este capítulo.

Enteros con signo

Los enteros con signo que se utilizan con el coprocesador son los mismos que describimos en el capítulo 1. Cuando lo hacen con el coprocesador aritmético, los enteros con signo son de 16 (palabra), 32 (entero de doble palabra) o 64 bits (entero de palabra cuádruple). El entero largo es nuevo para el coprocesador, por lo que no se describe en el capítulo 1, pero los principios son los mismos. La conversión entre el formato decimal y el entero con signo se maneja de la misma forma exacta que la de los enteros con signo que vimos en el capítulo 1. Como recordará, los números positivos se almacenan en formato verdadero con un bit de signo más a la izquierda de 0, mientras que los números negativos se almacenan en formato de complemento a dos, con un bit de signo más a la izquierda de 1.

Los enteros tipo palabra varían en valor desde -32768 a +32767, en tanto que el rango de los enteros tipo doble palabra es de $\pm 2 \times 10^9$ y el de los enteros tipo palabra cuádruple es de $\pm 9 \times 10^{18}$. Los datos tipo entero se encuentran en ciertas aplicaciones que utilizan el coprocesador aritmético. En la figura 14-1 verá estas tres formas de datos tipo entero con signo.

Los datos se almacenan en memoria mediante el uso de las mismas directivas de ensamblador que describimos y utilizamos en capítulos anteriores. La directiva DW define palabras; DD, enteros tipo doble palabra, y DQ, enteros tipo palabra cuádruple. El ejemplo 14-1 muestra cómo definir varios tamaños de enteros con signo para que los utilicen el ensamblador y el coprocesador aritmético.

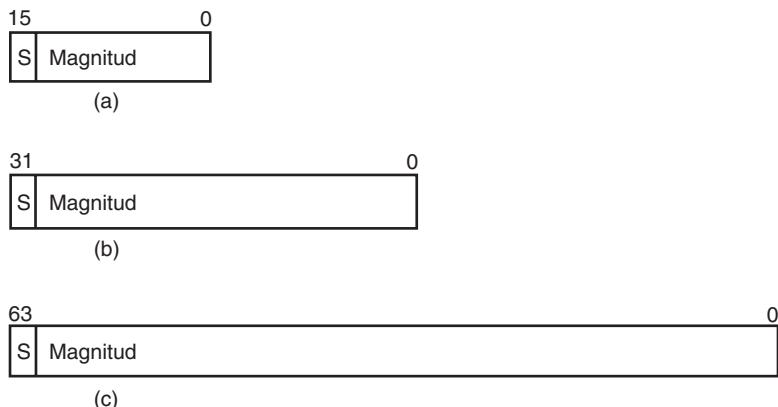
EJEMPLO 14-1

0000 0002	DATOS1	DW	2	;entero de 16 bits
0002 FFDE	DATOS2	DW	-34	;entero de 16 bits
0004 000004D2	DATOS3	DD	1234	;entero de 32 bits
0008 FFFFFFF9C	DATOS4	DD	-100	;entero de 32 bits
000C 000000000005BA0	DATOS5	DQ	23456	;entero de 64 bits
0014 FFFFFFFFFFFFFF86	DATOS6	DQ	-122	;entero de 64 bits

TABLA 14-1
Compatibilidad entre el microprocesador y el coprocesador.

	<i>Microprocesador</i>	<i>Coprocesador</i>
8086/8088		8087
80186/80188		80187
80286		80287
80386		80387
80486SX		80487SX
80486DX-Pentium 4		Integrado en el microprocesador

FIGURA 14-1 Formatos de enteros para la familia 80X87 de coprocesadores aritméticos: *a*) palabra, *b*) corto y *c*) largo.



Nota: S = bit de signo

Decimal codificado binario (BCD)

El formato decimal codificado binario (BCD) requiere 80 bits de memoria. Cada número se almacena como un entero empaquetado de 18 dígitos en nueve bytes de memoria con dos dígitos por byte. El décimo byte contiene sólo un bit de signo para el número BCD con signo de 18 dígitos. La figura 14-2 muestra el formato del número BCD que se utiliza con el coprocesador aritmético. Observe que tanto los números positivos como los negativos se almacenan en formato real y nunca en formato de complemento a diez. La directiva DT almacena datos BCD en la memoria, como se muestra en el ejemplo 14-2. Tal formato se utiliza pocas veces, ya que es único para el coprocesador Intel.

EJEMPLO 14-2

```
0000 000000000000000000200  DATOS1  DT      200      ;define 10 bytes
000A 800000000000000000000010  DATOS2  DT      -10      ;define 10 bytes
0014 000000000000000010020  DATOS3  DT     10020    ;define 10 bytes
```

Punto flotante

Los números de punto flotante se conocen como *números reales* porque almacenan enteros con signo, fracciones y números mixtos. Un número de punto flotante consta de tres partes: un bit de signo, un exponente desviado y una mantisa. Los números de punto flotante se escriben en notación binaria científica. La familia Intel de coprocesadores aritméticos soporta tres tipos de números de punto flotante: individual (32 bits), doble (64 bits) y temporal (80 bits). En la figura 14-3 verá las tres formas del número de punto flotante. Tome en cuenta que el formato individual también se conoce como número de precisión simple y el formato doble se conoce como número de doble precisión. Algunas veces al formato temporal de 80 bits se le llama número de precisión extendida. Los números de punto flotante y las operaciones que realiza el coprocesador aritmético se conforman con el estándar IEEE-754, adoptado por todos los productores importantes de software para computadoras personales. Microsoft, uno de ellos, en 1995 dejó de dar soporte el formato de punto flotante de Microsoft y también el estándar ANSI de punto flotante, que es popular en algunos sistemas de computadoras centrales (mainframe).

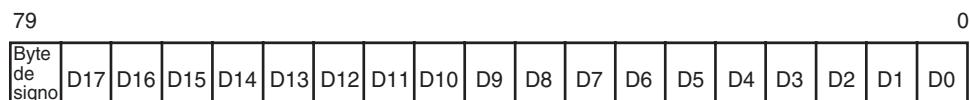
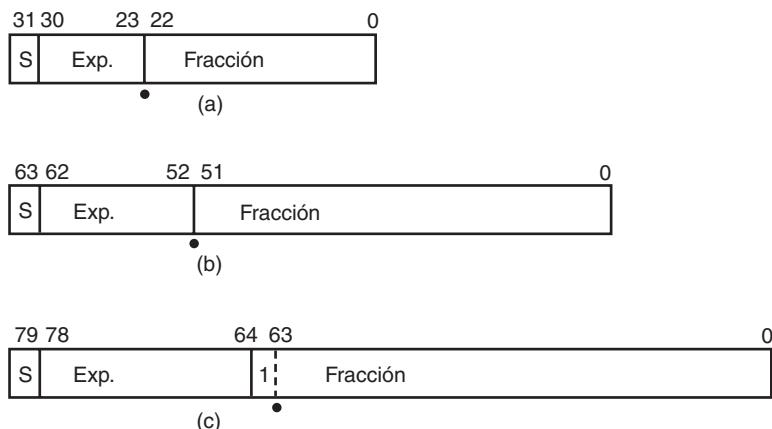


FIGURA 14-2 Formato de datos BCD para la familia 80X87 de coprocesadores aritméticos.

FIGURA 14-3 El formato de punto flotante (real) para la familia 80X87 de coprocesadores aritméticos. a) Corto (precisión simple), con una desviación de 7FH; b) largo (doble precisión), con una desviación de 3FFH, y c) temporal (precisión extendida), con una desviación de 3FFFH.



Nota: S = bit de signo y Exp. = exponente

Conversión a formato de punto flotante. El proceso de conversión de formato decimal a formato de punto flotante es una tarea simple, la cual se realiza mediante los siguientes pasos:

1. El número decimal se convierte en binario.
2. El número binario se normaliza.
3. El exponente desviado se calcula.
4. El número en el formato de punto flotante se almacena.

En el ejemplo 14-3 se muestran estos cuatro pasos para el número decimal 100.25_{10} . Aquí, el número decimal se convierte en un número de punto flotante de precisión simple (32 bits).

EJEMPLO 14-3

Paso	Resultado
1	$100.25 \Rightarrow 1100100.01$
2	$1100100.01 \Rightarrow 1.10010001 \times 2^6$
3	$110 + 0111111111 \Rightarrow 10000101$
4	Signo $\Rightarrow 0$ Exponente $\Rightarrow 10000101$ Mantisa $\Rightarrow 10010001000000000000000000$

En el paso 3 del ejemplo 14-3, el exponente desviado es el exponente 2^6 o 110, más una desviación de 011111 (7FH) o 10000101 (85H). Todos los números de precisión simple utilizan una desviación de 3FFFH.

En el paso 4 del ejemplo 14-3, la información de los pasos anteriores se combina para formar el número de punto flotante. El bit de más a la izquierda es el bit de signo del número. En este caso es un 0, debido a que el número es $+100.25_{10}$. El exponente desviado sigue después del bit de signo. La mantisa es un número de 23 bits con un bit uno implícito. Observe que la mantisa de un número 1.XXXX es la porción XXXX. El 1. es un **bit uno implícito**, que sólo se almacena en el formato de precisión temporal extendida del número de punto flotante como un bit uno explícito.

En unos cuantos números se aplican ciertas reglas especiales. Por ejemplo, el número 0 se almacena como sólo ceros, excepto el bit de signo, que puede ser un 1 lógico para representar un cero negativo. El infinito positivo y negativo se almacenan como 1s lógicos en el exponente con una mantisa de puros ceros y el bit de signo que representa positivo o negativo. Un NAN (no es un número) es un

resultado inválido de punto flotante que tiene sólo unos en el exponente con una mantisa que no consta sólo de ceros.

Conversión desde el formato de punto flotante. La conversión de un número de punto flotante en un número decimal se resume en los siguientes pasos:

1. Se separan el bit de signo, el exponente desviado y la mantisa.
2. Se convierte el exponente desviado en un exponente real restándole la desviación.
3. El número se escribe como número binario normalizado.
4. Se convierte en un número binario no normalizado.
5. El número binario no normalizado se convierte en decimal.

Estos cinco pasos convierten un número de punto flotante de precisión simple en decimal, como se muestra en el ejemplo 14-4. Observe cómo el bit de signo de 1 convierte el resultado decimal en negativo; también, que el bit uno implícito se agrega al resultado binario normalizado en el paso 3.

EJEMPLO 14-4

Paso	Resultado
1	Signo => 1 Exponente => 10000011 Mantisa => 100100100000000000000000
2	100 = 10000011 - 01111111
3	1.1001001 x 2 ⁴
4	11001.001
5	-25.125

Almacenamiento de datos tipo punto flotante en la memoria. Los números de punto flotante se almacenan con el ensamblador mediante la directiva DD para precisión simple, DQ para doble precisión y DT para precisión temporal extendida. En el ejemplo 14-5 se muestran algunos ejemplos de almacenamiento de datos tipo punto flotante. El autor descubrió que el macroensamblador de Microsoft contiene un error que no permite utilizar un signo positivo con los números de punto flotante positivos. Un +92.45 debe definirse como 92.45, para que el ensamblador funcione correctamente. Microsoft ha asegurado al autor que este error se corrige en la versión 6.11 de MASM si se utilizan las directivas REAL4, REAL8 o REAL10, en lugar de DD, DQ y DT, para especificar datos tipo punto flotante. El ensamblador proporciona el acceso al emulador del 8087 si su sistema no contiene un microprocesador con coprocesador. El emulador se incluye en todos los lenguajes de alto nivel de Microsoft o como programas de demostración (shareware), tal como el EM87. Para acceder al emulador debe incluir la instrucción OPTION EMULATOR inmediatamente después de la instrucción .MODEL en un programa. Tenga en cuenta que el emulador no emula algunas de las instrucciones del coprocesador. No utilice esta opción si su sistema contiene un coprocesador. En cualquier caso, incluya los modificadores .8087, .80187, .80287, .80387, .80487, .80587 o .80687 para habilitar la generación de instrucciones del coprocesador.

EJEMPLO 14-5

0000 C377999A	DATOS7	DD	-247.6	;precisión simple
0004 40000000	DATOS8	DD	2.0	;precisión simple
0008 486F4200	DATOS9	REAL4	2,45E+5	;precisión simple
000C 4059100000000000	DATOSA	DQ	100.25	;doble precisión
0014 3F543BF727136A40	DATOSB	REAL8	0.001235	;doble precisión
001C 400487F34D6A161E4F76	DATOSC	REAL10	33.9876	;precisión temporal

14-2

LA ARQUITECTURA DEL 80X87

El 80X87 se diseñó para operar en forma concurrente con el microprocesador. Observe que los microprocesadores del 80486DX al Pentium 4 contienen sus versiones internas y cien por ciento compatibles con el 80387. En otros miembros de la familia, el coprocesador es un circuito integrado externo que pone en paralelo a la mayoría de las conexiones en el microprocesador. El 80X87 ejecuta 68 instrucciones distintas. El microprocesador ejecuta todas las instrucciones normales y el 80X87 ejecuta instrucciones del coprocesador aritmético. Tanto el microprocesador como el coprocesador ejecutan sus respectivas instrucciones en forma simultánea o concurrente. El coprocesador numérico o aritmético es un microprocesador de propósito especial, diseñado para ejecutar con eficiencia operaciones aritméticas y trascendentales.

El microprocesador intercepta y ejecuta el conjunto normal de instrucciones, mientras el coprocesador intercepta y ejecuta sólo las instrucciones del coprocesador. Recuerde que las instrucciones del coprocesador son instrucciones de salida (ESC). El microprocesador utiliza tales instrucciones para generar una dirección de memoria para el coprocesador, de manera que éste ejecute una instrucción del coprocesador.

Estructura interna del 80X87

La figura 14-4 muestra la estructura interna del coprocesador aritmético. Observe que este dispositivo se divide en dos secciones principales: la unidad de control y la unidad de ejecución numérica.

La **unidad de control** sirve como interfaz entre el coprocesador y el bus de datos del sistema del microprocesador. Ambos dispositivos monitorean el flujo de instrucciones. Si la instrucción es un ESCape (coprocesador), el coprocesador la ejecuta; sino, lo hace el microprocesador.

La **unidad de ejecución numérica** (NEU, sus siglas en inglés) es responsable de la ejecución de todas las instrucciones del coprocesador. La NEU tiene una pila de ocho registros que almacena los operandos para las instrucciones aritméticas y los resultados de las mismas. Las instrucciones pueden direccionar los datos en registros de datos específicos de la pila o utilizar un mecanismo de “meter

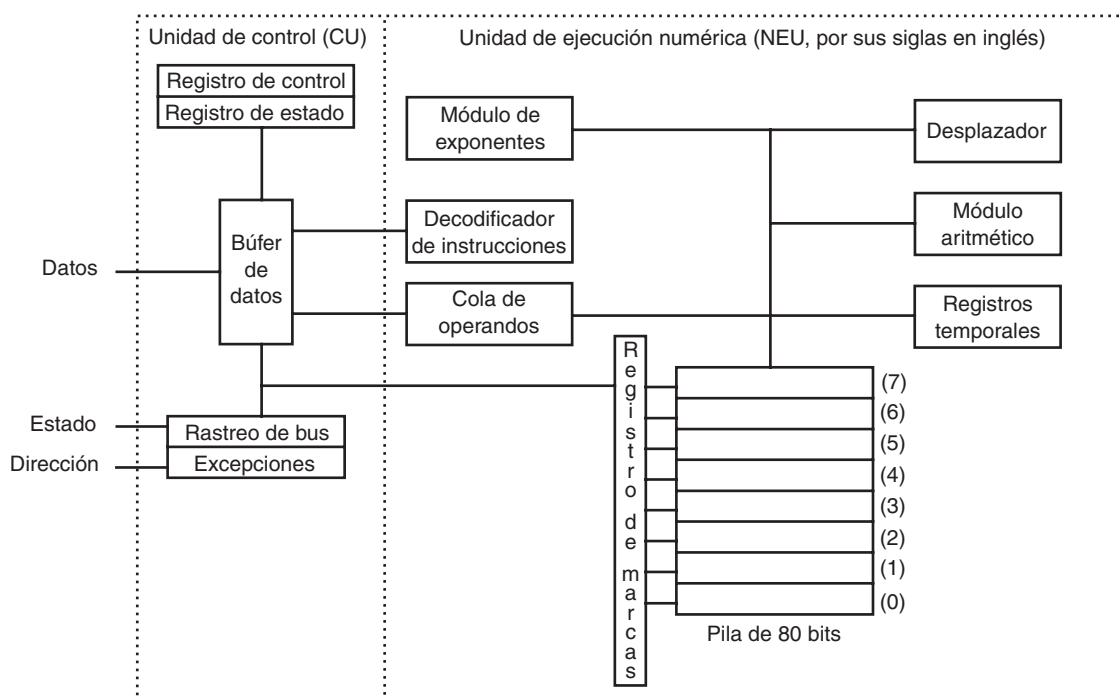


FIGURA 14-4 La estructura interna del coprocesador aritmético 80X87.

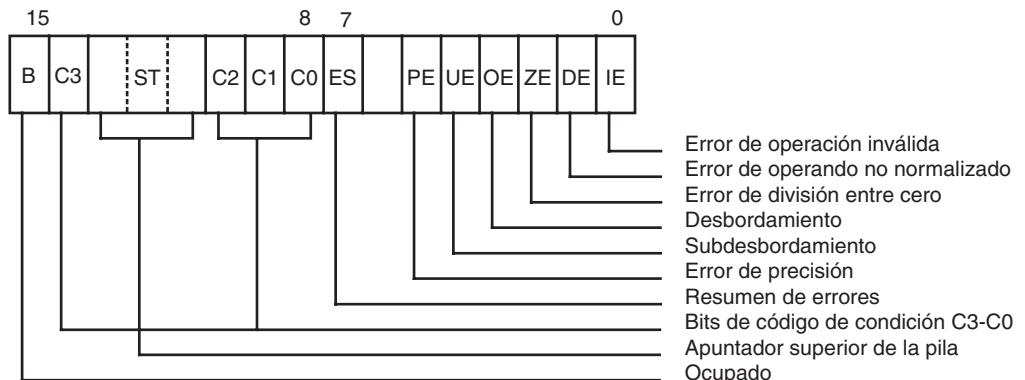


FIGURA 14-5 El registro de estado del coprocesador aritmético 80X87.

y sacar” para almacenar y extraer datos de la parte superior de la pila. Los otros registros en la NEU son: de estado, de control, de marca y apuntadores de excepción. Hay unas cuantas instrucciones que transfieren datos del coprocesador al registro AX en el microprocesador. La instrucción FSTSW AX es la instrucción disponible para el coprocesador que permite la comunicación directa con el microprocesador a través del registro AX. Observe que el 8087 no contiene la instrucción FSTSW AX, aunque los coprocesadores más recientes sí la traen.

La pila dentro del coprocesador contiene ocho registros de 80 bits cada uno. Estos registros de la pila siempre traen un número de punto flotante de precisión extendida de 80 bits. El único momento en que los datos aparecen en cualquier otro formato es cuando residen en el sistema de memoria. El coprocesador convierte los formatos entero con signo BCD, precisión simple o doble precisión, a medida que los datos se desplazan entre la memoria y la pila de registros del coprocesador.

Registro de estado. El registro de estado (vea la figura 14-5) refleja la operación del coprocesador en general. Para acceder a este registro se ejecuta la instrucción (FSTSW), la cual almacena el contenido del registro de estado en una palabra de memoria. La instrucción FSTSW AX copia el registro de estado en forma directa al registro AX del microprocesador en el coprocesador 80187 o superior. Una vez que se almacena el estado en la memoria o el registro AX, las posiciones de los bits del registro de estado logran examinarse mediante el software común. Las comunicaciones entre el coprocesador y el microprocesador se llevan a cabo a través de los puertos de E/S 00FAH-00FFH en el 801876 y en el 80287, y a través de los puertos de E/S 800000FAH-800000FFH en los microprocesadores del 80386 al Pentium 4. Nunca utilice estos puertos de E/S para conectar dispositivos de E/S al microprocesador.

Los coprocesadores más recientes (80187 y superiores) utilizan la posición del bit de estado 6 (SF) para indicar un desbordamiento de la pila o un error de subdesbordamiento. A continuación se muestra una lista de los bits de estado (excepto SF) y sus aplicaciones:

- | | |
|------------------------------------|--|
| B | El bit de ocupado indica que el coprocesador está ocupado ejecutando una tarea. Este bit se evalúa mediante el análisis del registro de estado o el uso de la instrucción FWAIT. Los coprocesadores más recientes se sincronizan con el microprocesador de manera automática, por lo que no surge la necesidad de evaluar la bandera de ocupado antes de realizar tareas adicionales con el coprocesador. |
| C₀-C₃ | Los bits de código de condición indican condiciones relacionadas con el coprocesador (en la tabla 14-2 verá un listado completo de cada una de las combinaciones de estos bits y sus funciones). Observe que tales bits tienen distintos significados para distintas instrucciones, como se indica en la tabla. La parte superior de la pila se denota como ST en la misma tabla. |
| TOP | El bit parte superior de la pila (ST) indica el registro direccionado como parte superior de la pila (ST). Por lo general, siempre es el registro ST(0). |
| ES | El bit resumen de errores está activo si se activa cualquier bit de error sin máscara (PE, UE, OE, ZE, DE o IE). En el coprocesador 8087, el resumen de errores también |

TABLA 14-2 Los bits del código de condición del registro de estado del coprocesador.

<i>Instrucción</i>	<i>C</i> ₃	<i>C</i> ₂	<i>C</i> ₁	<i>C</i> ₀	<i>Indicación</i>
FTST, FCOM	0	0	X	0	ST > Operando
	0	0	X	1	ST < Operando
	1	0	X	1	ST = Operando
	1	1	X	1	ST no puede compararse
FPREM	Q1	0	Q0	Q2	Los 3 bits de más a la derecha del cociente
	?	1	?	?	Incompleto
FXAM	0	0	0	0	+ anormal
	0	0	0	1	+ NAN
	0	0	1	0	- anormal
	0	0	1	1	- NAN
	0	1	0	0	+ normal
	0	1	0	1	+ ∞
	0	1	1	0	- normal
	0	1	1	1	- ∞
	1	0	0	0	+ 0
	1	0	0	1	Vacío
	1	0	1	0	- 0
	1	0	1	1	Vacío
	1	1	0	0	+ no normal
	1	1	0	1	Vacío
	1	1	1	0	- no normal
	1	1	1	1	Vacío

Observaciones: anormal = los bits a la izquierda de la mantisa son cero; no normal = el exponente está en su valor más negativo; normal = formato de punto flotante estándar; NAN (no es un número) = un exponente de sólo unos y una mantisa distinta de cero; el operando para TST es cero.

producía una interrupción del coprocesador. Desde el 80187, la interrupción del coprocesador ha estado ausente en la familia.

- PE** El **error de precisión** indica que el resultado o los operandos se excedieron de la precisión seleccionada.
- UE** Un **error de subdesbordamiento** indica un resultado distinto de cero que es demasiado pequeño como para representarlo con la precisión actual seleccionada por la palabra de control.
- OE** Un **error de desbordamiento** indica un resultado demasiado grande como para poder representarlo. Si el error está enmascarado, el coprocesador genera infinito para un error de desbordamiento.
- ZE** Un **error de cero** indica que el divisor es cero, mientras que el dividendo es un número distinto de infinito o de cero.
- DE** Un **error no normalizado** indica que cuando menos uno de los operandos no está normalizado.
- IE** Un **error inválido** indica un desbordamiento o subdesbordamiento de la pila, un formato indeterminado ($0 \div 0, +\infty, -\infty$, etc.) o el uso de un NAN como operando. Esta bandera indica errores tales como los que se producen al sacar la raíz cuadrada de un número negativo, y así sucesivamente.

Existen dos formas de evaluar los bits del registro de estado, una vez que se mueven hacia el registro AX con la instrucción FSTSW AX. Uno de los métodos utiliza la instrucción TEST para evaluar los bits individuales del registro de estado. El otro, la instrucción SAHF para transferir los ocho bits de más a la izquierda del registro de estado hacia el registro de banderas del microprocesador. En el ejemplo 14-6 se ilustran ambos métodos. Este ejemplo utiliza la instrucción DIV para dividir la parte superior de la pila entre el contenido de DATOS1 y la instrucción FSQRT para encontrar la raíz cuadrada de la parte

TABLA 14-3

Las condiciones del coprocesador se evalúan con saltos condicionales, como se muestra en el ejemplo 14-6.

C_3	C_2	C_0	Condición	Instrucción de salto
0	0	0	ST > Operando	JA (salta si ST es mayor)
0	0	1	ST < Operando	JB (salta si ST es menor)
1	0	0	ST = Operando	JE (salta si ST es igual)

superior de la pila. En el ejemplo también se utiliza la instrucción FCOM para comparar el contenido de la parte superior de la pila con DATOS1. Observe que las instrucciones de salto condicional se utilizan con la instrucción SAHF para evaluar la condición que se lista en la tabla 14-3. Aunque la instrucción SAHF y los saltos condicionales no pueden evaluar todas las posibles condiciones de operación del coprocesador, ayudan a reducir la complejidad de ciertas condiciones evaluadas. Observe que SAHF coloca a C_0 en la bandera de acarreo, a C_2 en la bandera de paridad y a C_3 en la bandera de cero.

EJEMPLO 14-6

```
; prueba para un error de división entre cero
FDIV    DATOS1
FSTSW   AX          ; copia el registro de estado en AX
TEST    AX, 4        ; evalúa el bit ZE
JNZ     ERROR_DIVISION

; prueba para una operación inválida después de una instrucción FSQRT
FSQRT
FSTSW   AX
TEST    AX, 1        ; evalúa IE
JNZ     ERROR_FSQRT

; prueba con SAHF para usar saltos condicionales
FCOM    DATOS1
FSTSW   AX
SAHF
JE     ST_IGUAL
JB     ST_MENOR
JA     ST_MAYOR      ; copia las banderas del coprocesador a las banderas
```

Cuando se ejecutan las instrucciones FXAM y FSTSW AX, seguidas de la instrucción SAHF, la bandera de cero contendrá a C_3 . La instrucción FXAM podría utilizarse para evaluar un divisor antes de una división para ver si es cero mediante la instrucción JZ, después de FXAM, FSTSW AX y SAHF.

Registro de control. En la figura 14-6 se muestra el registro de control. Dicho registro selecciona los controles de precisión, redondeo e infinito. También enmascara y desenmascara los bits de excepción que corresponden a los seis bits de más a la derecha del registro de estado. La instrucción FLDCW se utiliza para cargar un valor en el registro de control.

A continuación se muestra una descripción de cada bit o agrupamiento de bits que se encuentran en el registro de control:

- IC** **Control de infinito** selecciona entre infinito afín o proyectivo. La opción afín permite un infinito positivo y negativo; la opción proyectivo supone que el infinito no tiene signo.
- RC** **Control de redondeo** determina el tipo de redondeo, como se define en la figura 14-6.
- PC** El **control de precisión** establece la precisión del resultado, como se define en la figura 14-6.
- Máscaras de excepción** Determina si el error indicado por la excepción afecta al bit de error en el registro de estado. Si se coloca un 1 lógico en uno de los bits de control de excepción, se desenmascara el bit del registro de estado correspondiente.

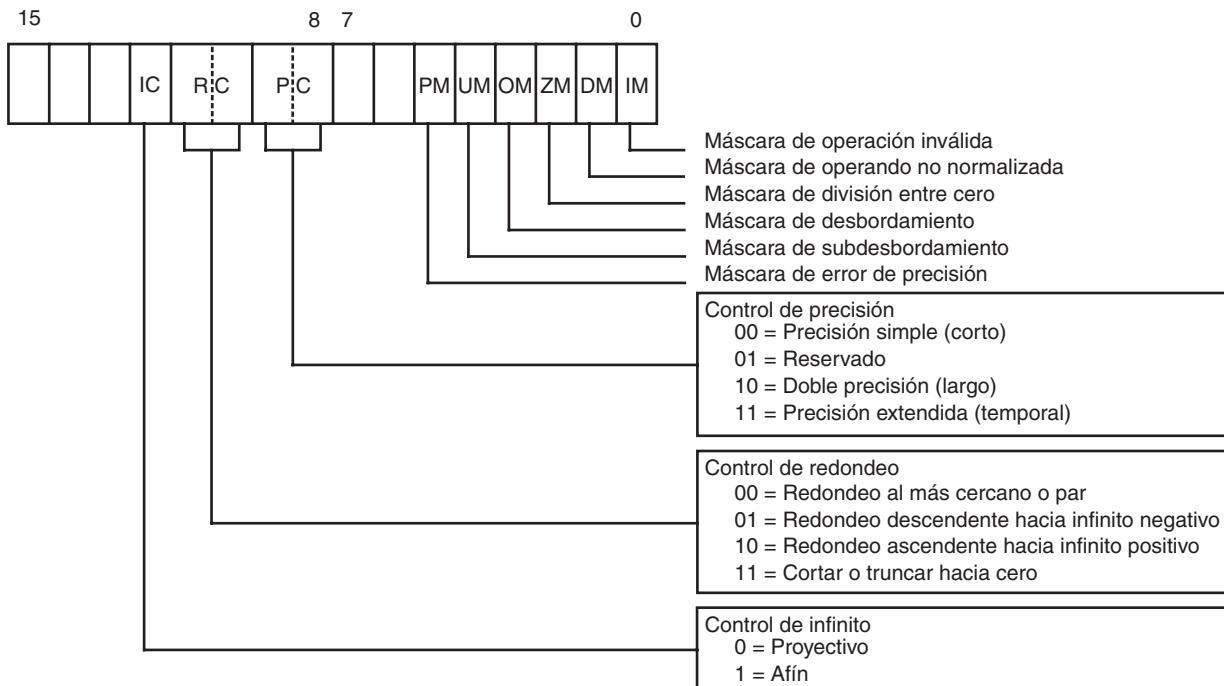
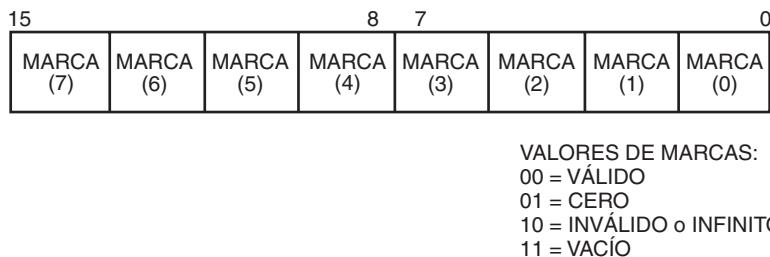


FIGURA 14-6 El registro de control del coprocesador aritmético 80X87.

FIGURA 14-7 El registro de marca del coprocesador aritmético 80X87.



Registro de marca. El **registro de marca** indica el contenido de cada posición en la pila del coprocesador. La figura 14-7 muestra el registro de marca y el estado indicado por cada marca. La marca indica si un registro es válido; cero; inválido o infinito; o si está vacío. La única forma en que un programa logra ver el registro de marca es mediante el almacenamiento del entorno del coprocesador con las instrucciones FSTENV, FSAVE o FRSTOR. Cada una de estas instrucciones almacena el registro de marca junto con otros datos del coprocesador.

14-3

CONJUNTO DE INSTRUCCIONES

El coprocesador aritmético ejecuta más de 68 instrucciones diferentes. Cada vez que una instrucción del coprocesador hace referencia a la memoria, el microprocesador genera de manera automática la dirección de memoria para la instrucción. El coprocesador utiliza el bus de datos para las transferencias de datos durante las instrucciones del coprocesador, en tanto que el microprocesador lo utiliza durante las instrucciones normales. Además, observe que el 80287 utiliza los puertos de E/S 00F8H-00FFH reservados por Intel para las comunicaciones entre el coprocesador y el microprocesador (aún cuando el coprocesador sólo utiliza puertos 00FCH-00FFH). Estos puertos se utilizan principalmente

para la instrucción FSTSW AX. Los coprocesadores del 80387 al Pentium 4 utilizan los puertos de E/S 800000F8H-800000FFH para tales comunicaciones.

En esta sección del libro describiremos la función de cada instrucción y listaremos su formato en lenguaje ensamblador. Como el coprocesador utiliza los modos de direccionamiento de memoria del microprocesador, no se muestran todas las formas de cada instrucción. Cada vez que el ensamblador encuentra un código de operación mnemónico del coprocesador, lo convierte en una instrucción ESC en lenguaje máquina. La instrucción ESC representa un código de operación para el coprocesador.

Instrucciones de transferencia de datos

Hay tres transferencias de datos básicas: de punto flotante, de entero con signo y BCD. La única parte en que los datos aparecen en formato de entero con signo o BCD es en la memoria. Dentro del coprocesador, los datos siempre se almacenan como números de punto flotante con precisión extendida de 80 bits.

Transferencias de datos de punto flotante. Existen cuatro instrucciones tradicionales de transferencia de datos tipo punto flotante en el conjunto de instrucciones del coprocesador: FLD (carga real), FST (almacena real), FSTP (almacena real y saca) y FXCH (intercambia). La nueva instrucción que se agregó a los microprocesadores del Pentium Pro al Pentium 4 se conoce como instrucción de movimiento condicional de punto flotante (la cual describiremos a continuación). Esta instrucción utiliza el código de operación FCMOV con una condición de punto flotante.

La instrucción FLD carga datos de punto flotante de la memoria hacia la parte interna de la pila interna, conocida como ST (parte superior de la pila). La instrucción almacena los datos en la parte superior de la pila y después decremente el apuntador de la pila en 1. Los datos que se cargan en la parte superior de la pila provienen de cualquier posición de memoria, o de otro registro del coprocesador. Por ejemplo, una instrucción FLD ST(2) copia el contenido del registro 2 en la parte superior de la pila, la cual es ST. La parte superior de la pila es el registro 0, cuando el coprocesador se reinicia o se inicializa. Otro ejemplo es la instrucción FLD DATOS7, la cual copia el contenido de la posición de memoria DATOS7 hacia la parte superior de la pila. El ensamblador determina el tamaño de la transferencia en forma automática a través de las directivas DD o REAL4 para precisión simple, DQ o REAL8 para doble precisión y DT o REAL10 para precisión temporal extendida.

La instrucción FST almacena una copia de la parte superior de la pila en la posición de memoria o en el registro del coprocesador que indique el operando. Al momento del almacenamiento, el número de punto flotante interno, de precisión temporal extendida, se redondea al tamaño del número de punto flotante que indique el registro de control.

La instrucción FSTP (almacena y saca punto flotante) almacena una copia de la parte superior de la pila en la memoria o en cualquier registro del coprocesador, y después saca los datos de la parte superior de la pila. Es posible considerar a FST como una instrucción para copiar y a FSTP como una instrucción de extracción.

La instrucción FXCH intercambia el registro indicado por el operando con la parte superior de la pila. Por ejemplo, la instrucción FXCH ST(2) intercambia la parte superior de la pila con el registro 2.

Instrucciones de transferencia de datos enteros. El coprocesador soporta tres instrucciones de transferencia de datos tipo entero: FIELD (carga entero), FIST (almacena entero) y FISTP (almacena entero y saca). Estas tres instrucciones funcionan igual que FLD, FST y FSTP, sólo que los datos que se transfieren son de tipo entero. El coprocesador convierte de manera automática los datos de punto flotante internos con precisión temporal extendida en datos enteros. El tamaño de los datos se determina con base en la manera en la que se define la etiqueta con DW, DD o DQ, en el programa en lenguaje ensamblador.

Instrucciones de transferencia de datos BCD. Hay dos instrucciones para cargar o almacenar datos enteros con signo BCD. La instrucción FBLD carga la parte superior de la pila con datos BCD de la memoria, en tanto que la instrucción FBSTP almacena en la parte superior de la pila y efectúa una extracción.

La instrucción FCMOV del Pentium Pro al Pentium 4. Los microprocesadores del Pentium Pro al Pentium 4 contienen una nueva instrucción llamada FCMOV, la cual también trae una condición. Si

TABLA 14-4 Las instrucciones FCMOV y las condiciones que evalúan.

Instrucción	Condición
FCMOVB	Mueve si es menor
FCMOVE	Mueve si es igual
FCMOVBE	Mueve si es menor o igual
FCMOVU	Mueve si está desordenado
FCMOVNB	Mueve si no es menor
FCMOVNE	Mueve si no es igual
FCMOVNBE	Mueve si no es menor o igual
FCMOVNU	Mueve si no está desordenado

ésta es verdadera, la instrucción FCMOV copia el origen en el destino. En la tabla 14-4 aparecen las condiciones que evalúa FCMOV y los códigos de operación utilizados. Observe que tales condiciones comprueban que hay una condición ordenada o desordenada. La instrucción FCMOV no evalúa números NAN o no normalizados.

El ejemplo 14-7 muestra cómo se utiliza la instrucción FCMOV (muestra si es menor) para copiar el contenido de ST(2) a la parte superior de la pila (ST) si el contenido de ST(2) es menor que ST. Observe que debe usarse la instrucción FCOM para realizar la comparación, en tanto que el contenido del registro de estado debe copiarse, de todas formas, a las banderas para que tal instrucción funcione. Con la instrucción FCOMI aparecen más detalles sobre la instrucción FCMOV; la instrucción FCOMI también es nueva para los microprocesadores del Pentium Pro al Pentium 4.

EJEMPLO 14-7

```

FCOM      ST(2)          ;compara ST y ST(2)
FSTSW    AX              ;banderas de punto flotante a AX
SAHF      ST(2)          ;banderas de punto flotante a banderas
FCMOVB   ST(2)          ;copia ST(2) a ST si es menor
O
FCOMI     ST(2)
FCMOVB   ST(2)

```

Instrucciones aritméticas

Las instrucciones aritméticas para el coprocesador son: suma, resta, multiplicación, división y cálculo de raíces cuadradas. Las instrucciones relacionadas con los comandos aritméticos son: escala, redondeo, valor absoluto y cambio de signo.

La tabla 14-5 muestra los modos básicos de direccionamiento permitidos para las operaciones aritméticas. Cada modo de direccionamiento se muestra con un ejemplo que utiliza la instrucción FADD

TABLA 14-5 Modos de direccionamiento del coprocesador.

Modo	Formato	Ejemplo
Pila	ST(1),ST	FADD
Registro	ST,ST(n)	FADD ST,ST(1)
	ST(n),ST	FADD ST(4),ST
Registro con extracción (pop)	ST(n),ST	FADDP ST(3),ST
Memoria	Operando	FADD DATOS3

Observaciones: La dirección de la pila está fija como ST(1),ST e incluye una extracción, por lo que sólo el resultado permanece en la parte superior de la pila y n = registro número 0-7.

(suma real). Todas las operaciones aritméticas son de punto flotante, excepto algunos casos en los que se hace referencia a los datos de la memoria como un operando.

La forma clásica de la pila para direccionar los datos de un operando (direcciónamiento de pila) utiliza la parte superior de la pila como operando de origen y la posición siguiente a la parte superior de la pila como operando de destino. Después de eso, una operación “pop” extrae el dato de origen de la pila, por lo que sólo queda el resultado en el registro de destino en la parte superior de la pila. Para utilizar este modo de direcciónamiento, la instrucción se coloca en el programa sin ningún operando, como FADD o FSUB. La instrucción FADD suma ST a ST(1) y almacena la respuesta en la parte superior de la pila; también extrae los dos datos originales de la pila mediante operaciones “pop”. Observe con cuidado que FSUB resta ST de ST(1) y deja la diferencia en ST. Por lo tanto, una resta inversa (FSUBR) resta ST(1) de ST y deja la diferencia en ST. (Nota: hay un error en la documentación de Intel, incluyendo el libro técnico de Pentium, en el cual se describe la operación de algunas instrucciones inversas.) Otro uso de las operaciones inversas es para encontrar un recíproco (1/X). Para lograr esto (si X está en la parte superior de la pila), se carga 1.0 a ST y después se coloca la instrucción FDIVR. Esta instrucción divide ST(1) entre ST o X entre 1 y deja el recíproco (1/X) en ST.

El modo de direcciónamiento de registro utiliza ST para la parte superior de la pila y ST(n) para otra posición, en donde n es el número de registro. Con esta forma, un operando debe ser ST y el otro ST(n). Observe que para duplicar la parte superior de la pila se utiliza la instrucción FADD ST,ST(0), en donde ST(0) también direcciona la parte superior de la pila. Uno de los dos operandos en el modo de direcciónamiento de registro debe ser ST, mientras que el otro debe estar en la forma ST(n), en donde n es uno de los registros de la pila del 0 al 7. Para muchas instrucciones, ST o ST(n) puede ser el destino. Es muy importante que la parte superior de la pila sea ST(0). Para lograr esto hay que reiniciar o inicializar el coprocesador antes de usarlo en un programa. Otro ejemplo de direcciónamiento de registro es la instrucción FADD ST(1), ST, en donde el contenido de ST se suma a ST(1) y el resultado se coloca en ST(1).

La parte superior de la pila siempre se utiliza como el destino para el modo de direcciónamiento de memoria, ya que el coprocesador es una máquina orientada a la pila. Por ejemplo, la instrucción FADD DATOS suma el contenido tipo número real de la posición de memoria DATOS a la parte superior de la pila.

Operaciones aritméticas. La letra P en un código de operación especifica la extracción (pop) de un registro después de la operación (comparación entre FADDP y FADD). La letra R en un código de operación (sólo resta y división) indica el modo inverso. Este modo es útil para datos en la memoria, debido a que por lo general éstos se restan de la parte superior de la pila. Una instrucción de resta inversa resta la parte superior de la pila de la memoria y almacena el resultado en la parte superior de la pila. Por ejemplo, si la parte superior de la pila contiene un 10 y la posición de memoria DATOS1 contiene un 1, la instrucción FSUB DATOS1 produce un +9 en la parte superior de la pila, en tanto que la instrucción FSUBR produce un -9. Otro ejemplo es FSUBR ST,ST(1), en el cual se resta ST de ST(1) y se almacena el resultado en ST. Una variante es FSUBR ST(1),ST, en la cual se resta ST(1) de ST y se almacena el resultado en ST(1).

La letra I, como segunda letra en un código de operación, indica que el operando de memoria es un entero. Por ejemplo, la instrucción FADD DATOS es una suma de punto flotante, mientras que la instrucción FIADD DATOS es una suma tipo entero que suma el entero de la posición de memoria DATOS con el número de punto flotante en la parte superior de la pila. La misma regla se aplica a las instrucciones FADD, FSUB, FMUL y FDIV.

Operaciones relacionadas con la aritmética. Hay otras operaciones de naturaleza aritmética: FSQRT (raíz cuadrada), FSCALE (escala un número), FPREM/FPREM1 (encuentra residuo parcial), FRNDINT (redondea a entero), FXTRACT (extrae exponente y mantisa), FABS (encuentra valor absoluto) y FCHG (cambia signo). A continuación se explican estas instrucciones y las funciones que realizan:

FSQRT	Encuentra la raíz cuadrada de la parte superior de la pila y deja el resultado ahí mismo. La raíz cuadrada de un número negativo produce un error inválido. Por esta razón debe evaluarse el bit IE del registro de estado cada vez que pueda ocurrir un
--------------	---

resultado inválido. El bit IE puede evaluarse si se carga el registro de estado hacia AX mediante la instrucción FSTSW AX, seguida de TEXT AX,1 para evaluar el bit de estado IE.

FSCALE	Suma el contenido de ST(1) (el cual se interpreta como entero) al exponente en la parte superior de la pila. FSCALE realiza una multiplicación o una división rápida en potencias de dos. El valor en ST(1) debe estar entre 2^{-15} y 2^{+15} .
FPREM/ FPREM1	Realiza la división módulo de ST entre ST(1). El residuo resultante se encuentra en la parte superior de la pila y tiene el mismo signo que el dividendo original. Observe, una división módulo produce un residuo sin cociente. Además, los coprocesadores 8086 y 80287 soportan FPREM, mientras que los coprocesadores más recientes deben usar FPREM1.
FRNDINT	Redondea la parte superior de la pila a un entero.
FXTRACT	Descompone el número en la parte superior de la pila en dos partes separadas que representan el valor del exponente sin desviación y el valor de la mantisa. La mantisa extraída se encuentra en la parte superior de la pila y el exponente sin desviación en ST(1). Esta instrucción se utiliza con frecuencia para convertir un número de punto flotante en un formato que pueda imprimirse como número mixto.
FABS	Cambia el signo de la parte superior de la pila a positivo.
FCHS	Cambia el signo de positivo a negativo, o de negativo a positivo.

Instrucciones de comparación

Todas las instrucciones de comparación examinan los datos en la parte superior de la pila en relación con otro elemento y devuelven el resultado de la comparación en los bits de código de condición C₃-C₀ del registro de estado. Las comparaciones permitidas por el coprocesador son FCOM (compara punto flotante), FCOMP (compara punto flotante con una extracción), FCOMPP (compara punto flotante con dos extracciones), FICOM (compara entero), FICOMP (compara entero con una extracción), FSTS (evalúa) y FXAM (examina). La instrucción FCOMI se introdujo con el Pentium Pro, en la que se compara un número de punto flotante y se mueve el resultado a las banderas. A continuación se muestra una lista de estas instrucciones, con una descripción de sus funciones:

FCOM	Compara los datos tipo punto flotante de la parte superior de la pila con un operando, el cual puede ser cualquier registro o cualquier operando de memoria. Si el operando no está codificado con la instrucción, el siguiente elemento de la pila ST(1) se compara con la parte superior de la misma, ST.
FCOMP/ FCOMPP	Ambas instrucciones funcionan como FCOM, pero también extraen (pop) uno o dos datos de la pila.
FICOM/ FICOMP	La parte superior de la pila se compara con el entero almacenado en un operando de memoria. Además de la comparación, FICOMP también extrae (pop) la parte superior de la pila.
FTST	Evalúa el contenido de la parte superior de la pila contra un cero. El resultado de la comparación se codifica en los bits de código de condición del registro de estado, como se muestra en la tabla 14-2 con el registro de estado. Además, en la tabla 14-3 será posible consultar una manera de utilizar SAHF y la instrucción de salto condicional con FTST.
FXAM	Examina la parte superior de la pila y modifica los bits de código de condición para que indiquen si el contenido es positivo, negativo, normalizado, etc. En la tabla 14-2, consulte el registro de estado.
FCOMI/ FUCOMI	Esta instrucción, nueva en los microprocesadores del Pentium Pro al Pentium 4, realiza una comparación de la misma forma que la instrucción FCOM, pero con una característica adicional: mueve las banderas de punto flotante hacia el registro de banderas, de igual forma que lo hacen las instrucciones FNSTSW AX y SAHF en el

ejemplo 14-8. Intel ha combinado las instrucciones FCOM, FNSTSW AX y SAHF para formar FCOMI. También está disponible la comparación desordenada, o FU-COMI. Cada una de estas instrucciones también está disponible con una extracción (pop) si se anexa una P al código de operación.

Operaciones trascendentales

Las instrucciones trascendentales son: FPTAN (tangente parcial), FPATAN (arcotangente parcial), FSIN (seno), FCOS (coseno), FSINCOS (seno y coseno), F2XM1 ($2^X - 1$), FYL2X ($Y \log_2 X$) y FYL2XP1 ($Y \log_2 (X + 1)$). A continuación se muestra una lista de estas operaciones con una descripción de cada operación trascendental:

FPTAN	Encuentra la tangente parcial de $Y/X = \tan \theta$. El valor de θ se encuentra en la parte superior de la pila. Debe estar entre 0 y $n/4$ radianes para los microprocesadores 8087 y 80287, y debe ser menor que 2^{63} para los microprocesadores 80387, 80486/7 y Pentium-Pentium 4. El resultado es una proporción que se encuentra como ST = X y ST(1) = Y. Si el valor está fuera del rango permitido se produce un error de inválido, como lo indica el bit IE del registro de estado. Además, ST(7) debe estar vacío para que esta instrucción funcione de manera apropiada.
FPATAN	Encuentra la arcotangente parcial como $\theta = \text{ARCTAN } X/Y$. El valor de X está en la parte superior de la pila y Y está en ST(1). Los valores de X y Y deben ser como se muestra a continuación: $0 \leq Y < X < \infty$. La instrucción saca datos de la pila y deja a θ en radianes en la parte superior de la pila.
F2XM1	Encuentra la función $2^X - 1$. El valor de X se toma de la parte superior de la pila y el resultado se devuelve a esa misma parte. Para obtener 2^X se suma uno al resultado de la parte superior de la pila. El valor de X debe estar en el rango de -1 a $+1$. La instrucción F2XM1 se utiliza para derivar las funciones que se muestran en la tabla 14-6. Observe que las constantes $\log_2 10$ y $\log_2 \varepsilon$ están integradas como valores estándar para el coprocesador.
FSIN/FCOS	Encuentra el seno o el coseno del argumento ubicado en ST, el cual se expresa en radianes ($360^\circ = 2\pi$ radianes), y el resultado se encuentra en ST. Los valores de ST deben ser menores que 2^{63} .
FSINCOS	Encuentra el seno y el coseno de ST, expresado en radianes, y deja los resultados como ST = seno y ST(1) = coseno. Al igual que con FSIN o FCOS, el valor inicial de ST debe ser menor que 2^{63} .
FYL2X	Encuentra $Y \log_2 X$. El valor X se toma de la parte superior de la pila y Y se toma de ST(1). El resultado se encuentra en la parte superior de la pila después de una extracción (pop). El valor de X debe estar entre 0 y ∞ , mientras que el valor de Y debe estar entre $-\infty$ y $+\infty$. Para encontrar un logaritmo con cualquier base positiva (b) se utiliza la ecuación $\text{LOG}_b X = (\text{LOG}_2 b)^{-1} \times \text{LOG}_2 X$.
FYL2P1	Encuentra $Y \log_2 (X + 1)$. El valor de X se toma de la parte superior de la pila y Y se toma de ST(1). El resultado se encuentra en la parte superior de la pila después de una extracción (pop). El valor de X debe estar entre 0 y $1 - \sqrt{2}/2$ y el valor de Y debe estar entre $-\infty$ y $+\infty$.

TABLA 14-6 Funciones exponenciales.

Función	Ecuación
10^Y	$2^Y \times \log_2 10$
ε^Y	$2^Y \times \log_2 \varepsilon$
X^Y	$2^Y \times \log_2 X$

TABLA 14-7 Operaciones de constantes.

Instrucción	Constante que se mete a ST
FLDZ	+0.0
FLD1	+1.0
FLDP1	π
FLDL2T	$\log_2 10$
FLDL2E	$\log_2 \varepsilon$
FLDLG2	$\log_{10} 2$
FLDLN2	$\log \varepsilon 2$

Operaciones de constantes

El conjunto de instrucciones del coprocesador incluye códigos de operación que devuelven constantes a la parte superior de la pila. En la tabla 14-7 aparece una lista de estas instrucciones.

Instrucciones de control del coprocesador

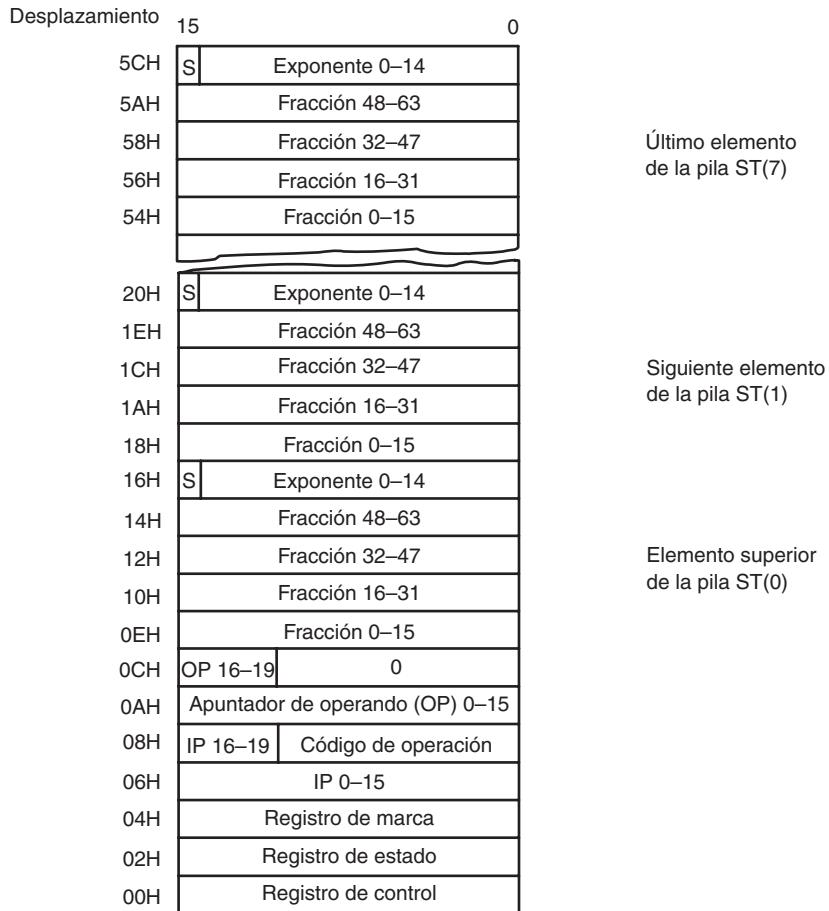
El coprocesador tiene instrucciones de control para inicialización, manejo de excepciones y commutación de tareas. Las instrucciones de control tienen dos formas. Por ejemplo, FINIT inicializa el coprocesador, al igual que FNINIT. La diferencia es que FNINIT no produce estados de espera, mientras que FINIT sí lo hace. Para esperar a la instrucción FINIT, el microprocesador evalúa la terminal BUSY en el coprocesador. Todas las instrucciones de control tienen estas dos formas. A continuación se muestra una lista de cada instrucción de control con su función:

FINIT/ FNINIT	Realiza una operación de reinicio (inicialización) en el coprocesador aritmético (en la tabla 14-8 consulte las condiciones de reinicio). El coprocesador opera con un cierre de proyectivo (infinito sin signo), redondea al más cercano o par, y utiliza precisión extendida cuando se reinicia o se inicializa. También establece el registro 0 como la parte superior de la pila.
FSETPM	Cambia el modo de direccionamiento del coprocesador al modo de direccionamiento protegido . Este modo se utiliza cuando el microprocesador también se opera en el modo protegido. Al igual que con el microprocesador, sólo se puede salir del modo protegido mediante un reinicio de hardware o, en el caso del 80386 al Pentium 4, con un cambio en el registro de control.

TABLA 14-8 Estado del coprocesador después de la instrucción FINIT.

Campo	Valor	Condición
Infinito	0	Proyectivo
Redondeo	00	Redondeo al más cercano
Precisión	11	Precisión extendida
Máscaras de error	11111	Bits de error deshabilitados
Ocupado	0	No ocupado
C_0-C_3	????	Desconocido
TOP	000	Registro 000 o ST(0)
ES	0	Sin errores
Bits de error	00000	Sin errores
Todas las marcas	11	Vacío
Registros	ST(0)-ST(7)	Sin cambio

FIGURA 14-8 Formato de memoria cuando se guardan los registros del 80X87 con la instrucción FSAVE.



FLDCW	Carga el registro de control con la palabra direccionada por el operando.
FSTCW	Almacena el registro de control en el operando de memoria tipo palabra.
FSTSW AX	Copia el contenido del registro de control al registro AX. Tal instrucción no está disponible para el coprocesador 8087.
FCLEX	Borra las banderas de error en el registro de estado y también la bandera de ocupado.
FSAVE	Escribe todo el estado de la máquina en la memoria. La figura 14-8 muestra la distribución de la memoria para esta instrucción.
FRSTOR	Restaura el estado de la máquina desde la memoria. Esta instrucción se utiliza para restaurar la información que guarda FSAVE.
FSTENV	Almacena el entorno del coprocesador, como se muestra en la figura 14-9.
FLDENV	Recarga el entorno que guarda FSTENV.
FINCSP	Incrementa el apuntador de la pila.
FDECSP	Decrementa el apuntador de la pila.
FFREE	Libera un registro mediante la modificación de la marca del registro de destino para vaciarlo. No afecta el contenido del registro.
FNOP	NOP del coprocesador de punto flotante.

FIGURA 14-9 Formato de memoria para la instrucción FSTENV:
a) modo real y b) modo protegido.

0CH	OP 16-19	0	0CH	Selector de operando
0AH	Apuntador de operando 0-15		0AH	Desplazamiento de operando
08H	IP 16-19	Código de operación	08H	Selector de CS
06H	Apuntador de instrucción 0-15		06H	Desplazamiento de IP
04H	Registro de marca		04H	Registro de marca
02H	Registro de estado		02H	Registro de estado
00H	Registro de control		00H	Registro de control

(a)

(b)

FWAIT Hace que el microprocesador espere a que el coprocesador termine una operación. FWAIT debe utilizarse antes de que el microprocesador acceda a datos en memoria que se vean afectados por el coprocesador.

Instrucciones del coprocesador

Aunque no hemos hablado sobre los circuitos del microprocesador, sí lo podemos hacer sobre los conjuntos de instrucciones de estos coprocesadores y sus diferencias en comparación con las demás versiones del coprocesador. Estos coprocesadores más recientes contienen las mismas instrucciones básicas que se proporcionan en las primeras versiones, con unas cuantas instrucciones adicionales.

Los microprocesadores 80387, 80486, 80487SX y del Pentium al Pentium 4 contienen las siguientes instrucciones adicionales: FCOS (coseno), FPREM1 (residuo parcial), FSIN (seno), FSINCOS (seno y coseno) y FUCOM/FUCOMP/FUCOMPP (comparación desordenada). Las instrucciones de seno y coseno son la adición más importante al conjunto de instrucciones. En las primeras versiones del coprocesador, el seno y el coseno se calculan a partir de la tangente. Los microprocesadores del Pentium Pro al Pentium 4 contienen dos nuevas instrucciones de punto flotante: FCMOV (un movimiento condicional) y FCOMI (compara y mueve a las banderas).

La tabla 14-9 muestra los conjuntos de instrucciones para todas las versiones del coprocesador. También lista el número de períodos de reloj que se requieren para ejecutar cada instrucción. Se listan los tiempos de ejecución para el 8087, 80287, 80387, 80486, 80487 y del Pentium al Pentium 4. (La sincronización para los microprocesadores del Pentium al Pentium 4 es la misma, ya que el coprocesador es idéntico en cada uno de ellos.) Para determinar el tiempo de ejecución de una instrucción, el tiempo de reloj se multiplica por el tiempo de ejecución listado. La instrucción FADD requiere de 70 a 143 períodos de reloj para el 80287. Suponga que se utiliza un reloj de 8 MHz con el 80287. El periodo de reloj es de 1/8 MHz, o 125 ns. La instrucción FADD requiere entre 8.75 µs y 17.875 µs para ejecutarse. Si se utiliza un 80486DX2 de 33 MHz (33 ns), esta instrucción requiere entre 0.264 µs y 0.66 µs para ejecutarse. En el Pentium, la instrucción FADD requiere de 1 a 7 períodos de reloj, por lo que si se opera a 133 MHz (7.52 ns), la instrucción requiere entre 0.00752 µs y 0.05264 µs. Los microprocesadores del Pentium Pro al Pentium 4 son mucho más rápidos que el Pentium. Para un Pentium 4 de 2 GHz, que tiene un periodo de reloj de 0.5 ns, la instrucción FADD tarda entre 0.5 ns y 3.5 ns.

La tabla 14-9 utiliza algunas notaciones abreviadas para representar el desplazamiento que pueda o no requerirse para una instrucción que utilice un modo de direccionamiento de memoria. También utiliza la abreviación *mmm* para representar un modo de direccionamiento de registro/memoria y *rrr* para representar uno de los registros del coprocesador de punto flotante: ST(0)-ST(7). El bit *d* (destino), que aparece en los códigos de operación de algunas instrucciones, define la dirección del flujo de datos, como en FADD ST,ST(2) o en FADD ST(2),ST. El bit *d* es un 0 lógico para el flujo hacia ST, como en FADD ST,ST(2), en donde ST guarda la suma después de la adición y un 1 lógico para FADD ST(2),ST, en donde ST(2) guarda la suma.

Además, observe que algunas instrucciones permiten elegir si se va a insertar o no un estado de espera. Por ejemplo, la instrucción FSTSW AX copia el registro de estado en AX. La instrucción FNSTSW AX también copia el registro de estado hacia AX, pero sin un estado de espera.

TABLA 14-9 El conjunto de instrucciones del coprocesador aritmético.

F2XM1 $2^{ST} - 1$					
11011001 11110000					
Ejemplo	Periodos de reloj				
F2XM1	8087	310-630			
	80287	310-630			
	80387	211-476			
	80486/7	140-279			
	Pentium-Pentium 4	13-57			
FABS Valor absoluto de ST					
11011001 11100001					
Ejemplo	Periodos de reloj				
FABS	8087	10-17			
	80287	10-17			
	80387	22			
	80486/7	3			
	Pentium-Pentium 4	1			
FADD/FADDP/FIADD Suma					
11011000 oo000mmm desp memoria de 32 bits (FADD) 11011100 oo000mmm desp memoria de 64 bits (FADD) 11011d00 11000rrr FADD ST,ST(rrr) 11011110 11000rrr FADDP ST,ST(rrr) 11011110 oo000mmm desp memoria de 16 bits (FIADD) 11011010 oo000mmm desp memoria de 32 bits (FIADD)					
Formato	Ejemplos				
FADD FADDP FIADD	FADD DATOS FADD ST,ST(1) FADDP FIADD NUMERO FIADD ST,ST(3) FIADDP ST,ST(2) FADD ST(2),ST	8087	70-143		
		80287	70-143		
		80387	23-72		
		80486/7	8-20		
		Pentium-Pentium 4	1-7		

FCLEX/FNCLEX		Borra errores		
11011011 11100010				
Ejemplo				Periodos de reloj
FCLEX FNCLEX		8087	2-8	
		80287	2-8	
		80387	11	
		80486/7	7	
		Pentium-Pentium 4	9	
FCOM/FCOMP/FCOMPP/FICOM/FICOMP		Comparación		
11011000 oo010mmm desp		memoria de 32 bits (FCOM)		
11011100 oo010mmm desp		memoria de 64 bits (FCOM)		
11011000 11010rrr		FCOM ST(rrr)		
11011000 oo011mmm desp		memoria de 32 bits (FCOMP)		
11011100 oo011mmm desp		memoria de 64 bits (FCOMP)		
11011000 11011rrr		FCOMP ST(rrr)		
11011110 11011001		FCOMPP		
11011110 oo010mmm desp		memoria de 16 bits (FICOM)		
11011010 oo010mmm desp		memoria de 32 bits (FICOM)		
11011110 oo011mmm desp		memoria de 16 bits (FICOMP)		
11011010 oo011mmm desp		memoria de 32 bits (FICOMP)		
Formato		Ejemplos	Periodos de reloj	
FCOM FCOMP FCOMPP FICOM FICOMP	FCOM ST(2) FCOMP DATOS FCOMPP FICOM NUMERO FICOMP DATOS3		8087	40-93
			80287	40-93
			80387	24-63
			80486/7	15-20
			Pentium-Pentium 4	1-8
FCOMI/FUCOMI/COMIP/FUCOMIP		Compara y carga banderas		
11011011 11110rrr FCOMI ST(rrr)				
11011011 11101rrr FUCOMI ST(rrr)				
11011111 11110rrr FCOMIP ST(rrr)				
11011111 11101rrr FUCOMIP ST(rrr)				
Formato		Ejemplos	Periodos de reloj	
FCOM FUCOMI FCOMIP FUCOMIP	FCOMI ST(2) FUCOMI ST(4) FCOMIP ST(0) FUCOMIP ST(1)		8087	—
			80287	—
			80387	—
			80486/7	—
			Pentium-Pentium 4	—

FCMOVcc Movimiento condicional				
11011010 11000rrr FCMOVB ST(rrr) 11011010 11001rrr FCMOVE ST(rrr) 11011010 11010rrr FCMOVBE ST(rrr) 11011010 11011rrr FCMOVU ST(rrr) 11011011 11000rrr FCMOVNB ST(rrr) 11011011 11001rrr FCMOVNE ST(rrr) 11011011 11010rrr FCMOVENBE ST(rrr) 11011011 11011rrr FCMOVNU ST(rrr)				
Formato	Ejemplos	Periodos de reloj		
FCMOVB FCMOVE	FCMOVB ST(2) FCMOVE ST(3)	8087 — 80287 — 80387 — 80486/7 — Pentium-Pentium 4 —		
FCOS Coseno de ST				
11011001 11111111				
Ejemplo	Periodos de reloj			
FCOS	8087 — 80287 — 80387 123-772 80486/7 193-279 Pentium-Pentium 4 18-124			
FDECSTP Decrementa apuntador de la pila				
11011001 11110110				
Ejemplo	Periodos de reloj			
FDECSTP	8087 6-12 80287 6-12 80387 22 80486/7 3 Pentium-Pentium 4 1			

FDISI/FNDISI		Deshabilita interrupciones	
11011011 11100001			
(Se ignora en el 80287, 80387, 80486/7, Pentium-Pentium 4)			
Ejemplo		Periodos de reloj	
FDISI	8087	2-8	
FNDISI	80287	—	
	80387	—	
	80486/7	—	
	Pentium-Pentium 4	—	
FDIV/FDIVP/FIDIV		División	
11011000 oo110mmm desp memoria de 32 bits (FDIV) 11011100 oo100mmm desp memoria de 64 bits (FDIV) 11011d00 11111rrr FDIV ST,ST(rrr) 11011110 11111rrr FDIVP ST,ST(rrr) 11011110 oo110mmm desp memoria de 16 bits (FIDIV) 11011010 oo110mmm desp memoria de 32 bits (FIDIV)			
Formato	Ejemplos	Periodos de reloj	
FDIV	FDIV DATOS	8087	191-243
FDIVP	FDIV ST,ST(3)	80287	191-243
FIDIV	FDIVP FIDIV NUMERO FIDIV ST,ST(5) FDIVP ST,ST(2) FDIV ST(2),ST	80387 80486/7 Pentium-Pentium 4	88-140 8-89 39-42
FDIVR/FDIVRP/FIDIVR		División invertida	
11011000 oo111mmm desp memoria de 32 bits (FDIVR) 11011100 oo111mmm desp memoria de 64 bits (FDIVR) 11011d00 11110rrr FDIVR ST,ST(rrr) 11011110 11110rrr FDIVRP ST,ST(rrr) 11011110 oo111mmm desp memoria de 16 bits (FIDIVR) 11011010 oo111mmm desp memoria de 32 bits (FIDIVR)			
Formato	Ejemplos	Periodos de reloj	
FDIVR	FDIVR DATOS	8087	191-243
FDIVRP	FDIVR ST,ST(3)	80287	191-243
FIDIVR	FDIVR P FIDIVR NUMERO FIDIVR ST,ST(5) FDIVRP ST,ST(2) FDIVR ST(2),ST	80387 80486/7 Pentium-Pentium 4	88-140 8-89 39-42

FENI/FNENI	Deshabilita interrupciones	
11011011 11100000		
(Se ignora en el 80287, 80387, 80486/7, Pentium-Pentium 4)		
Ejemplo		Periodos de reloj
FENI FNENI		8087 2-8 80287 — 80387 — 80486/7 — Pentium-Pentium 4 —
FFREE	Libera el registro	
11011101 1100rrr		
Formato	Ejemplos	Periodos de reloj
FFREE	FFREE FFREE ST(1) FFREE ST(2)	8087 9-16 80287 9-16 80387 18 80486/7 3 Pentium-Pentium 4 1
FINCSTP	Incrementa el apuntador de la pila	
11011001 11110111		
Ejemplo		Periodos de reloj
FINCSTP		8087 6-12 80287 6-12 80387 21 80486/7 3 Pentium-Pentium 4 1

FINIT/FNINIT		Inicializa el coprocesador
11011001 11110110		
Ejemplo		Periodos de reloj
FINIT	8087	2-8
FNINIT	80287	2-8
	80387	33
	80486/7	17
	Pentium-Pentium 4	12-16
FLD/FILD/FBLD		Carga datos en ST(0)
11011001 oo0000mmm desp memoria de 32 bits (FLD) 11011101 oo0000mmm desp memoria de 64 bits (FLD) 11011011 oo101mmm desp memoria de 80 bits (FLD) 11011111 oo0000mmm desp memoria de 16 bits (FILD) 11011011 oo0000mmm desp memoria de 32 bits (FILD) 11011111 oo101mmm desp memoria de 64 bits (FILD) 11011111 oo100mmm desp memoria de 80 bits (FBLD)		
Formato	Ejemplos	Periodos de reloj
FLD	FLD DATOS	8087
FILD	FILD DATOS1	80287
FBLD	FBLD DATOS_DEC	80387
		80486/7
		Pentium-Pentium 4
FLD1		Carga +1.0 en ST(0)
11011001 11101000		
Ejemplo		Periodos de reloj
FLD1	8087	15-21
	80287	15-21
	80387	24
	80486/7	4
	Pentium-Pentium 4	2

FLDZ	Carga +0.0 en ST(0)	
11011001 11101110		
Ejemplo		Periodos de reloj
FLDZ	8087	11-17
	80287	11-17
	80387	20
	80486/7	4
	Pentium-Pentium 4	2
FLDPI	Carga π en ST(0)	
11011001 11101011		
Ejemplo		Periodos de reloj
FLDPI	8087	16-22
	80287	16-22
	80387	40
	80486/7	8
	Pentium-Pentium 4	3-5
FLDL2E	Carga $\log_2 e$ en ST(0)	
11011001 11101010		
Ejemplo		Periodos de reloj
FLDL2E	8087	15-21
	80287	15-21
	80387	40
	80486/7	8
	Pentium-Pentium 4	3-5
FLDL2T	Carga $\log_2 10$ en ST(0)	
11011001 11101001		
Ejemplo		Periodos de reloj
FLDL2T	8087	16-22
	80287	16-22
	80387	40
	80486/7	8
	Pentium-Pentium 4	3-5

FLDLG2 Carga $\log_{10}2$ en ST(0)				
11011001 11101000				
Ejemplo			Periodos de reloj	
FLDLG2	8087	18-24		
	80287	18-24		
	80387	41		
	80486/7	8		
	Pentium-Pentium 4	3-5		
FLDLN2 Carga \log_e2 en ST(0)				
11011001 11101101				
Ejemplo			Periodos de reloj	
FLDLN2	8087	17-23		
	80287	17-23		
	80387	41		
	80486/7	8		
	Pentium-Pentium 4	3-5		
FLDCW Carga el registro de control				
11011001 oo101mmm desp				
Formato	Ejemplos	Periodos de reloj		
FLDCW	FLDCW DATOS FLDCW ESTADO	8087	7-14	
		80287	7-14	
		80387	19	
		80486/7	4	
		Pentium-Pentium 4	7	
FLDENV Carga el entorno				
11011001 oo100mmm desp				
Formato	Ejemplos	Periodos de reloj		
FLDENV	FLDENV ENTORNO FLDENV DATOS	8087	35-45	
		80287	25-45	
		80387	71	
		80486/7	34-44	
		Pentium-Pentium 4	32-37	

FMUL/FMULP/FIMUL		Multiplicación
Formato	Ejemplos	Periodos de reloj
FMUL	FMUL DATOS	8087
FMULP	FMUL ST,ST(2)	80287
FIMUL	FMUL ST(2),ST	80387
	FMULP	29-82
	FIMUL DATOS3	80486/7
		Pentium-Pentium 4
		1-7

FNOP		Sin operación
Ejemplo	Periodos de reloj	
FNOP	8087	
	10-16	
	80287	
	10-16	
	80387	
	12	
	80486/7	
	3	
	Pentium-Pentium 4	
	1	

FPATAN		Arco tangente parcial de ST(0)
Ejemplo	Periodos de reloj	
FPATAN	8087	
	250-800	
	80287	
	250-800	
	80387	
	314-487	
	80486/7	
	218-303	
	Pentium-Pentium 4	
	17-173	

FPREM Residuo parcial		
11011001 11111000		
Ejemplo	Periodos de reloj	
FPREM	8087	15-190
	80287	15-190
	80387	74-155
	80486/7	70-138
	Pentium-Pentium 4	16-64
FPREM1 Residuo parcial (IEEE)		
11011001 11110101		
Ejemplo	Periodos de reloj	
FPREM1	8087	—
	80287	—
	80387	95-185
	80486/7	72-167
	Pentium-Pentium 4	20-70
FPTAN Tangente parcial de ST(0)		
11011001 11110010		
Ejemplo	Periodos de reloj	
FPTAN	8087	30-450
	80287	30-450
	80387	191-497
	80486/7	200-273
	Pentium-Pentium 4	17-173
FRNDINT Redondea ST(0) a un entero		
11011001 11111100		
Ejemplo	Periodos de reloj	
FRNDINT	8087	16-50
	80287	16-50
	80387	66-80
	80486/7	21-30
	Pentium-Pentium 4	9-20

FRSTOR Restaura el estado			
11011101 oo110mmm desp			
Formato	Ejemplo	Periodos de reloj	
FRSTOR	FRSTOR DATOS FRSTOR ESTADO FRSTOR MAQUINA	8087	197-207
		80287	197-207
		80387	308
		80486/7	120-131
		Pentium-Pentium 4	70-95
FSAVE/FNSAVE Guarda el estado de la máquina			
11011101 oo110mmm desp			
Formato	Ejemplo	Periodos de reloj	
FSAVE	FSAVE ESTADO FNSAVE ESTATUS FSAVE MAQUINA	8087	197-207
		80287	197-207
		80387	375
		80486/7	143-154
		Pentium-Pentium 4	124-151
FSCALE Escala ST(0) con base en ST(1)			
11011001 11111101			
Ejemplo		Periodos de reloj	
FSCALE		8087	32-38
		80287	32-38
		80387	67-86
		80486/7	30-32
		Pentium-Pentium 4	20-31
FSETPM Establece el modo protegido			
11011011 11100100			
Ejemplo		Periodos de reloj	
FSETPM		8087	—
		80287	2-18
		80387	12
		80486/7	—
		Pentium-Pentium 4	—

FSIN	Seno de ST(0)	
11011001 11111110		
Ejemplo		Periodos de reloj
FSIN	8087	—
	80287	—
	80387	122-771
	80486/7	193-279
	Pentium-Pentium 4	16-126
FSINCOS	Encuentra el seno y el coseno de ST(0)	
11011001 11111011		
Ejemplo		Periodos de reloj
FSINCOS	8087	—
	80287	—
	80387	194-809
	80486/7	243-329
	Pentium-Pentium 4	17-137
FSQRT	Raíz cuadrada de ST(0)	
11011001 11111010		
Ejemplo		Periodos de reloj
FSQRT	8087	180-186
	80287	180-186
	80387	122-129
	80486/7	83-87
	Pentium-Pentium 4	70

FST/FSTP/FIST/FISTP/FBSTP		Almacena	
Formato	Ejemplos	Periodos de reloj	
FST	FST DATOS	8087	15-540
FSTP	FST ST(3)	80287	15-540
FIST	FIST	80387	11-534
FISTP	FISTP	80486/7	3-176
FBSTP	FIST DATOS2 FBSTP DATOS6 FISTO DATOS9	Pentium-Pentium 4	1-3

FSTCW/FNSTCW		Almacena el registro de control	
Formato	Ejemplos	Periodos de reloj	
FSTCW	FSTCW CONTROL	8087	12-18
FNSTCW	FNSTCW ESTADO	80287	12-18
	FSTCW MAQUINA	80387	15
		80486/7	3
		Pentium-Pentium 4	2

FSTENV/FNSTENV		Almacena el entorno	
Formato	Ejemplos	Periodos de reloj	
FSTENV	FSTENV CONTROL	8087	40-50
FNSTENV	FNSTENV ESTADO	80287	40-50
	FSTENV MAQUINA	80387	103-104
		80486/7	58-67
		Pentium-Pentium 4	48-50

FSTSW/FNSTSW		Almacena el registro de estado	
Formato	Ejemplos	Periodos de reloj	
FSTSW FNSTSW	FSTSW CONTROL	8087	12-18
	FNSTSW ESTADO	80287	12-18
	FSTSW MAQUINA	80387	15
	FSTSW AX	80486/7	3
		Pentium-Pentium 4	2-5
FSUB/FSUBP/FISUB		Resta	
11011000 oo100mmm desp		memoria de 32 bits (FSUB)	
11011100 oo100mmm desp		memoria de 64 bits (FSUB)	
11011d00 11101rrr		FSUB ST,ST(rrr)	
11011110 11101rrr		FSUBP ST,ST(rrr)	
11011110 oo100mmm desp		memoria de 16 bits (FISUB)	
11011010 oo100mmm desp		memoria de 32 bits (FISUB)	
Formato	Ejemplos	Periodos de reloj	
FSUB FSUBP FISUB	FSUB DATOS	8087	70-143
	FSUB ST,ST(2)	80287	70-143
	FSUB ST(2),ST	80387	29-82
	FSUBP	80486/7	8-35
	FISUB DATOS3	Pentium-Pentium 4	1-7
FSUBR/FSUBRP/FISUBR		Resta inversa	
11011000 oo101mmm desp		memoria de 32 bits (FSUBR)	
11011100 oo101mmm desp		memoria de 64 bits (FSUBR)	
11011d00 11100rrr		FSUBR ST,ST(rrr)	
11011110 11100rrr		FSUBRP ST,ST(rrr)	
11011110 oo101mmm desp		memoria de 16 bits (FISUBR)	
11011010 oo101mmm desp		memoria de 32 bits (FISUBR)	
Formato	Ejemplos	Periodos de reloj	
FSUBR FSUBRP FISUBR	FSUBR DATOS	8087	70-143
	FSUBR ST,ST(2)	80287	70-143
	FSUBR ST(2),ST	80387	29-82
	FSUBRP	80486/7	8-35
	FISUBR DATOS3	Pentium-Pentium 4	1-7

FTST	Compara ST(0) con + 0.0	
11011001 11100100		
Ejemplo		Periodos de reloj
FTST		8087 38-48 80287 38-48 80387 28 80486/7 4 Pentium-Pentium 4 1-4
FUCOM/FUCOMP/FUCOMPP	Comparación desordenada	
11011101 11100rrr 11011101 11101rrr 11011101 11101001	FUCOM ST,ST(rrr) FUCOMP ST,ST(rrr) FUCOMPP	
Formato	Ejemplos	Periodos de reloj
FUCOM FUCOMP FUCOMPP	FUCOM ST,ST(2) FUCOM FUCOMP ST,ST(3) FUCOMP FUCOMPP	8087 — 80287 — 80387 24-26 80486/7 4-5 Pentium-Pentium 4 1-4
FWAIT	Espera	
10011011		
Ejemplo		Periodos de reloj
FWAIT		8087 4 80287 3 80387 6 80486/7 1-3 Pentium-Pentium 4 1-3
FXAM	Examina ST(0)	
11011001 11100101		
Ejemplo		Periodos de reloj
FXAM		8087 12-23 80287 12-23 80387 30-38 80486/7 8 Pentium-Pentium 4 21

FXCH Intercambia ST(0) con otro registro			
11011001 11001rrr		FXCH ST,ST(rrr)	
Formato	Ejemplos	Periodos de reloj	
FXCH	FXCH ST,ST(1)	8087	10-15
	FXCH	80287	10-15
	FXCH ST,ST(4)	80387	18
		80486/7	4
		Pentium-Pentium 4	1
FXTRACT Extrae componentes de ST(0)			
11011001 11110100			
Ejemplo	Periodos de reloj		
FXTRACT		8087	27-55
		80287	27-55
		80387	70-76
		80486/7	16-20
		Pentium-Pentium 4	13
FYL2X ST(1) x log ₂ ST(0)			
11011001 11110001			
Ejemplo	Periodos de reloj		
FYL2X		8087	900-1100
		80287	900-1100
		80387	120-538
		80486/7	196-329
		Pentium-Pentium 4	22-111
FXL2XP1 ST(1) x log ₂ [ST(0) + 1.0]			
11011001 11111001			
Ejemplo	Periodos de reloj		
FXL2XP1		8087	700-1000
		80287	700-1000
		80387	257-547
		80486/7	171-326
		Pentium-Pentium 4	22-103
Datos: d = dirección, en donde d = 0 para ST como destino y d = 1 para ST como origen; rrr = número de registro de punto flotante; oo = modo; mmm = campo r/m; y desp = desplazamiento.			

14-4**PROGRAMACIÓN CON EL COPROCESADOR ARITMÉTICO**

En esta sección del capítulo proporcionaremos ejemplos para el coprocesador aritmético. Cada uno de los ejemplos se eligió para ilustrar una técnica de programación para el coprocesador.

Cálculo del área de un círculo

Este primer ejemplo de programación muestra un método simple para direccionar la pila del coprocesador. Primero hay que recordar que la ecuación para calcular el área de un círculo es $A = \pi R^2$. En el ejemplo 14-8 se muestra un programa que realiza este cálculo. Observe que este programa toma datos de prueba del arreglo RAD que contiene cinco radios de muestra. Las cinco áreas se almacenan en un segundo arreglo llamado ÁREA. En este programa no se utilizan los datos del arreglo ÁREA.

EJEMPLO 14-8

;Un procedimiento corto que encuentra las áreas de 5 círculos cuyos radios se
;almacenán en el arreglo RAD.

```
RAD      DD      2.34          ;arreglo de radios
        DD      5.66
        DD      9.33
        DD      234.5
        DD      23.4

ÁREA    DD      5 DUP(?)      ;arreglo para las áreas

BUSCA   PROC   NEAR
        FLDPI
        MOV ECX,0          ;carga pi
        .REPEAT
                FLD  RAD[ECX*4]  ;obtiene el radio
                FMUL ST,ST(0)    ;eleva el radio al cuadrado
                FMUL ST,ST(1)    ;multiplica el radio al cuadrado por pi
                FSTP ÁREA[ECX*4]  ;almacena el área
                INC   ECX         ;índice del siguiente radio
        .UNTIL ECX = 5       ;se repite 5 veces
        FCOMP
        RET
        BUSCA  ENDP
```

Aunque éste es un programa simple, nos muestra la operación de la pila. Para tener una mejor comprensión de la operación de la pila, la figura 14-10 muestra su contenido después de la ejecución de cada instrucción del ejemplo 14-8. Observe que sólo se muestra una pasada a través del ciclo, ya que el programa calcula cinco áreas y cada pasada es idéntica.

La primera instrucción carga π en la parte superior de la pila. A continuación se carga en la parte superior de la pila el contenido de la posición de memoria RAD [ECX*4], uno de los elementos del arreglo. Con esta operación se mete π en ST(1). Ahora, la instrucción FMUL ST,ST(0) eleva el radio

FIGURA 14-10 Operación de la pila del coprocesador para una iteración del ciclo en el ejemplo 14-8.

Instrucción	ST(0)	ST(1)
FLDPI	π	
FLD RAD[ECX*4]	2.34	π
FMUL ST,ST(0)	5.4756	π
FMUL ST,ST(1)	17.202	π
FSTP AREA[ECX*4]	π	

al cuadrado en la parte superior de la pila. La instrucción FMUL ST,ST(1) forma el área. Por último, la parte superior de la pila se almacena en el arreglo ÁREA y también se saca el resultado de la pila como preparación para la siguiente iteración.

Observe cómo siempre se tiene el cuidado de extraer todos los datos de la pila. La última instrucción antes de RET saca a π de la pila. Esto es importante, ya que si quedan datos en la pila al final del procedimiento, la parte superior de la pila ya no será el registro 0. Esto ocasionaría problemas debido a que el software asume que la parte superior de la pila es el registro 0. Otra manera de asegurar la inicialización del coprocesador es mediante la colocación de la instrucción FINIT (inicialización) al principio del programa.

Cómo hallar la frecuencia de resonancia

Una ecuación que se utiliza a menudo en electrónica es la fórmula para determinar la frecuencia resonante de un circuito LC. La ecuación que resuelve el programa del ejemplo 14-9 es

$$Fr = \frac{1}{2\pi\sqrt{LC}}$$

En este ejemplo se usa L1 para la inductancia L, C1 para el capacitor C y RES para la frecuencia de resonancia resultante.

EJEMPLO 14-9

```

RES      DD      ?          ;frecuencia de resonancia
L1       DD      0.0001    ;inductor de 1 mH
C1       DD      47E-6     ;capacitor de 47 μF

FR       PROC    NEAR

        FLD    L1          ;obtiene L
        FMUL   C1          ;forma LC
        FSQRT
        FLDPI
        FADD   ST,ST(0)    ;obtiene pi
        FADD   ST,ST(0)    ;forma 2 pi
        FMUL
        FLD1
        FDIVR
        FSTP   RES         ;forma el recíproco
        RET

FR       ENDP

```

Observe la manera tan simple en la que el programa resuelve tal ecuación. Se requiere muy poca manipulación de datos adicional debido a la pila dentro del coprocesador. También, cómo se utiliza FDIVR (mediante el uso del direccionamiento de pila clásico) para formar el recíproco. Si posee una calculadora de notación polaca inversa, como las que produce Hewlett-Packard, estará familiarizado con el direccionamiento de la pila. Si no, el uso del coprocesador aumentará su experiencia con este tipo de elementos.

Cómo encontrar las raíces mediante la ecuación cuadrática

En este ejemplo se muestra cómo encontrar las raíces de una expresión polinomial ($ax^2 + bx + c = 0$) con la ecuación cuadrática. Esta ecuación es:

$$b \pm \frac{\sqrt{b^2 - 4ac}}{2a}$$

El ejemplo 14-10 muestra un programa que encuentra las raíces (R_1 y R_2) de la ecuación cuadrática. Las constantes se almacenan en las posiciones de memoria A1, B1 y C1. Observe que no hay un intento por determinar si las raíces son imaginarias. En este ejemplo se evalúa el caso de raíces imaginarias; si se encuentran se produce una salida al DOS con un cero en las raíces (R_1 y R_2). En la práctica podrían resolverse las raíces imaginarias para almacenarlas en un conjunto separado de posiciones de memoria de resultado.

EJEMPLO 14-10

;Un procedimiento que encuentra las raíces de una ecuación polinomial mediante ;el uso de la ecuación cuadrática. Si R1 y R2 se regresan con cero, esto indica ;que se encontraron raíces imaginarias.

```

CUATRO DW 4          ;entero de 4
A1    DD ?           ;valor para a
B1    DD ?           ;valor para b
C1    DD ?           ;valor para c
R1    DD ?           ;raíz 1
R2    DD ?           ;raíz 2

RAÍCES PROC NEAR

        FLDZ             ;obtiene 0.0
        FST   R1           ;borra las raíces
        FSTP  R2
        FLD   A1           ;de 2a
        FADD  ST, ST(0)
        FILD  CUATRO       ;obtiene un 4
        FMUL  A1           ;de 4ac
        FMUL  C1
        FLD   B1           ;de  $b^2$ 
        FMUL  ST, ST(0)
        FSUBR            ;de  $b^2 - 4ac$ 
        FTST
        SAHF
        .IF !ZERO?
                FSQRT           ;encuentra la raíz cuadrada de  $b^2 - 4ac$ 
                FSTSW AX          ;comprueba si hay error inválido
                TEST AX, 1
                .IF !ZERO?
                        FCOMPP          ;borra la pila
                        RET
                .ENDIF
        .ENDIF
        FLD   B1
        FSUB  ST, ST(1)
        FDIV  ST, ST(2)
        FSTP  R1           ;almacena la raíz 1
        FLD   B1
        FADD
        FDIVR
        FSTP  R2           ;almacena la raíz 2
        RET

RAÍCES ENDP

```

Uso de un arreglo de memoria para almacenar los resultados

En el siguiente ejemplo de programación se ilustran el uso de un arreglo de memoria y el modo de direccionamiento por índice escalado para acceder al arreglo. El ejemplo 14-11 muestra un programa que calcula 100 valores de reactancia inductiva. La ecuación para la reactancia inductiva es $XL = 2\pi FL$. En este ejemplo, el rango de frecuencia está entre 10 Hz y 1000 Hz para F y una inductancia de 4 mH. Observe cómo se utiliza la instrucción FSTP XL[ECX*4 + 4] para almacenar la reactancia de cada frecuencia, comenzando con la última a 1000 Hz y terminando con la primera a 10 Hz. También, cómo se utiliza la instrucción FCOMP para borrar la pila justo antes de la instrucción RET.

EJEMPLO 14-11

;Un procedimiento que calcula la reactancia inductiva de L a ;frecuencias de 10 Hz a 1000 Hz en intervalos de 10 Hz y las almacena ;en el arreglo llamado XL.

```

XL      DD 100 DUP(?)      ;arreglo para XL
L       DD 4E-3            ;L = 4 mH
F       DW 10              ;entero de 10 para F

```

```

XLS      PROC    NEAR

        MOV     ECX,100          ;cuenta = 100
        FLDPI
        FADD   ST,ST(0)         ;obtiene pi
        FMUL   L
        .REPEAT
                FILD   F           ;obtiene F
                FMUL   ST,ST(1)       ;encuentra XL
                FSTP   XL[ECX*4+4]    ;almacena el resultado
                MOV    AX,F            ;suma 10 a F
                ADD    AX,10
                MOV    F,AX
        .UNTILCXZ
        FCOMP
        RET

XLS      ENDP

```

Conversión de un número de punto flotante con precisión simple en una cadena

En esta sección mostraremos cómo tomar el contenido tipo punto flotante de un número de punto flotante con precisión simple de 32 bits y almacenarlo en una cadena de caracteres ASCII. El procedimiento convierte el número de punto flotante como un número mixto con una parte entera y una parte fraccionaria, separadas por un punto decimal. Para simplificar el procedimiento, se coloca un límite en el tamaño del número mixto, de forma que la porción entera sea un número binario de 32 bits (± 2 G) y que la fracción sea un número binario de 24 bits (1/16 M). El procedimiento no funcionará de manera apropiada para números más grandes o más pequeños.

El ejemplo 14-12 muestra un procedimiento que convierte el contenido de la posición de memoria NUME en una cadena que se almacena en el arreglo CAD. El procedimiento primero evalúa el signo del número y almacena un signo de resta para un número negativo. Después de almacenar un signo de resta, si es necesario el número se convierte en positivo mediante la instrucción FABS. A continuación, se divide en una parte entera y fraccionaria, y se almacena en ENTERO y FRACC. Observe cómo se utiliza la instrucción FRNDINT para redondear (mediante el modo de corte) la parte superior de la pila para formar la parte correspondiente al número entero de NUME. Después se resta esta parte del número original para generar la parte fraccionaria. Esto se logra mediante la instrucción FSUB, la cual resta de ST el contenido de ST(1).

EJEMPLO 14-12

;Un procedimiento que convierte un número de punto flotante en una
;cadena de caracteres ASCII

```

CAD      DB      40 DUP(?)          ;almacenamiento para la cadena
NUME     DD      -2224.125         ;evalúa el número
ENTERO   DD      ?
FRACC   DD      ?
TEMP    DW      ?                 ;lugar para CW
DIEZ    DW      10                ;entero de 10

FTOA    PROC    NEAR USES EBX ECX EDX

        MOV     ECX,0          ;inicializa el apuntador
        FSTCW  TEMP          ;almacena la palabra de control actual
        MOV    AX,TEMP         ;cambia el redondeo a modo de corte
        PUSH   AX
        OR     AX,0C00H
        MOV    TEMP,AX
        FLDCW  TEMP
        FTST   NUME          ;evalúa NUME
        FSTSW  AX
        AND    AX,4500H        ;obtiene C0, C2 y C3
        .IF AX == 100H         ;si es negativo
                MOV CAD[ECX], '-'
                INC ECX

```

```

        FABS           ;lo hace positivo
.ENDIF
FRNDINT        ;redondea a entero
FIST    ENTERO   ;almacena la parte del número entero
FLD     NUME      ;forma y almacena la fracción
FABS
FSUBR
FSTP    FRACC
MOV     EAX,ENTERO ;convierte la parte entera
MOV     EBX,10
PUSH    EBX
.REPEAT
        MOV     EDX,0
        DIV     EBX
        ADD     DL,30H ;convierte en ASCII
        PUSH    EDX
.UNTIL EAX == 0
POP     EDX
MOV     AH,3          ;contador de comas
.WHILE EDX != 10      ;parte entera a ASCII
        POP     EBX
        DEC     AH
        .IF AH == 0 && EBX != 10
                MOV CAD[ECX],','
                INC ECX
                MOV AH,3
        .ENDIF
        MOV CAD[ECX],DL
        INC ECX
        MOV EDX,EBX
.ENDW
MOV     CAD[ECX],'.'
;almacena el punto decimal
INC     ECX
POP     TEMP        ;restaura CW original
FLDCW  TEMP
FLD    FRACT       ;convierte parte fraccionaria
.REPEAT
        FIMUL   DIEZ
        FIST    TEMP
        MOV     AX,TEMP
        ADD     AL,30H
        MOV     CAD[ECX],AL
        INC     ECX
        FISUB  TEMP
        FXAM
        SAHF
.UNTIL ZERO?
FCOMP
MOV     CAD[ECX],0  ;borra la pila
;almacena nulo
RET
FTOA    ENDP

```

14-5

INTRODUCCIÓN A LA TECNOLOGÍA MMX

La tecnología MMX¹ (**extensiones multimedia**) agrega 57 instrucciones nuevas al conjunto de instrucciones de los microprocesadores del Pentium al Pentium 4. La tecnología MMX también introduce nuevas instrucciones de propósito general. Las nuevas instrucciones MMX se diseñaron para aplicaciones tales como vídeo en movimiento, la combinación de gráficas con vídeo, el procesamiento de imágenes, la síntesis de audio, la síntesis y compresión de voz, telefonía, videoconferencias, gráficos 2D y 3D. Estas instrucciones (nuevas desde el Pentium) trabajan en paralelo con otras operaciones, como las instrucciones para el coprocesador aritmético.

¹ MMX es marca registrada de Intel Corporation.

Tipos de datos

La arquitectura MMX introduce nuevos tipos de datos empaquetados. Estos tipos de datos son ocho bytes empaquetados de 8 bits consecutivos; cuatro palabras empaquetadas de 16 bits consecutivos y dos dobles palabras empaquetadas de 32 bits consecutivos. Los bytes, en este formato multibyte, tienen direcciones de memoria consecutivas y utilizan el formato Little Endian, al igual que otros datos de Intel. En la figura 14-11 verá el formato para estos nuevos tipos de datos.

Los registros de la tecnología MMX tienen el mismo formato que una cantidad de 64 bits en memoria; además, dos modos de acceso a los datos: modo de acceso de 64 bits y modo de acceso de 32 bits. El modo de acceso de 64 bits se utiliza para la memoria de 64 bits y las transferencias de registros en la mayoría de las instrucciones. El modo de acceso de 32 bits se utiliza para la memoria de 32 bits y también para las transferencias de registros en la mayoría de las instrucciones. Las transferencias de 32 bits se llevan a cabo entre los registros del microprocesador y las transferencias de 64 bits entre los registros del coprocesador de punto flotante.

La figura 14-12 muestra el conjunto de registros internos de la extensión de tecnología MMX y la manera en que utiliza el conjunto de registros del coprocesador de punto flotante. Esta técnica se conoce como renombramiento (**aliasing**) porque los registros de punto flotante se comparten como registros MMX. Esto es, los registros MMX (MM₀-MM₇) son los mismos que los registros de punto flotante. Tenga en cuenta que el conjunto de registros MMX es de 64 bits y utiliza los 64 bits de más a la derecha del conjunto de registros de punto flotante.

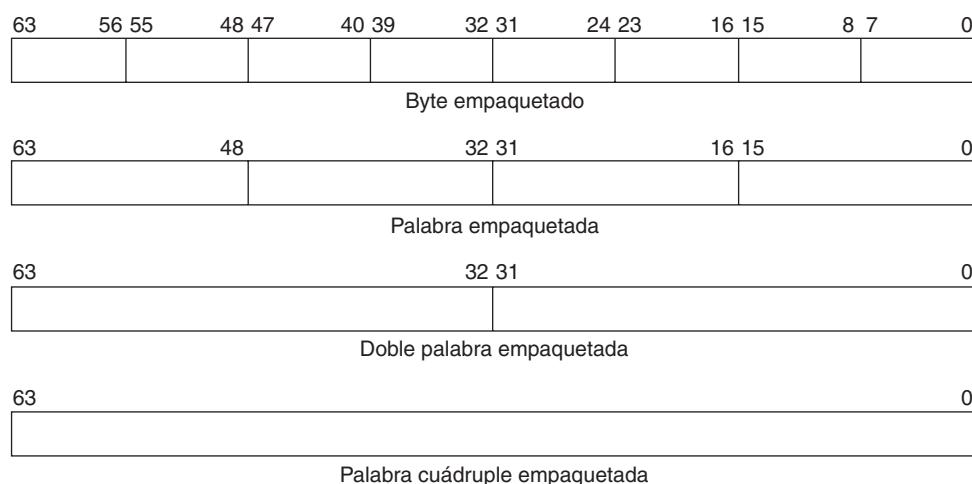


FIGURA 14-11 La estructura de los datos almacenados en los registros MMX.

FIGURA 14-12 La estructura del conjunto de registros MMX. Observe que los registros de MM₀ y FF₀ hasta MM₇ y FP₇ se intercambian entre sí.



Conjunto de instrucciones

Los tipos de instrucciones para la tecnología MMX son: aritméticas, de comparación, lógicas, de desplazamiento y de transferencia de datos. Aunque los tipos de instrucciones son similares al conjunto de instrucciones del microprocesador, la principal diferencia es que las instrucciones MMX utilizan los tipos de datos que se muestran en la figura 14-11, en vez de los tipos de datos normales que utiliza el microprocesador.

Instrucciones aritméticas. El conjunto de instrucciones aritméticas incluye: suma, resta, multiplicación, una multiplicación especial con una suma, etcétera. Existen tres sumas. Las instrucciones PADD y PSUB suman o restan bytes empaquetados con signo o sin signo, palabras empaquetadas o dobles palabras empaquetadas. Hay que anexar una B, W o D a las instrucciones de suma para seleccionar el tamaño, como en PADDB para un byte, PADDW para una palabra y PADDD para una doble palabra. Lo mismo se aplica para la instrucción PSUB. Las instrucciones PMULHW y PMULLW realizan multiplicaciones en cuatro pares de operandos de 16 bits, lo cual produce resultados de 32 bits. La instrucción PMULHW multiplica los 16 bits de mayor orden y la instrucción PMULLW multiplica los 16 bits de menor orden. La instrucción PMADDWD multiplica y suma. Después de multiplicar, los cuatro resultados de 32 bits se suman para producir dos resultados tipo doble palabra de 32 bits.

Las instrucciones MMX utilizan operandos al igual que las instrucciones de enteros o de punto flotante. La diferencia está en los nombres de los registros (MM₀-MM₇). Por ejemplo, la instrucción PADDB MM₁, MM₂ suma todo el contenido de 64 bits de MM₂ a MM₁, byte por byte. El resultado se lleva a MM₁. Cuando se suma cada sección de 8 bits se descarta cualquier acarreo que se genere. Por ejemplo, si el byte A0H se suma al byte 70H se produce el resultado de 10H. La suma verdadera es 110H, pero el acarreo se descarta. El segundo operando u origen puede ser una posición de memoria que contenga el origen empaquetado de 64 bits o un registro MMX. Podría decirse que esta instrucción realiza la misma función que ocho instrucciones ADD tipo byte separadas. Si se utiliza en una aplicación, esto en verdad agiliza la ejecución de la aplicación. Al igual que PADD, PSUB tampoco considera los acarreos. La diferencia es que si se produce un desbordamiento o subdesbordamiento, se convierte en 7FH (+127) para un desbordamiento y en 80H (-128) para un subdesbordamiento. Intel llama a esto **saturación**, debido a que estos valores representan los bytes con signo más grande y más pequeño.

Instrucciones de comparación. Existen dos instrucciones de comparación: PCMPEQ (igual) y PCMPGT (mayor que). Al igual que con PADD y PSUB, hay tres versiones de cada instrucción de comparación: por ejemplo, PCMPEQUB (compara bytes), PCMPEQUW (compara palabras) y PCMPEQUD (compara dobles palabras). Estas instrucciones no modifican los bits de bandera del microprocesador, sino que el resultado es sólo unos para una condición verdadera y sólo ceros para una condición falsa. Por ejemplo, si se ejecuta la instrucción PCMPEQB MM₂, MM₃, y los bytes menos significativos de MM₂ y MM₃ = 10H y 11H, respectivamente, el resultado que se encontraría en el byte menos significativo de MM₂ sería 00H. Esto indicaría que los bytes menos significativos no eran iguales. Si el byte menos significativo fuera FFH, esto indicaría que los dos bytes eran iguales.

Instrucciones de conversión. Hay dos instrucciones básicas de conversión: PACK y PUNPCK. PACK está disponible como PACKSS (saturación con signo) y PACKUS (saturación sin signo). PUNPCK está disponible como PUNPCKH (desempaquetar datos de parte superior) y PUNPCKL (desempaquetar datos de parte inferior). De manera similar a las instrucciones anteriores, se puede anexar B, W o D a las instrucciones para empaquetar y desempaquetar bytes, palabras o dobles palabras; pero deben usarse en las combinaciones WB (palabra a byte) o DW (doble palabra a palabra). Por ejemplo, la instrucción PACKUSWB MM₃ empaqueta las palabras de MM₆ en bytes en MM₃. Si la palabra sin signo no cabe (demasiado grande) en un byte, el byte de destino se convierte en FFH. Para la saturación con signo utilizamos los mismos valores que explicamos para la suma.

Instrucciones lógicas. Las instrucciones lógicas son PAND (AND), PANDN (NAND), POR (OR) y PXOR (OR exclusivo). Estas instrucciones no tienen extensiones de tamaño y realizan tales operaciones a nivel de bit en todos los 64 bits de datos. Por ejemplo, la instrucción POR MM₂, MM₃, aplica un OR a todos los 64 bits de MM₃ con MM₂. La suma lógica se coloca en MM₂ después de la operación OR.

Instrucción de desplazamiento. Esta instrucción contiene desplazamientos lógicos y una instrucción de desplazamiento aritmético a la derecha. Los desplazamientos lógicos son PSLL (izquierda) y

PSRL (derecha). Las variaciones son tipo palabra (W), doble palabra (D) y palabra cuádruple (Q). Por ejemplo, la instrucción PSLLQ MM_{3,2} desplaza todos los 64 bits en MM₃ dos posiciones a la izquierda. Otro ejemplo es la instrucción PSLLD MM_{3,2}, la cual desplaza las dos dobles palabras de 32 bits en MM₃ dos posiciones a la izquierda cada una.

La instrucción PSRA (desplazamiento aritmético a la derecha) funciona de la misma forma que los desplazamientos lógicos, sólo que se preserva el bit de signo.

Instrucciones de transferencia de datos. Hay dos instrucciones de transferencia de datos: MOVED y MOVEQ. Tales instrucciones permiten transferencias entre registros y entre un registro y la memoria. La instrucción MOVED transfiere 32 bits de datos entre un registro entero o una posición de memoria y un registro MMX. Por ejemplo, la instrucción MOVED ECX, MM₂ copia los 32 bits de más a la derecha de MM₂ hacia ECX. No hay una instrucción para transferir los 32 bits de más a la izquierda de un registro MMX. Podría utilizarse un desplazamiento a la derecha antes de una instrucción MOVED para realizar esa transferencia.

La instrucción MOVEQ copia todos los 64 bits de un registro MMX entre la memoria u otro registro MMX. La instrucción MOVEQ MM₂, MM₃ transfiere todos los 64 bits de MM₃ hacia MM₂.

Instrucción EMMS. La instrucción EMMS (vacía el estado de MMX) activa todas (11) las marcas en la unidad de punto flotante, de manera que los registros de punto flotante se listen como vacíos. La instrucción EMMS debe ejecutarse antes de la instrucción de retorno al final de cualquier procedimiento MMX; de lo contrario, la siguiente operación de punto flotante producirá un error de interrupción de punto flotante, haciendo que Windows o cualquiera otra aplicación fallen. Si planea utilizar instrucciones de punto flotante dentro de un procedimiento MMX, debe utilizar la instrucción EMMS antes de ejecutar esa instrucción de punto flotante. Todas las demás instrucciones MMX borran las marcas, lo cual indica que todos los registros de punto flotante están en uso.

Listado de instrucciones. La tabla 14-10 muestra todas las instrucciones MMX con el código máquina para que puedan utilizarse con el ensamblador. En este momento, MASM no soporta estas nuevas instrucciones, a menos que se haya actualizado a la versión más reciente (6.15). Encuentra esta versión en el kit de desarrollo de controladores de Windows (DDK de Windows), el cual se obtiene de Microsoft Corporation por un pequeño costo de envío. También está disponible en Visual Studio .NET 2003 (busque ML.EXE). Es posible usar cualquier instrucción MMX en Visual C++ mediante el ensamblador en línea.

Ejemplo de programación. El ejemplo 14-13 muestra un ejemplo simple de programación que utiliza las instrucciones MAX para realizar una tarea que requiere ocho veces más tiempo que si se usan instrucciones normales del microprocesador. En este ejemplo se suma un arreglo de 1000 bytes de datos (BLOQUEA) a un segundo arreglo de 1000 bytes (BLOQUEB). El resultado se almacena en un tercer arreglo llamado BLOQUEC. El ejemplo 14-13(a) muestra un procedimiento que utiliza lenguaje ensamblador tradicional para realizar la suma y el ejemplo 14-13(b) muestra el mismo proceso mediante el uso de instrucciones MMX.

EJEMPLO 14-13(A)

```
; Procedimiento que suma BLOQUEA a BLOQUEB y almacena las sumas en BLOQUEC

BLOQUEA DB      1000 DUP (?)
BLOQUEB DB      1000 DUP (?)
BLOQUEC DB      1000 DUP (?)

SUMA     PROC    NEAR

        MOV     ECX, 1000
        .REPEAT
                MOV     AL, BLOQUEA[ECX-1]
                ADD     AL, BLOQUEB[ECX-1]
                MOV     BLOQUEC[ECX-1]
        .UNTILCXZ
        RET

SUMA     ENDP
```

TABLA 14-10 El conjunto de instrucciones de las extensiones MMX.

EMMS	Vacia el estado MMX
0000 1111 0111 1111	
Ejemplo	
EMMS	
MOVED	Mueve doble palabra
0000 1111 0110 1110 11 xxx rrr	reg→xreg
Ejemplos	
MOVED MM3, EDX	
MOVED MM4, EAX	
0000 1111 0111 1110 11 xxx rrr	xreg→reg
Ejemplos	
MOVED EAX, MM3	
MOVED EBP, MM7	
0000 1111 0110 1110 oo xxx mmm	mem→xreg
Ejemplos	
MOVED MM3, DATOS1	
MOVED MM5, UNO_GRANDE	
0000 1111 0111 1110 oo xxx mmm	xreg→mem
Ejemplos	
MOVED DATOS2, MM3	
MOVED SMALL_POTS, MM7	
MOVEQ	Mueve palabra cuádruple
0000 1111 0110 1111 11 xxx1 xxx2	xreg2→xreg1
Ejemplos	
MOVEQ MM3, MM2 ;copia MM2 a MM3	
MOVEQ MM7, MM3	
0000 1111 0111 1111 11 xxx1 xx2	xreg1→xreg2
Ejemplos	
MOVEQ MM3, MM2 ;copia MM3 a MM2	
MOVEQ MM7, MM3	
0000 1111 0110 1111 oo xxx mm	mem→xreg
Ejemplos	
MOVEQ MM3, DATOS1	
MOVEQ MM5, DATOS3	
0000 1111 0111 1111 oo xxx mmm	xreg→mem
Ejemplos	
MOVEQ DATOS2, MM0	
MOVEQ SMALL_POTS, MM3	

PACKSSDW	Empaqueta doble palabra con signo en palabra	
0000 1111 0110 1011	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
PACKSSDW MM1, MM2 PACKSSDW MM7, MM3		
0000 1111 0111 1011	oo xxx mmm	mem→xreg
Ejemplos		
PACKSSDW MM3, BOTON PACKSSDW MM7, SONIDO		
PACKSSWB	Empaqueta palabra con signo en byte	
0000 1111 0110 0011	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
PACKSSWB MM1, MM2 PACKSSWB MM7, MM3		
0000 1111 0111 0011	oo xxx mmm	mem→xreg
Ejemplos		
PACKSSWB MM3, BOTON PACKSSWB MM7, SONIDO		
PACKUSWB	Empaqueta palabra sin signo en byte	
0000 1111 0110 0111	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
PACKUSWB MM1, MM2 PACKUSWB MM7, MM3		
0000 1111 0111 0111	oo xxx mmm	mem→xreg
Ejemplos		
PACKUSWB MM3, BOTON PACKUSWB MM7, SONIDO		
PADD	Suma con truncamiento	Byte, palabra y doble palabra
0000 1111 1111 11gg	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
PADD MM1, MM2 PADDW MM7, MM3 PADDD MM3, MM4		
0000 1111 1111 11gg	oo xxx mmm	mem→xreg
Ejemplos		
PADD MM3, BOTON PADDW MM7, SONIDO PADDD MM3, MANTEQUILLA		

PADDS	Suma con saturación con signo	Byte y palabra
0000 1111 1110 11gg	11 xxx1 xx2	xreg2→xreg1
Ejemplos		
PADD\$B MM1, MM2		
PADD\$W MM7, MM3		
0000 1111 1110 11gg	oo xxx mmm	mem→xreg
Ejemplos		
PADD\$B MM3, BOTON		
PADD\$W MM7, SONIDO		
PADDUS	Suma con saturación sin signo	Byte y palabra
0000 1111 1101 11gg	11 xxx1 xx2	xreg2→xreg1
Ejemplos		
PADD\$UB MM1, MM2		
PADD\$UW MM7, MM3		
0000 1111 1101 11gg	oo xxx mmm	mem→xreg
Ejemplos		
PADD\$UB MM3, BOTON		
PADD\$UW MM7, SONIDO		
PAND	And	
0000 1111 1101 1011	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
PAND MM1, MM2		
PAND MM7, MM3		
0000 1111 1101 1011	oo xxx mmm	mem→xreg
Ejemplos		
PAND MM3, BOTON		
PAND MM7, SONIDO		
PAND	Nand	
0000 1111 1101 1111	11 xxx1 xx2	xreg2→xreg1
Ejemplos		
PANDN MM1, MM2		
PANDN MM7, MM3		
0000 1111 1101 1111	oo xxx mmm	mem→xreg
Ejemplos		
PANDN MM3, BOTON		
PANDN MM7, SONIDO		

PCMPEQU	Compara igualdad	Byte, palabra y doble palabra
0000 1111 0111 01gg	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
PCMPEQUB MM1, MM2		
PCMPEQUW MM7, MM3		
PCMPEQUD MM0, MM5		
0000 1111 0111 01gg	oo xxx mmm	mem→xreg
Ejemplos		
PCMPEQUB MM3, BOTON		
PCMPEQUW MM7, SONIDO		
PCMPEQUD MM0, RANA		
PCMPGT	Compara para mayor que	Byte, palabra y doble palabra
0000 1111 0110 01gg	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
PCMPGTTB MM1, MM2		
PCMPGTTW MM7, MM3		
PCMPGTD MM0, MM5		
0000 1111 0110 01gg	oo xxx mmm	mem→xreg
Ejemplos		
PCMPGTTB MM3, BOTON		
PCMPGTTW MM7, SONIDO		
PCMPGTD MM0, RANA		
PMADD	Multiplica y suma	
0000 1111 1111 0101	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
PMADD MM1, MM2		
PMADD MM7, MM3		
0000 1111 1111 0101	oo xxx mmm	mem→xreg
Ejemplos		
PMADD MM3, BOTON		
PMADD MM7, SONIDO		
PMULH	Multiplicación-parte superior	
0000 1111 1110 0101	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
PMULH MM1, MM2		
PMULH MM7, MM3		
0000 1111 1110 0101	oo xxx mmm	mem→xreg
Ejemplos		
PMULH MM3, BOTON		
PMULH MM7, SONIDO		

PMULL	Multiplicación-parte inferior	
0000 1111 1101 0101	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
PMULL MM1, MM2 PMULL MM7, MM3		
0000 1111 1101 0101	oo xxx mmm	mem→xreg
Ejemplos		
PMULL MM3, BOTON PMULL MM7, SONIDO		
POR	Or	
0000 1111 1110 1011	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
POR MM1, MM2 POR MM7, MM3		
0000 1111 1110 1011	oo xxx mmm	mem→xreg
Ejemplos		
POR MM3, BOTON POR MM7, SONIDO		
PSLL	Desplazamiento a la izquierda	Palabra, doble palabra y palabra cuádruple
0000 1111 1111 00gg	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
PSLLW MM1, MM2 PSLLD MM7, MM3 PSLLLQ MM6, MM5		
0000 1111 1111 00gg	oo xxx mmm	mem→xreg
cuenta de desplazamiento en memoria		
Ejemplos		
PSLLW MM3, BOTON PSLLD MM7, SONIDO PSLLLQ MM2, CUENTA1		
0000 1111 0111 00gg	11 110 mmm	datos8
xreg con base en cuenta la cuenta de desplazamiento es datos8		
Ejemplos		
PSLLW MM3, 2 PSLLD MM0, 6 PSLLLQ MM7, 1		

PSRA	Desplazamiento aritmético a la derecha	Palabra, doble palabra y palabra cuádruple
0000 1111 1110 00gg	11 xx1 xxx2	xreg2→xreg1
Ejemplos		
PSRAW MM1, MM2 PSRAD MM7, MM3 PSRAQ MM6, MM5		
0000 1111 1110 00gg	oo xxx mmm	mem→xreg cuenta de desplazamiento en memoria
Ejemplos		
PSRAW MM3, BOTON MM7, SONIDO PSRAQ MM2, CUENTA1		
0000 1111 0111 00gg	11 100 mmm	datos8
		xreg con base en cuenta la cuenta de desplazamiento es datos8
Ejemplos		
PSRAW MM3, 2 PSRAD MM0, 6 PSRAQ MM7, 1		
PSRL	Desplazamiento a la derecha	Palabra, doble palabra y palabra cuádruple
0000 1111 1101 00gg	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
PSRLW MM1, MM2 PSRLD MM7, MM3 PSRLQ MM6, MM5		
0000 1111 1101 00gg	oo xxx mmm	mem→xreg cuenta de desplazamiento en memoria
Ejemplos		
PSRLW MM3, BOTON PSRLD MM7, SONIDO PSRLQ MM2, CUENTA1		
0000 1111 0111 00gg	11 010 mm	datos8
		xreg con base en cuenta la cuenta de desplazamiento es datos8
Ejemplos		
PSRLW MM3, 2 PSRLD MM0, 6 PSRLQ MM7, 1		

PSUB	Resta con truncamiento	Byte, palabra y doble palabra
0000 1111 1111 10gg	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
PSUBB MM1, MM2		
PSUBW MM7, MM3		
PSUBD MM3, MM4		
0000 1111 1111 10gg	oo xxx mmm	mem→xreg
Ejemplos		
PSUBB MM3, BOTON		
PSUBW MM7, SONIDO		
PSUBD MM3, MANTEQUILLA		
PSUBS	Resta con saturación con signo	Byte, palabra y doble palabra
0000 1111 1110 10gg	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
PSUBSB MM1, MM2		
PSUBSW MM7, MM3		
PSUBSD MM3, MM4		
0000 1111 1110 10gg	oo xxx mmm	mem→xreg
Ejemplos		
PSUBSB MM3, BOTON		
PSUBSW MM7, SONIDO		
PSUBSD MM3, MANTEQUILLA		
PSUBUS	Resta con saturación sin signo	Byte, palabra y doble palabra
0000 1111 1101 10gg	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
PSUBUSB MM1, MM2		
PSUBUSW MM7, MM3		
PSUBUSD MM3, MM4		
0000 1111 1101 10gg	oo xxx mmm	mem→xreg
Ejemplos		
PSUBUSB MM3, BOTON		
PSUBUSW MM7, SONIDO		
PSUBUSD MM3, MANTEQUILLA		

PUNPCKH	Desempaquea datos parte superior a siguiente más grande	Byte, palabra, doble palabra
0000 1111 0110 10gg	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
PUNPCKH MM1, MM2 PUNPCKH MM3, MM4		
0000 1111 0110 00gg	oo xxx mmm	mem→xreg
Ejemplos		
PUNPCKH MM7, AGUA PUNPCKH MM2, PERRITO		
PUNPCKL	Desempaquea datos parte inferior a siguiente más grande	Byte, palabra, doble palabra
0000 1111 0110 00gg	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
PUNPCKL MM1, MM2 PUNPCKL MM3, MM4		
0000 1111 0110 00gg	oo xxx mmm	mem→xreg
Ejemplos		
PUNPCKL MM7, AGUA PUNPCKL MM2, PERRITO		
PXOR	OR exclusivo a nivel de bits	Byte, palabra, doble palabra
0000 1111 1110 1111	11 xxx1 xxx2	xreg2→xreg1
Ejemplos		
PXOR MM2, MM3 PXOR MM4, MM7 PXOR MM0, MM1		
0000 1111 1110 1111	oo xxx mmm	mem→xreg
Ejemplos		
PXOR MM2, RANAS PXOR MM4, WALTER		

EJEMPLO 14-13(B)

```
;Procedimiento que suma BLOQUEA a BLOQUEB y almacena las sumas en BLOQUEC
BLOQUEA DB      1000 DUP(?)
BLOQUEB DB      1000 DUP(?)
BLOQUEC DB      1000 DUP(?)  

SUMAM    PROC    NEAR
        MOV     ECX,125
        .REPEAT
                MOVEQ MM0, QWORD PTR BLOQUEA[ECX-8]
                PADDB MM0, QWORD PTR BLOQUEB[ECX-8]
                MOVEQ QWORD PTR BLOQUEC[ECX-8],MM0
        .UNTILCXZ
        RET  

SUMAM    ENDP
```

Si realizamos una comparación exhaustiva de los programas, veremos que la versión MMX ejecuta el ciclo de tres instrucciones 125 veces, mientras que el software tradicional pasa a través de su ciclo 1000 veces. La versión MMX se ejecutará ocho veces más rápido. Esto ocurre debido a que se suman ocho bytes (QWORD) a la vez.

14-6**INTRODUCCIÓN A LA TECNOLOGÍA SSE**

El tipo más reciente de instrucción que se ha agregado al conjunto de instrucciones del Pentium 4 es SIMD (**una instrucción, múltiples datos**). Como el nombre lo implica, una sola instrucción opera sobre múltiples datos en forma muy parecida a las instrucciones MMX, las cuales son instrucciones SIMD que operan sobre múltiples datos. El conjunto de instrucciones MMX funciona con enteros, en tanto que el conjunto de instrucciones SIMD lo hace con números de punto flotante y también con enteros. Las instrucciones de la extensión SIMD aparecieron por primera vez en el Pentium III como instrucciones SSE (**extensiones SMD de flujo**). Después, se agregaron las instrucciones SSE 2 al Pentium 4; las instrucciones SSE 3 son la más reciente adición al Pentium 4 (el modelo E de 90 nanómetros).

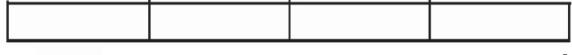
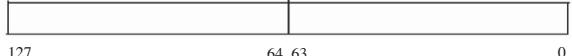
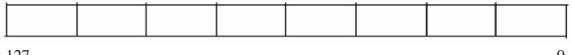
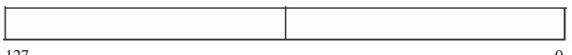
Recuerde que las instrucciones MMX comparten los registros con el coprocesador aritmético. Las instrucciones SSE utilizan un arreglo de registros nuevo y separado para operar sobre los datos. La figura 14-13 muestra un arreglo de ocho registros de 128 bits que funcionan con las instrucciones SSE. Estos nuevos registros se llaman **registros XMM** (XMM₀-XMM₇), lo cual denota registros multimedia extendidos. Para alojar este nuevo tamaño de datos de 128 bits se agregó una nueva palabra clave llamada **OWORD**. Una OWORD (**palabra octal**) designa una variable de 128 bits, como en OWORD PTR para el conjunto de instrucciones SSE. También se utiliza una doble palabra cuádruple a veces para especificar un número de 128 bits.

Al igual que los registros MMX, los registros XMM de la unidad SSE también es posible que contengan múltiples tipos de datos. La figura 14-14 muestra los tipos de datos que pueden aparecer en

FIGURA 14-13 Los registros XMM utilizados por las instrucciones SSE.

127		0
	XMM7	
	XMM6	
	XMM5	
	XMM4	
	XMM3	
	XMM2	
	XMM1	
	XMM0	

FIGURA 14-14 Formatos de datos para las instrucciones SSE 2 y SSE 3.

	4 números de punto flotante de tamaño individual
	2 números de punto flotante de tamaño doble
	16 enteros tipo byte
	8 enteros tipo palabra
	4 enteros tipo doble palabra
	2 enteros tipo palabra cuádruple

cualquier registro XMM para diversas instrucciones SSE. Un registro XMM llega a almacenar cuatro números de punto flotante de precisión simple o dos números de punto flotante de doble precisión. Los registros XMM también almacenan diecisés enteros de 8 bits, ocho enteros de 16 bits, cuatro enteros de 32 bits o dos enteros de 64 bits. Esto representa un incremento del doble de la capacidad del sistema, si se le compara con los enteros contenidos en los registros MMX, y por consecuencia de las velocidades de ejecución de las operaciones tipo entero que utilizan los registros XMM e instrucciones SSE. Para nuevas aplicaciones destinadas a ejecutarse en un microprocesador Pentium 4 o más reciente, se utilizan las instrucciones SSE en lugar de las instrucciones MMX. Como no todos los equipos son clase Pentium 4 todavía, existe la necesidad de incluir instrucciones con tecnología MMX en un programa, para que sean compatibles con tales sistemas anteriores.

Datos de punto flotante

Los datos de punto flotante se manejan como empaquetados o escalares, y de precisión simple o doble. La operación empaquetada se lleva a cabo en todas las secciones a la vez; la forma escalar sólo se opera en la sección de más a la derecha del contenido de los registros. La figura 14-15 muestra las operaciones empaquetadas y escalar sobre datos SSE en registros XMM. La forma escalar es comparable con la operación que realiza el coprocesador aritmético. A los códigos de operación se les anexa PS (empaquetado individual), SS (escalar individual), PD (empaquetado doble) o SD (escalar doble) para formar la instrucción deseada. Por ejemplo, el código de operación para una multiplicación es MUL, pero el código de operación para una operación con un valor tipo doble empaquetado es MULPD y MULSD para una multiplicación de un doble escalar. Las multiplicaciones de precisión simple son MULPS y MULSS. En otras palabras, una vez que se comprende la extensión de dos letras y su significado, es bastante sencillo dominar las nuevas instrucciones SSE.

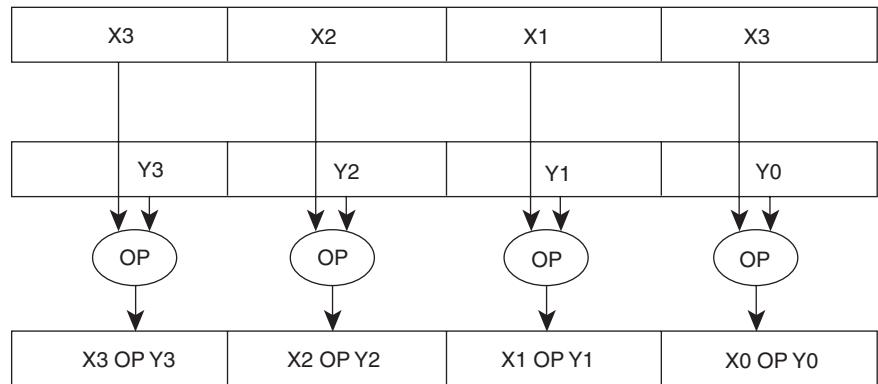
El conjunto de instrucciones

Las instrucciones SSE agregan unos cuantos tipos nuevos al conjunto de instrucciones. La unidad de punto flotante no tiene una instrucción recíproca, la cual se utiliza muy a menudo para resolver ecuaciones complejas. La instrucción recíproca ($\frac{1}{n}$) ahora aparece en las extensiones SSE como la instrucción RCP, la cual genera recíprocos y se escribe como RCPPS, RCPSS, RCPPD y RCPSD. También existe el recíproco de la instrucción de raíz cuadrada ($\frac{1}{\sqrt{n}}$), llamada RSQRT, que se escribe como RSQRTPS, RSQRTSS, RSQRTPD y RSQRTSD.

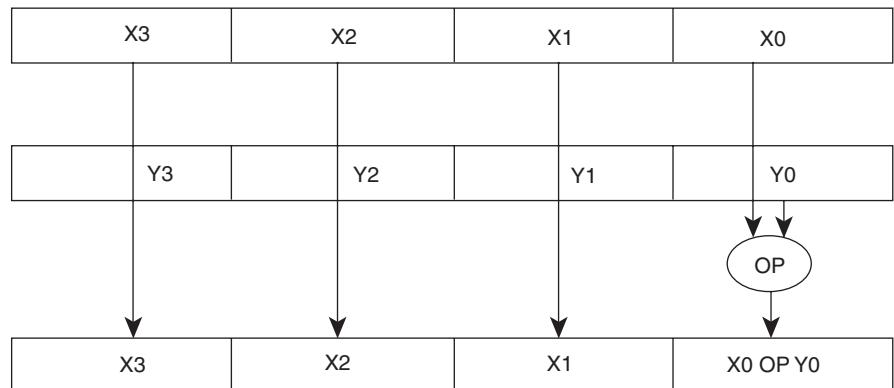
El resto de las instrucciones para la unidad SSE son básicamente las mismas que para el microprocesador y la unidad MMX, excepto en unos cuantos casos. La tabla de instrucciones en el apéndice B lista tales instrucciones, pero no las extensiones (PS, SS, PD y DS) para las mismas. Hay que tener de

FIGURA 14-15

Operaciones empaquetadas a) y escalares b) para números de punto flotante con precisión simple.



(a)



(b)

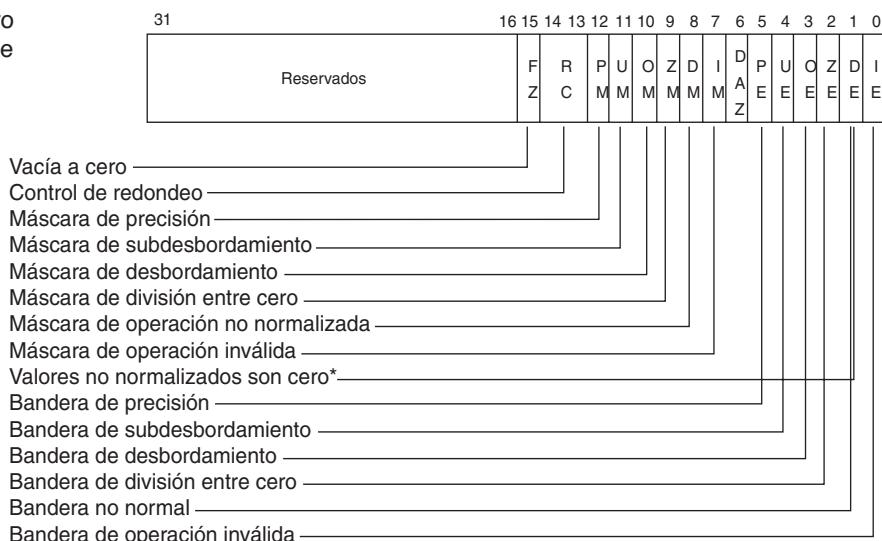
nuevo en cuenta que SSE 2 y SSE 3 contienen operaciones de doble precisión, mientras que SSE no. Las instrucciones que empiezan con la letra P operan sobre datos enteros tipo byte, palabra, doble palabra o palabra cuádruple. Por ejemplo, la instrucción PADDB XMM₀, XMM₁ suma los enteros de 16 bytes en el registro XMM₁ con los enteros de 16 bytes en el registro XMM₀. PADDW suma enteros de 16 bits, PADDD suma dobles palabras y PADDQ suma palabras cuádruples. Como Intel no proporciona los tiempos de ejecución para estas instrucciones, no aparecen en el apéndice.

El registro de control/estado

La unidad SSE también contiene un registro de control/estado, al cual se accede como MXCSR. La figura 14-16 muestra el MXCSR para la unidad SSE. Observe que este registro es muy similar al registro de control/estado del coprocesador aritmético que presentamos en una sección anterior de este capítulo. Este registro establece los modos de precisión y redondeo para el coprocesador como lo hace el registro de control para el coprocesador aritmético; además, proporciona información sobre la operación de la unidad SSE.

El registro SSE de control/estado se carga desde memoria mediante el uso de las instrucciones LDMXCSR y FXRSTOR, o se almacena en la memoria usando las instrucciones STMXCSR y FXSAVE. Suponga que el control de redondeo (en la figura 14-6 consulte el estado de los bits de control

FIGURA 14-16 El registro MXCSR (control/estado) de la unidad SSE.



* La bandera valores no normalizados son cero se introdujo en los procesadores Pentium 4 e Intel Xeon.

de redondeo) necesita modificarse para redondear hacia infinito positivo ($RC = 10$). El ejemplo 14-4 muestra el software que modifica sólo los bits de control de redondeo del registro de control/estado.

EJEMPLO 14-4

;modifica el control de redondeo a 10.

```
STMXCSR      CONTROL    ;almacena el registro de control/estado
BTS          CONTROL,14  ;activa el bit 14
BTR          CONTROL,13  ;borra el bit 13
LDMXCSR      CONTROL    ;vuelve a cargar el registro de control/estado
```

Ejemplos de programación

Se necesitan unos cuantos ejemplos de programación para mostrar cómo utilizar la unidad SSE. Como ya se mencionó, la unidad SSE permite operaciones de punto flotante y de enteros sobre múltiples datos. Suponga que la reactancia capacitiva es necesaria para un circuito que contiene un capacitor de $1.0 \mu\text{F}$ a varias frecuencias, desde 100 Hz hasta 10,000 Hz en intervalos de 100 Hz. La ecuación que se utiliza para calcular la reactancia capacitiva es:

$$XC = \frac{1}{2\pi FC}$$

El ejemplo 14-15 muestra un procedimiento que genera los 100 resultados para esta ecuación mediante el uso de la unidad SSE y datos de punto flotante con precisión simple. El programa que se lista en el ejemplo 14-15(a) utiliza la unidad SSE para realizar cuatro cálculos por iteración, mientras que el programa en el ejemplo 14-15(b) utiliza el coprocesador de punto flotante para calcular XC un valor a la vez. El ejemplo 14-15(c) es otro ejemplo en C++. Examine el programa para ver si el primer ejemplo pasa a través del ciclo 25 veces y si el segundo pasa a través del ciclo 100 veces. Cada vez que se ejecuta el ciclo en el ejemplo 14-15(a) se ejecutan siete instrucciones ($25 \times 7 = 175$), las cuales ocupan 175 tiempos (ciclos) de instrucción. El ejemplo 14-15(b) ejecuta ocho instrucciones por cada iteración de su ciclo ($100 \times 8 = 800$), las cuales requieren 800 tiempos (ciclos) de instrucción. Mediante el uso de este paralelismo, la unidad SSE permite realizar los cálculos en mucho menos tiempo que cualquier otro método. La versión en C++ del ejemplo 14-15(c) utiliza la directiva `_declspec(align(16))` antes de cada variable para asegurar que se alineen de manera correcta en la memoria. Si no se incluyen

estas directivas, el programa no funcionará debido a que las variables de memoria SSE deben alinearse en límites de palabras cuádruples por lo menos (16). La velocidad aproximada de ejecución de esta versión final es de 4 ½ veces más alta que el ejemplo 14-15(b).

EJEMPLO 14-15(A)

```
;uso de la unidad SSE

XC      DD      100 DUP(?)
CAP     DD      1.0E-6, 1.0E-6, 1.0E-6
F       DD      100.0, 200.0, 300.0, 400.0
INCR    DD      400.0, 400.0, 400.0, 400.0
PI      DD      4 DUP(?)  
  

FXC     PROC    NEAR  
  

        MOV     ECX, 0
        FLDPI
        FADD   ST, ST(0)           ;obtiene π
        FST    PI
        FST    PI+4
        FST    PI+8
        FSTP   PI+12
        MOVUPS XMM0, OWORD PTR PI  ;obtiene cuatro 2πs
        MOVUPS XMM1, OWORD PTR INCR ;obtiene incremento
        .REPEAT
            MOVUPS   XMM2, OWORD PTR F  ;carga frecuencias
            MULPS    XMM2, XMM0
            MULPS    XMM2, CAPS
            RCPPS    XMM3, XMM2
            MOVUPS   OWORD PTR XC[ECX], XMM3 ;encuentra el recíproco
            ADD      ECX, 16
            ADDPS   OWORD PTR F, XMM1  ;almacena cuatro XCs
        .UNTIL ECX == 100
        RET  
  

FXC     ENDP
```

EJEMPLO 14-15(B)

```
;uso del coprocesador

XC      DD      100 DUP(?)
CAP     DD      1.0E-6
F       DD      0
INCR    DD      100.0  
  

FXC1    PROC    NEAR  
  

        FLDPI
        FADD   ST, ST(0)           ;obtiene π
        FMUL   ST, CAP
        MOV    ECX, 0
        .REPEAT
            FLD    F
            FADD   INCX
            FST    F
            FMUL   ST, ST(1)
            FLD1
            FDIVR
            FSTP   XC[ECX*4]         ;almacena XC
            INC    ECX
        .UNTIL ECX == 100
        FCOMP
        RET  
  

FXC1    ENDP
```

EJEMPLO 14-15(C)

```

void EncuentraXC()
{
    //ejemplo de punto flotante en el que se usa C++ con el ensamblador
    //en linea

    _declspec(align(16)) float f[4] = {-300, -200, -100.0};
    _declspec(align(16)) float pi[4];
    _declspec(align(16)) float caps[4] = {1.0E-6, 1.0E-6, 1.0E-6, 1.0E-6};
    _declspec(align(16)) float incr[4] = {400, 400, 400, 400};
    _declspec(align(16)) float Xc[100];
    _asm
    {
        Fldpi                                ; forma 2π
        fadd      st,st(0)
        fst       pi
        fst       pi+4
        fst       pi+8
        fstp     pi+12
        movaps   xmm0,word ptr pi
        movaps   xmm1,word ptr incr
        movaps   xmm3,word ptr f
        mulps    xmm0,word ptr caps           ; 2πC
        mov      ecx,0

        CICLO1:
        movaps   xmm2,xmm3
        addps   xmm2,xmm1
        movaps   xmm3,xmm2
        mulps    xmm2,xmm0
        rcpss   xmm2,xmm2                   ; reciproco
        movaps   word otr Xc[ecx],xmm2
        add      ecx,16
        cmp      ecx,400
        jnz     CICLO1
    }
}

```

El ejemplo 14-15 (el primero en esta sección) utiliza números de punto flotante para realizar múltiples cálculos, pero la unidad SSE también puede operar en enteros. El ejemplo 14-16 utiliza la operación con enteros para sumar el BloqueA con el BloqueB y almacenar la suma en el BloqueC. Cada bloque contiene 4000 números de ocho bits. El ejemplo 14-16(a) lista un procedimiento en lenguaje ensamblador que forma las sumas con el uso de la unidad tipo entero estándar del microprocesador, la cual requiere 4000 iteraciones para realizarse.

EJEMPLO 14-16(A)

```

;Un procedimiento que forma 4000 sumas de ocho bits

SUMAS    PROC    NEAR

    MOV     ECX,0
    .REPEAT
        MOV     AL,BLOQUEA[ECX]
        ADD     AL,BLOQUEB[ECX]
        MOV     BLOQUEC,[ECX]
        INC     ECX
    .UNTIL ECX == 4000
    RET

SUMAS    ENDP

```

EJEMPLO 14-16(B)

```

;Un procedimiento que utiliza SSE para formar 4000 sumas de ocho bits

SUMAS1   PROC    NEAR

```

```

MOV     ECX, 0
.REPEAT
    MOVDQA XMM0, OWORD PTR BLOQUEA[ECX]
    PADDB  XMM0, OWORD PTR BLOQUEB[ECX]
    MOVDQA OWORD PTR BLOQUEC[ECX]
    ADD    ECX, 16
.UNTIL ECX == 4000
RET

SUMAS1 ENDP

```

Ambos programas del ejemplo generan 4000 sumas, aunque el segundo ejemplo que utiliza la unidad SSE pasa a través de su ciclo 250 veces, mientras que el primer ejemplo requiere 4000 pasadas. Por ende, el segundo ejemplo funciona 16 veces más rápido debido a la unidad SSE. Observe cómo se utiliza la instrucción PADDB (una instrucción que se presentó con la unidad MMX) con la unidad SSE. Esta unidad utiliza los mismos comandos que la unidad MMX, sólo que los registros son distintos. La unidad MMX utiliza registros MM de 64 bits y la unidad SSE, registros XMM de 128 bits.

Optimización

El compilador en Visual C++ cuenta con optimización para la unidad SSE, pero no optimiza los ejemplos que se presentaron en este capítulo. Tratará de optimizar una ecuación individual en una instrucción si logra utilizarse la unidad SSE para la ecuación. No analiza un programa para ver los bloques de operaciones que pueden optimizarse, como en los ejemplos que presentamos aquí. Hasta que se desarrolle un compilador y las extensiones para incluir operaciones en paralelo, tales como éstas, los programas que requieren de alta velocidad necesitarán un lenguaje ensamblador codificado a mano para la optimización. Esto se aplica en especial a la unidad SSE.

14-7

RESUMEN

1. El coprocesador aritmético funciona en paralelo con el microprocesador. Esto significa que el microprocesador y el coprocesador pueden ejecutar sus instrucciones respectivas en forma simultánea.
2. Los tipos de datos que manipula el coprocesador son: entero con signo, punto flotante y decimal codificado en binario (BCD).
3. Hay tres formas de enteros que se utilizan con el coprocesador: palabra (16 bits), corto (32 bits) y largo (64 bits). Cada entero contiene un número con signo en magnitud verdadera para los números positivos y en formato de complemento a dos para los números negativos.
4. Un número BCD se almacena como un número de 18 dígitos en 10 bytes de memoria. El byte más significativo contiene el bit de signo, en tanto que los nueve bytes restantes contienen un número BCD empaquetado de 18 dígitos.
5. El coprocesador soporta tres tipos de números de punto flotante: de precisión simple (32 bits), de doble precisión (64 bits) y de precisión extendida temporal (80 bits). Un número de punto flotante tiene tres partes: el signo, el exponente con desviación y la mantisa. En el coprocesador, el exponente se desvía con una constante, mientras que el bit entero del número normalizado no se almacena en la mantisa, excepto en la forma con precisión extendida temporal.
6. Los números decimales se convierten en números de punto flotante mediante el siguiente procedimiento: *a)* el número se convierte en binario, *b)* el número binario se normaliza, *c)* se suma la desviación al exponente y *d)* el número se almacena en formato de punto flotante.
7. Los números de punto flotante se convierten en decimales mediante el siguiente procedimiento: *a)* se resta la desviación del exponente, *b)* se denormaliza el número y *c)* se convierte en decimal.

8. El 80287 utiliza el espacio de E/S para la ejecución de algunas de sus instrucciones. Este espacio es invisible para el programa; el sistema 80286/80287 lo utiliza en forma interna. Estas direcciones de E/S de 16 bits (00F8H-00FFH) no deben utilizarse para transferencias de datos de E/S en un sistema que contenga un 80287. Los coprocesadores 80387, 80486/7 y del Pentium al Pentium 4 utilizan las direcciones de E/S 800000H8H-800000FFH.
9. El coprocesador contiene un registro de estado que indica si está ocupado, las condiciones que siguen después de una comparación o prueba, la posición de la parte superior de la pila y el estado de los bits de error. La instrucción FSTSW AX, seguida de SAHF, se utiliza con frecuencia con las instrucciones de salto condicional para evaluar ciertas condiciones del coprocesador.
10. El registro de control del coprocesador contiene bits de control que seleccionan infinito, redondeo, precisión y máscaras de error.
11. Las siguientes directivas se utilizan a menudo con el coprocesador para almacenar datos: DW (define palabra), DD (define doble palabra), DQ (define palabra cuádruple) y DT (define 10 bytes).
12. El coprocesador utiliza una pila para transferir datos entre sí mismo y el sistema de memoria. En general, los datos se cargan en la parte superior de la pila o se extraen de la misma para su almacenamiento.
13. Todos los datos internos del coprocesador están siempre en el formato de precisión extendida de 80 bits. La única vez que los datos están en cualquier otra forma es cuando se almacenan o se cargan hacia/desde la memoria.
14. Los modos de direccionamiento del coprocesador incluyen el modo de pila clásico, de registro, de registro con una extracción y de memoria. El direccionamiento de pila está implícito. Los datos en ST se convierten en el origen y los de ST(1) en el destino, mientras que el resultado se coloca en ST después de una extracción.
15. Las operaciones aritméticas del coprocesador son: suma, resta, multiplicación, división y cálculo de la raíz cuadrada.
16. En el conjunto de instrucciones del coprocesador hay funciones trascendentales. Estas funciones buscan la tangente parcial o arcotangente, $2X - 1$, $Y \log_2 X$ y $Y \log_2 (X+1)$. Los coprocesadores 80387, 80486/7 y del Pentium al Pentium 4 también incluyen las funciones seno y coseno.
17. Hay constantes que se almacenan dentro del coprocesador para proporcionar +0.0, +1.0, π , $\log_2 10$, $\log_2 e$, $\log_2 2$ y $\log_e 2$.
18. El 80387 funciona con el microprocesador 80386 y el 80487SX lo hace con el microprocesador 80486SX, pero los microprocesadores 80486DX y del Pentium al Pentium 4 contienen su propio coprocesador aritmético interno. En estos coprocesadores están disponibles las instrucciones que ejecutaban las primeras versiones. Además de estas instrucciones, los coprocesadores 80387, 80486/7 y del Pentium al Pentium 4 también sirven para encontrar el seno y el coseno.
19. Los microprocesadores del Pentium Pro al Pentium 4 contienen dos nuevas instrucciones de punto flotante: FCMOV y FCOMI. La instrucción FCMOV es un movimiento condicional y la instrucción FCOMI realiza la misma tarea que FCOM, pero además coloca las banderas de punto flotante en el registro de banderas del sistema.
20. La extensión MMX utiliza los registros del coprocesador aritmético para MM₀-MM₇. Por lo tanto, es importante que el software del coprocesador y el software MMX no traten de usarlos al mismo tiempo.
21. Las instrucciones para la extensión MMX realizan operaciones aritméticas y lógicas en bytes (ocho a la vez), palabras (cuatro a la vez), dobles palabras (dos a la vez) y palabras cuádruples. Las operaciones que se realizan son: suma, resta, multiplicación, división, AND, OR, OR exclusivo y NAND.
22. Tanto la unidad MMX como la unidad SSE emplean técnicas SIMD para realizar operaciones en paralelo sobre múltiples datos con una sola instrucción. La unidad SSE efectúa operaciones sobre enteros y números de punto flotante. Los registros en la unidad SSE son de 128 bits y logran almacenar (SSE 2 o más reciente) 16 bytes a la vez o cuatro números de punto flotante de precisión simple. La unidad SSE contiene los registros XMM₀-XMM₇.
23. Las nuevas aplicaciones escritas para el Pentium 4 deberían contener instrucciones SSE en lugar de instrucciones MMX.
24. Se ha agregado el apuntador OWORD para direccionar números de 128 bits, los cuales se conocen como palabras octales o dobles palabras cuádruples.

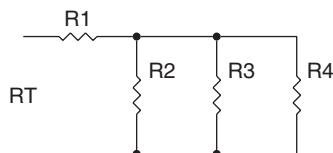
14-8**PREGUNTAS Y PROBLEMAS**

1. Liste los tres tipos de datos que se cargan o se almacenan en memoria mediante el coprocesador.
2. Liste los tres tipos de datos enteros, el rango de los enteros que puede almacenarse en ellos y el número de bits que se asigna a cada uno.
3. Explique cómo el coprocesador almacena un número BCD en memoria.
4. Liste los tres tipos de números de punto flotante que se utilizan con el coprocesador y el número de bits binarios que se asignan a cada uno.
5. Convierta los siguientes números decimales en números de punto flotante con precisión simple:
 - a) 28.75
 - b) 624
 - c) -0.615
 - d) +0.0
 - e) -1000.5
6. Convierta los siguientes números de punto flotante con precisión simple en decimales
 - a) 11000000 11110000 00000000 00000000
 - b) 00111111 00010000 00000000 00000000
 - c) 01000011 10011001 00000000 00000000
 - d) 01000000 00000000 00000000 00000000
 - e) 01000001 00100000 00000000 00000000
 - f) 00000000 00000000 00000000 00000000
7. Explique qué hace el coprocesador cuando se ejecuta una instrucción normal del microprocesador.
8. Explique qué hace el microprocesador cuando se ejecuta una instrucción del coprocesador.
9. ¿Cuál es el propósito de los bits C_3-C_0 en el registro de estado?
10. ¿Qué operación se logra mediante la instrucción FSTSW AX?
11. ¿Cuál es el propósito del bit IE en el registro de estado?
12. ¿Cómo pueden utilizarse las instrucciones SAHF y de salto condicional para determinar si la parte superior de la pila (ST) es igual al registro ST(2)?
13. ¿Cómo se selecciona el modo de redondeo en el 80X87?
14. ¿Qué instrucción del coprocesador utiliza el registro AX del microprocesador?
15. ¿Qué puertos de E/S se reservan para el uso del coprocesador en el 80287?
16. ¿Cómo se almacenan los datos dentro del coprocesador?
17. ¿Qué es un NAN?
18. Cada vez que se restablece el coprocesador, la parte superior del registro de pila es el registro número _____.
19. ¿Qué significa el término *cortar* en cuanto al redondeo de los bits de control del registro de control?
20. ¿Cuál es la diferencia entre afín y control de infinito proyectivo?
21. ¿Qué instrucción del microprocesador forma los códigos de operación para el coprocesador?
22. La instrucción FINIT selecciona la precisión _____ para todas las operaciones del coprocesador.
23. Mediante el uso de seudocódigos de operación del ensamblador, forme instrucciones que realicen lo siguiente
 - a) Almacenar 23.44 en la posición de memoria RANA de punto flotante, con doble precisión.
 - b) Almacenar -123 en la posición DATOS₃ tipo entero con signo de 32 bits.
 - c) Almacenar -23.8 en la posición de memoria DATOSL de punto flotante con precisión simple.
 - d) Reservar la posición de memoria DATOS₂ de doble precisión.
24. Describa cómo funciona la instrucción FST DATOS. Suponga que DATOS se define como una posición de memoria de 64 bits.
25. ¿Qué realiza la instrucción FILD DATOS?
26. Forme una instrucción que agregue el contenido del registro 3 a la parte superior de la pila.
27. Describa la operación de la instrucción FADD.
28. Seleccione una instrucción que reste el contenido del registro 2 de la parte superior de la pila y almacene el resultado en el registro 2.

29. ¿Cuál es la función de la instrucción FBSTP DATOS?
30. ¿Cuál es la diferencia entre una división directa y una inversa?
31. ¿Cuál es el propósito de la instrucción FCOMI del Pentium Pro?
32. ¿Qué es lo que realiza la instrucción FCMOV B del Pentium Pro?
33. ¿Qué debe ocurrir antes de ejecutar cualquier instrucción FCMOV?
34. Desarrolle un procedimiento que encuentre el recíproco del número de punto flotante de precisión simple. Este número deberá pasarse al procedimiento en EAX y devolverse como recíproco en EAX.
35. ¿Cuál es la diferencia entre las instrucciones FTST y FXAM?
36. Explique qué es lo que calcula la instrucción F2XM1.
37. ¿Cuál bit del registro de estado del coprocesador debe evaluarse después de que se ejecuta la instrucción FQSRT?
38. ¿Cuál instrucción del coprocesador mete el valor π en la parte superior de la pila?
39. ¿Cuál instrucción del coprocesador coloca 1.0 en la parte superior de la pila?
40. ¿Qué realizará la instrucción FFREE ST(2) al ejecutarse?
41. ¿Cuál instrucción almacena el entorno?
42. ¿Qué es lo que la instrucción FSAVE guarda?
43. Desarrolle un procedimiento que encuentre el área de un rectángulo ($A = L \times W$). Las posiciones de memoria para este procedimiento son las posiciones A, L y W de punto flotante con precisión simple.
44. Escriba un procedimiento que encuentre la reactancia capacitiva ($XC = \frac{1}{2\pi FC}$). Las posiciones de memoria para este procedimiento son las posiciones de punto flotante con precisión simple XC, F y C1 para C.
45. Desarrolle un procedimiento que genere una tabla de raíces cuadradas para los enteros del 2 al 10. Los resultados deberán almacenarse como números de punto flotante con precisión simple en un arreglo llamado RAÍCES.
46. ¿Cuándo se utiliza la instrucción FWAIT en un programa?
47. ¿Cuál es la diferencia entre las instrucciones FSTSW y FNSTSW?
48. Dado el circuito en serie/paralelo y la ecuación que se muestran en la figura 14-17, desarrolle un programa mediante el uso de valores de precisión simple para R₁, R₂, R₃ y R₄ que encuentre la resistencia total y almacene el resultado en la posición RT de precisión simple.
49. Desarrolle un procedimiento que encuentre el coseno de un número de punto flotante con precisión simple. El ángulo (en grados) deberá pasarse al procedimiento en EAX, mientras que el coseno se devolverá en EAX. Recuerde que FCOS encuentra el coseno de un ángulo expresado en radianes.
50. Dados dos arreglos de datos de punto flotante con doble precisión (ARREGLO₁ y ARREGLO₂), en los que cada uno contiene 100 elementos, desarrolle un procedimiento que encuentre el producto de ARREGLO₁ por ARREGLO₂, y después almacene el resultado de punto flotante con doble precisión en un tercer arreglo (ARREGLO₃).
51. Desarrolle un procedimiento que tome el contenido de precisión simple del registro EBX por π y que almacene el resultado en el registro EBX como un número de punto flotante con precisión simple. Debe utilizar la memoria para realizar esta tarea.

FIGURA 14-17

El circuito en serie/paralelo (pregunta 48).



$$RT = R_1 + \frac{1}{\frac{1}{R_2} + \frac{1}{R_3} + \frac{1}{R_4}}$$

52. Escriba un procedimiento que eleve un número X (de punto flotante con precisión simple) a la potencia Y. Los parámetros se pasarán al procedimiento mediante $EAX = X$ y $EBX = Y$. El resultado se pasará de vuelta en ECX a la secuencia que hizo la llamada.
53. Dado que el $\text{LOG}_{10} X = (\text{LOG}_2 10)^{-1} \times \text{LOG}_2 X$, escriba un procedimiento llamado LOG_{10} que encuentre el LOG_{10} del valor (X) en la parte superior de la pila. Devuelva el LOG_{10} en la parte superior de la pila al final del procedimiento.
54. Use el procedimiento que desarrolló en la pregunta 53 para resolver la ecuación

$$\text{Ganancia en decibeles} = 20 \log_{10} \frac{V_{\text{salida}}}{V_{\text{entrada}}}$$

El programa deberá tomar arreglos de valores con precisión simple para V_{salida} y V_{entrada} , y almacenar las ganancias en decibeles en un tercer arreglo llamado GDB. Deben ser 100 valores para V_{salida} y V_{entrada} .

55. ¿Cuál es la extensión MMX para los microprocesadores del Pentium al Pentium 4?
56. ¿Cuál es el propósito de la instrucción EMMS?
57. ¿En dónde se encuentran los registros MM₀-MM₇ en el microprocesador?
58. ¿Qué es la saturación con signo?
59. ¿Qué es la saturación sin signo?
60. ¿Cómo podrían almacenarse todos los registros MMX en la memoria mediante una sola instrucción?
61. Escriba un programa corto que utilice una instrucción MMX para multiplicar los números tipo palabra en dos arreglos y almacene los resultados de 32 bits en un tercer arreglo. Los arreglos de origen son de 256 palabras.
62. ¿Qué son las instrucciones SIMD?
63. ¿Qué son las instrucciones SSE?
64. Los registros XMM son de _____ bits.
65. Un solo registro XMM puede almacenar _____ números de punto flotante con precisión simple.
66. Un solo registro XMM puede almacenar _____ enteros tipo byte.
67. ¿Qué es una OWORD?
68. ¿Pueden ejecutarse las instrucciones de punto flotante para el coprocesador aritmético al mismo tiempo que las instrucciones SSE?
69. Desarrolle una función en C++ (mediante el uso de código ensamblador en línea) que calcule (mediante el uso de instrucciones SSE escalares e instrucciones de punto flotante) y devuelva un número de precisión simple que represente la frecuencia resonante de los parámetros (L y C) que reciba para resolver la siguiente ecuación:

$$Fr = \frac{1}{2\pi\sqrt{LC}}$$

CAPÍTULO 15

Interfaz de bus

INTRODUCCIÓN

Muchas aplicaciones requieren el conocimiento de los sistemas de buses que se encuentran dentro de la computadora personal. Algunas veces, los tableros principales de estas máquinas se utilizan como sistemas básicos en aplicaciones industriales. En consecuencia, a menudo tales sistemas requieren interfaces comunes que se conectan a uno de los buses en el tablero principal. En este capítulo veremos los buses ISA (arquitectura estándar de la industria), PCI (interconexión de componentes periféricos), PCI Express, USB (bus serial universal) y AGP (puerto de gráficos acelerados). También proporcionaremos algunas interfaces simples para muchos de esos sistemas de bus, junto con guías de diseño.

Aunque es probable que no se incluyan en las computadoras personales del futuro, hablaremos sobre los puertos paralelo y de comunicaciones en serie. Éstos fueron los primeros puertos de E/S en la computadora personal y han sobrepasado la prueba del tiempo, pero el bus serial universal parece haberlos reemplazado casi por completo.

OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

1. Describir con detalle las conexiones de terminales y de señales del bus en los puertos paralelo y serial, así como en los buses ISA, AGP, PCI y PCI Express.
2. Desarrollar interfaces simples que se conecten a los puertos paralelo y serial, así como a los buses ISA y PCI.
3. Programar las interfaces ubicadas en tableros que se conectan a los buses ISA y PCI.
4. Describir la operación del USB y desarrollar algunos programas cortos para transferir datos.
5. Explicar cómo el AGP incrementa la eficiencia del subsistema de gráficos.

15-1

EL BUS ISA

El bus ISA, o **arquitectura estándar de la industria**, ha estado presente desde el comienzo del sistema de la computadora personal compatible con IBM (alrededor de 1982). De hecho, cualquier tarjeta de las primeras computadoras personales logra conectarse y funcionará en cualquiera de las computadoras modernas basadas en el Pentium 4, siempre y cuando tenga una ranura ISA. Lo anterior es posible gracias a la interfaz del bus ISA con la que cuentan algunos de estos equipos; tal interfaz sigue siendo compatible con las primeras computadoras personales. El bus ISA casi ha desaparecido en la PC del hogar, pero aún se encuentra en muchas aplicaciones industriales; por tal razón, hablaremos aquí sobre dicho bus.

Evolución del bus ISA

El bus ISA ha cambiado desde sus primeros días. Con el paso de los años ha evolucionado de su estándar original de 8 bits al estándar de 16 bits, que todavía traen algunos sistemas de la actualidad. Los últimos sistemas computacionales que contenían el bus ISA en masa fueron los basados en el Pentium III. Cuando el Pentium 4 empezó a aparecer, el bus ISA comenzó a desaparecer. Durante este proceso hubo incluso una versión de 32 bits conocida como bus EISA (**ISA extendido**), pero éste parece haber desaparecido por completo. Lo que permanece hoy en algunas computadoras personales es una ranura ISA (**conexión**) en el tablero principal, la cual puede aceptar ya sea una tarjeta ISA de 8 bits o una tarjeta de circuito impreso ISA de 16 bits. Las tarjetas de circuito impreso de 32 bits son de bus PCI o, en algunos equipos antiguos basados en el 80486, de bus VESA. El bus ISA casi ha desaparecido por completo en las computadoras del hogar, pero está disponible como pedido especial para la mayoría de los tableros principales. El bus ISA sigue presente en muchas aplicaciones industriales, aunque sus días ahora parecen estar limitados.

La interfaz de salida del bus ISA de 8 bits

La figura 15-1 muestra el conector ISA de 8 bits que se encuentra en el tablero principal de todos los sistemas de computadora personal (de nuevo, éste puede combinarse con un conector de 16 bits). El conector del bus ISA contiene todo el bus de direcciones demultiplexado ($A_{19}-A_0$) para el sistema 8088 de 1 Mbyte, el bus de datos de 8 bits (D_7-D_0) y las cuatro señales de control MEMR, MEMW, IOR e IOW para controlar la E/S y cualquier memoria que lograra colocarse en la tarjeta de circuito impreso. Es muy raro que se agregue memoria a cualquier tarjeta de bus ISA hoy, ya que la tarjeta ISA sólo opera a una velocidad de 8 MHz. En algunas tarjetas ISA podría utilizarse una EPROM o una Memoria Flash para configurar la información, pero nunca RAM.

FIGURA 15-1 El bus ISA de 8 bits.

Parte posterior de la computadora

Número de terminal

1	GND	I/O CHK
2	RESET	D7
3	+5V	D6
4	IRQ9	D5
5	-5V	D4
6	DRQ2	D3
7	-12V	D2
8	OWS	D1
9	+12V	D0
10	GND	IO RDY
11	MEMW	AEN
12	MEMR	A19
13	IOW	A18
14	IOR	A17
15	DACK3	A16
16	DRQ3	A15
17	DACK1	A14
18	DRQ1	A13
19	DACK0	A12
20	CLOCK	A11
21	IRQ7	A10
22	IRQ6	A9
23	IRQ5	A8
24	IRQ4	A7
25	IRQ3	A6
26	DACK2	A5
27	T/C	A4
28	ALE	A3
29	+5V	A2
30	OSC	A1
31	GND	A0

Lado de la soildadura

Lado de los componentes

Otras señales útiles para la interfaz de E/S son las **líneas de petición de interrupción IRQ₂–IRQ₇**. Hay que tomar en cuenta que IRQ₂ se redirecciona a IRQ₉ en los sistemas modernos, lo cual se indica en el conector de la figura 15-1. Las señales de control de los canales DMA 0–3 también están presentes en el conector. Las **entradas de petición de DMA** se identifican como DRQ₁–DRQ₃, en tanto que las **salidas de reconocimiento de DMA** se identifican como DACK₀–DACK₃. Observe que no está la terminal de entrada DRQ₀ debido a que las primeras computadoras la utilizaban junto con la salida DACK₀ como señal para refreshar cualquier DRAM que llegara a encontrarse en la tarjeta ISA. Hoy esta terminal de salida contiene una señal de reloj de 15.2 µs que se utilizaba para refreshar la DRAM. El resto de las terminales son para la energía y para RESET.

Suponga que debe conectarse una serie de cuatro enclavamientos de 8 bits con la computadora personal para obtener 32 bits de datos en paralelo. Para lograr esto, se compra una tarjeta de interfaz ISA (número de pieza 4713-1) de una compañía tal como Vector Electronics o de cualquiera otra. Además del conector de borde para el bus ISA, la tarjeta contiene espacio en su parte posterior para los conectores de la interfaz. Puede colocarse un conector tipo D subminiatura de 37 terminales en la parte posterior de la tarjeta para transferir los 32 bits de datos al origen externo.

La figura 15-2 muestra una interfaz simple para el bus ISA, la cual provee 32 bits de datos TTL en paralelo. Este sistema de ejemplo ilustra algunos puntos importantes sobre cualquier interfaz del sistema. En primer lugar, es en extremo importante que la carga para el bus ISA se mantenga en una carga TTL de baja energía (LS). En este circuito se utiliza un búfer 74LS244 para reducir la carga en el bus de

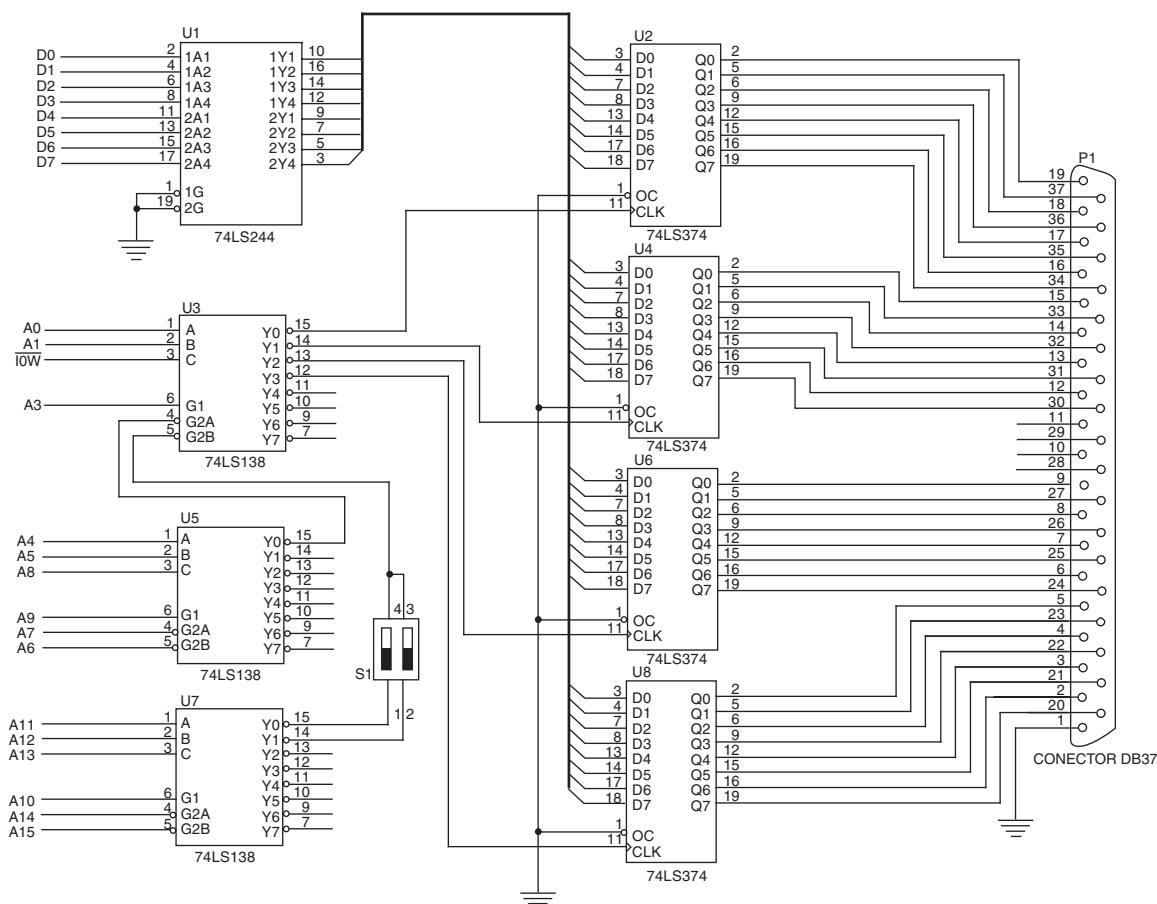


FIGURA 15-2 Un puerto paralelo de 32 bits conectado al bus ISA de 8 bits.

TABLA 15-1 Las asignaciones de puertos de E/S de la figura 15-2.

<i>Interruptor DIP</i>	<i>Enclavamiento U₂</i>	<i>Enclavamiento U₄</i>	<i>Enclavamiento U₆</i>	<i>Enclavamiento U₈</i>
1-4 Encendido	0608H o 060CH	0609H o 060DH	060AH o 060EH	060BH o 060FH
2-3 Encendido	0E08H o 0E0CH	0E09H o 0E0DH	0E0AH o 0E0EH	0E0BH o 0E0FH

datos. Si el 74LS244 no estuviera ahí, este sistema presentaría al bus de datos cuatro cargas unitarias. Si todas las tarjetas del bus presentaran cargas pesadas, el sistema no operaría en forma apropiada (o tal vez no funcionaría del todo).

En este circuito, la salida de la tarjeta ISA se proporciona mediante un conector de 37 terminales, identificado como P₁. Las terminales de salida del circuito se conectan a P₁ y se adjunta un cable de tierra. Debemos proporcionar tierra al mundo exterior; de lo contrario, los datos TTL en el puerto paralelo serían inútiles. Si es necesario, las terminales de control de salida (\overline{OC}) en cada uno de los chips de enclavamiento 74LS374 también pueden quitarse de tierra y conectarse a las cuatro terminales restantes en P₁. Esto permite que un circuito externo controle las salidas de los enclavamientos.

Un pequeño interruptor DIP se coloca en dos de las salidas de D₇, de manera que pueda modificarse la dirección si ocurre un conflicto de direcciones con otra tarjeta. Esto es improbable, a menos que planee utilizar dos de estas tarjetas en el mismo sistema. La conexión de dirección A₂ no se decodifica en este sistema, por lo que se convierte en “no importa” (x). En la tabla 15-1 verá las direcciones de cada enclavamiento y cada posición de S₁. Observe que sólo uno de los dos interruptores está encendido a la vez y que cada puerto tiene dos direcciones posibles para cada posición de interruptor, ya que A₂ no está conectada.

En la computadora personal, el bus ISA fue diseñado para operar en las direcciones de E/S 0000H a 03FFH. Dependiendo de la versión y del fabricante de la tarjeta principal, las tarjetas ISA pueden o no funcionar más allá de estas posiciones. Algunos sistemas más recientes a menudo permiten puertos ISA en las posiciones más allá de 03FFH, pero los sistemas antiguos no. Tal vez haya que modificar los puertos en este ejemplo para algunos sistemas. Algunas tarjetas antiguas sólo decodifican las direcciones de E/S 0000H-03FFH y logran llegar a enfrentar conflictos de direcciones si las direcciones de los puertos más allá de 03FFH entran en conflicto. Los puertos se decodifican en este ejemplo mediante tres decodificadores 74LS138. Sería más eficiente y efectivo en costo decodificar los puertos con un dispositivo lógico programable.

La figura 15-3 muestra el circuito de la figura 15-2 rediseñado mediante el uso de un PLD con la finalidad de decodificar las direcciones para el sistema. Observe que los bits de dirección A₁₅-A₄ se decodifican mediante el PLD y que el interruptor está conectado a dos de las entradas del PLD. Este cambio permite cuatro direcciones de puerto de E/S distintas para cada enclavamiento, lo cual hace que el circuito sea más flexible. La tabla 15-2 muestra el número de puerto seleccionado por el interruptor 1-4 y el interruptor 2-3. El ejemplo 15-1 muestra el programa para el PLD que realiza las asignaciones de puertos de la tabla 15-2.

EJEMPLO 15-1

```
-- Código en VHDL para el decodificador de la figura 15-3

library ieee;
use ieee.std_logic_1164.all;

entity DECODIFICADOR_15_3 is

port (
    IOW, A14, A13, A12, A11, A10, A9, A8, A7, A6
        A5, A4, A3, A2, A1, A0, S1, S2: in STD_LOGIC;
    U3, U4, U5, U6: out STD_LOGIC;
);

end;
```

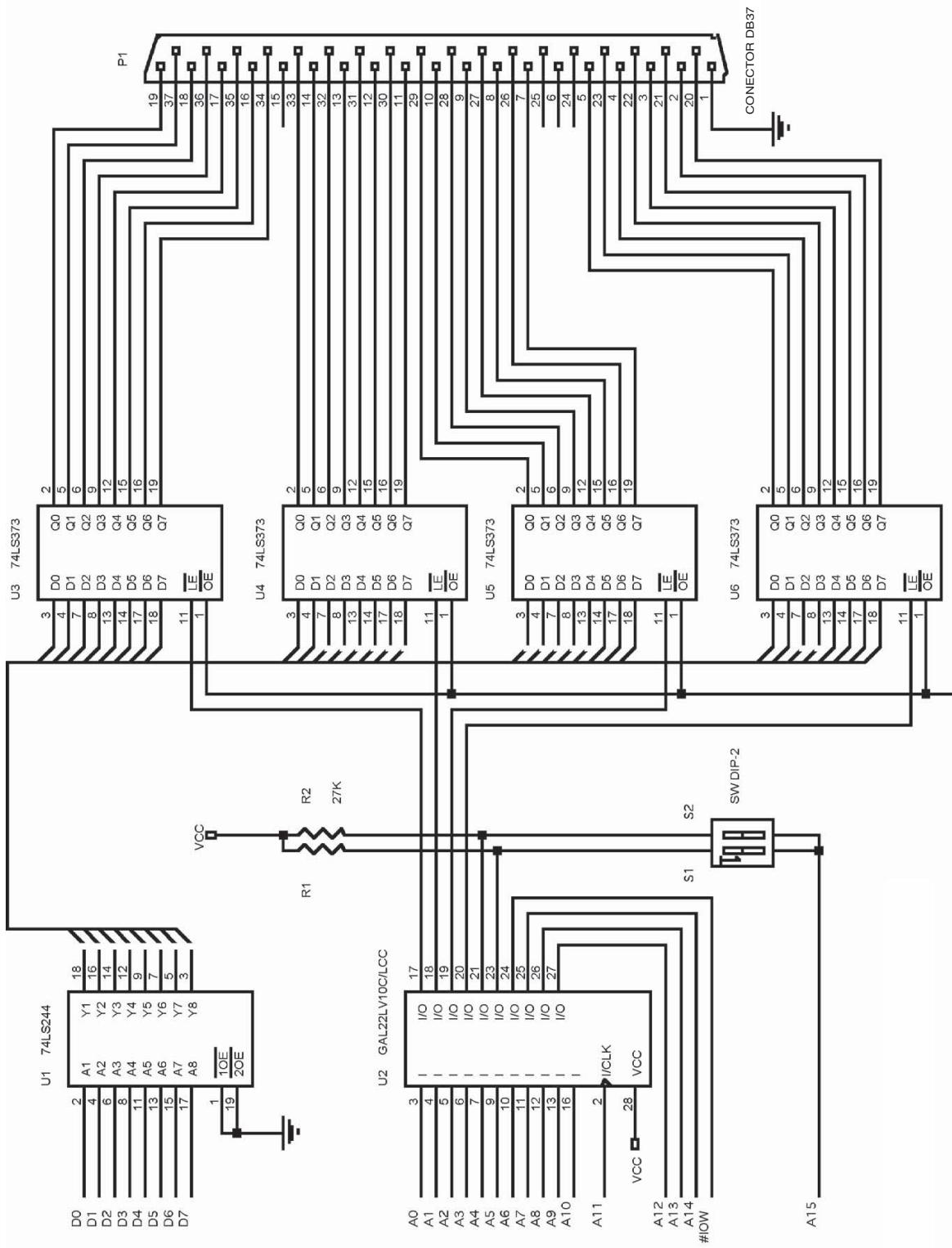


FIGURA 15-3 Una interfaz de 32 bits en paralelo para el bus ISA.

TABLA 15-2 Asignaciones de puertos de la figura 15-3.

S_2	S_1	U_3	U_4	U_5	U_6
Encendido	Encendido	0300H	0301H	0302H	0303H
Encendido	Apagado	0304H	0305H	0306H	0307H
Apagado	Encendido	0308H	0309H	030AH	030BH
Apagado	Apagado	030CH	030DH	030EH	030FH

Datos: Encendido es un interruptor cerrado (0) y apagado es abierto (1).

```

architecture V1 of DECODIFICADOR_15_3 is
begin
    U3 <= IOW or A14 or A13 or A12 or A11 or A10 or not A9 or not A8 or A7
          or A6 or A5 or A4 or A1 or A0 or (S2 or S1 or A3 or A2) and (S2 or
          not S1 or A3 or not A2) and (not S2 or S1 or not A3 or A2) and
          (not S2 or not S1 or not A3 or not A2);
    U4 <= IOW or A14 or A13 or A12 or A11 or A10 or not A9 or not A8 or A7
          or A6 or A5 or A4 or A1 or not A0 or (S2 or S1 or A3 or A2) and
          (S2 or not S1 or A3 or not A2) and (not S2 or S1 or not A3 or A2)
          and (not S2 or not S1 or not A3 or not A2);
    U5 <= IOW or A14 or A13 or A12 or A11 or A10 or not A9 or not A8 or A7
          or A6 or A5 or A4 or not A1 or A0 or (S2 or S1 or A3 or A2) and
          (S2 or not S1 or A3 or not A2) and (not S2 or S1 or not A3 or A2)
          and (not S2 or not S1 or not A3 or not A2);
    U6 <= IOW or A14 or A13 or A12 or A11 or A10 or not A9 or not A8 or A7
          or A6 or A5 or A4 or not A1 or not A0 or (S2 or S1 or A3 or A2)
          and (S2 or not S1 or A3 or not A2) and (not S2 or S1 or not A3 or
          A2) and (not S2 or not S1 or not A3 or not A2);
end V1;

```

Observe en el ejemplo 15-1 cómo el primer término (U_3) genera un 0 lógico en la salida al decodificador sólo cuando ambos interruptores están en sus posiciones de apagado para el puerto de E/S 0300H. También genera una señal de reloj para U_3 , para los puertos de E/S 304H, 308H o 30CH, dependiendo de la posición de los interruptores. El segundo término (U_4) está activo para los puertos 301H, 305H, 309H o 30DH, dependiendo de la posición de los interruptores. De nuevo, en la tabla 15-2 consulte el conjunto completo de asignaciones de puertos para diversas posiciones de los interruptores. Como A_{15} está conectada a la parte inferior de los interruptores, este circuito también activará los enclavamientos para otras posiciones de E/S, ya que no está decodificada. Las direcciones de E/S 830XH también generan señales de reloj para el enclavamiento, puesto que A_{15} no está decodificada.

El ejemplo 15-2 muestra dos funciones en C++, que transfieren un entero hacia el puerto de 32 bits. Cualquiera de estas funciones envía datos al puerto; la primera es más eficiente, pero la segunda puede ser más legible. (El ejemplo 15-2(c) muestra el ejemplo 15-2(b) en formato desensamblado.) Se pasan dos parámetros a la función: uno corresponde a los datos que se envían al puerto y el otro es la dirección del puerto base. La dirección base es 0300H, 0304H, 0308H o 030CH, y debe concordar con la posición de los interruptores de la figura 15-3.

EJEMPLO 15-2(A)

```

void PuertoSalida(int direccion, int datos)
{
    __asm
    {
        mov  edx,direccion
        mov  eax,datos
        mov  ecx, 4
    PuertoSalida:
        out  dx,al           ;salida de 8 bits
        shr  eax,8           ;obtiene siguiente sección de 8 bits
    }
}

```

```

        inc dx           ;direcciona siguiente puerto
        loop PuertoSalida1 ;se repite 4 veces
    }
}

```

EJEMPLO 15-2(B)

```

void PrtSal(int direccion, int datos)
{
    for ( int a = direccion; a < direccion + 4; a++ )
    {
        __asm
        {
            mov edx,a
            mov eax,datos
            out dx,al
        }
        datos >>= 8;           //obtiene siguiente sección de 8 bits
    }
}

```

EJEMPLO 15-2(C)

```

//Ejemplo 15-2(b) desensamblado

for ( int a = direccion; a < direccion + 4; a++ )
00413823  mov      eax,dword ptr [direccion]
00413826  mov      dword ptr [a],eax
00413829  jmp      CSSEDlg::PrtSal+54h (413834h)
0041382B  mov      eax,dword ptr [a]
0041382E  add      eax,1
00413831  mov      dword ptr [a],eax
00413834  mov      eax,dword ptr [direccion]
00413837  add      eax,4
0041383A  cmp      dword ptr [a],eax
0041383D  jge      CSSEDlg::PrtSal+71h (413851h)
{
    __asm
    {
        mov edx,a
        0041383F  mov      edx,dword ptr [a]
        mov eax,datos
        00413842  mov      eax,dword ptr [datos]
        out dx,al
        00413845  out      dx,al
    }
    datos >>= 8;           //obtiene siguiente sección de 8 bits
    00413846  mov      eax,dword ptr [datos]
    00413849  sar      eax,8
    0041384C  mov      dword ptr [datos],eax
}
0041384F  jmp      CSSEDlg::PrtSal+4Bh (41382Bh)

```

La interfaz de entrada del bus ISA de 8 bits

Para ilustrar la interfaz de entrada para el bus ISA, en la figura 15-4 se conecta un par de convertidores ADC804 de analógico a digital al bus ISA. Las conexiones a los convertidores se realizan a través de un conector DB₉ de nueve terminales. La tarea de decodificar las direcciones de los puertos de E/S es más compleja, ya que cada convertidor necesita un pulso de escritura para iniciar una conversión, un pulso de lectura para leer los datos digitales una vez que se han convertido los datos de entrada analógicos y un pulso para habilitar la selección de la salida INTR. Observe que la salida INTR se conecta a la posición de bit D₀ del bus de dato. Cuando se introduce INTR en el microprocesador, se evalúa el bit de más a la derecha de AL para determinar si el convertidor está ocupado.

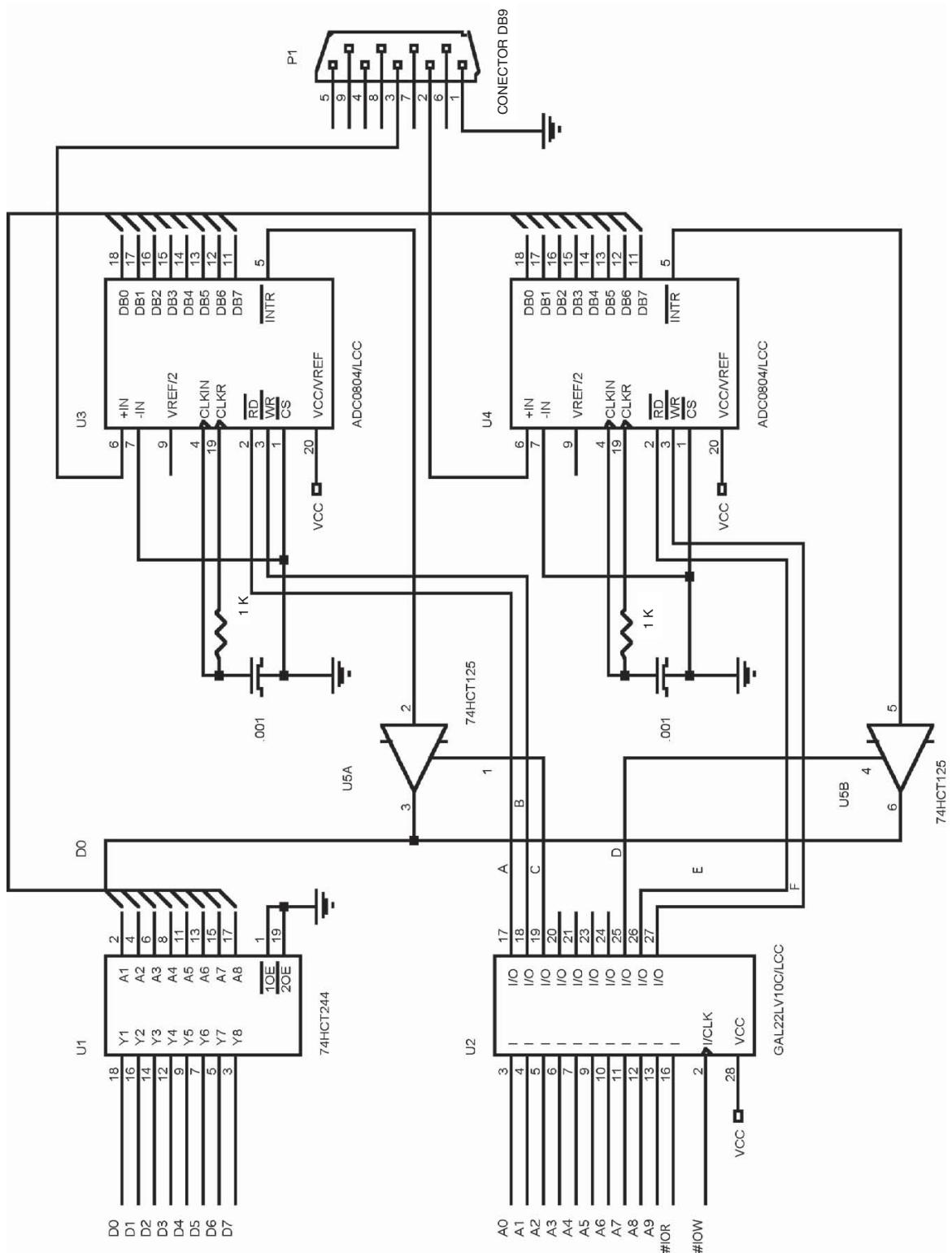


FIGURA 15-4 Un par de convertidores de analógico a digital conectados al bus ISA.

TABLA 15-3 Asignaciones de puertos para la figura 15-4.

<i>Dispositivo</i>	<i>Puerto</i>
Inicia ADC (U ₃)	0300H
Lee ADC (U ₃)	0300H
Lee INTR (U ₃)	0301H
Inicia ADC (U ₄)	0302H
Lee SDC (U ₄)	0302H
Lee INTR (U ₄)	0303H

Como antes, en este caso se ha tenido mucho cuidado en cuanto a que las conexiones del bus ISA presenten una carga unitaria al sistema. La tabla 15-3 ilustra la asignación de puertos de E/S decodificados por el PLD (en el ejemplo 15-3 verá el programa). En este ejemplo hemos supuesto que se utiliza el bus ISA estándar, el cual sólo contiene las conexiones de dirección de la A₀ a la A₉.

EJEMPLO 15-3

```
-- Código VHDL para el decodificador de la figura 15-4

library ieee;
use ieee.std_logic_1164.all;

entity DECODIFICADOR_15_4 is
port (
    IOW, IOR, A9, A8, A7, A6, A5, A4, A3, A2, A1, A0: in STD_LOGIC;
    A, B, C, D, E, F: out STD_LOGIC
);
end;

architecture V1 of DECODER_15_4 is
begin
    A <= not A9 or not A8 or A7 or A6 or A5 or A4 or A3 or A2 or A1 or A0 or
        IOR;
    B <= not A9 or not A8 or A7 or A6 or A5 or A4 or A3 or A2 or A1 or A0 or
        IOW;
    C <= not A9 or not A8 or A7 or A6 or A5 or A4 or A3 or A2 or A1 or not A0
        or IOR;
    D <= not A9 or not A8 or A7 or A6 or A5 or A4 or A3 or A2 or not A1 or
        not A0 or IOR;
    E <= not A9 or not A8 or A7 or A6 or A5 or A4 or A3 or A2 or not A1 or A0
        or IOR;
    F <= not A9 or not A8 or A7 or A6 or A5 or A4 or A3 or A2 or not A1 or A0
        or IOW;
end V1;
```

El ejemplo 15-4 lista una función que lee el ADC U₃ o U₄. La dirección se genera al pasar un 0 para U₃ o un 1 para U₄ al parámetro de dirección de la función. La función inicia el convertidor al escribir en él, pero después espera hasta que la terminal INTR regrese a un 0 lógico, lo cual indica que la conversión está completa antes de que los datos se lean y la función los devuelva como un carácter.

EJEMPLO 15-4

```

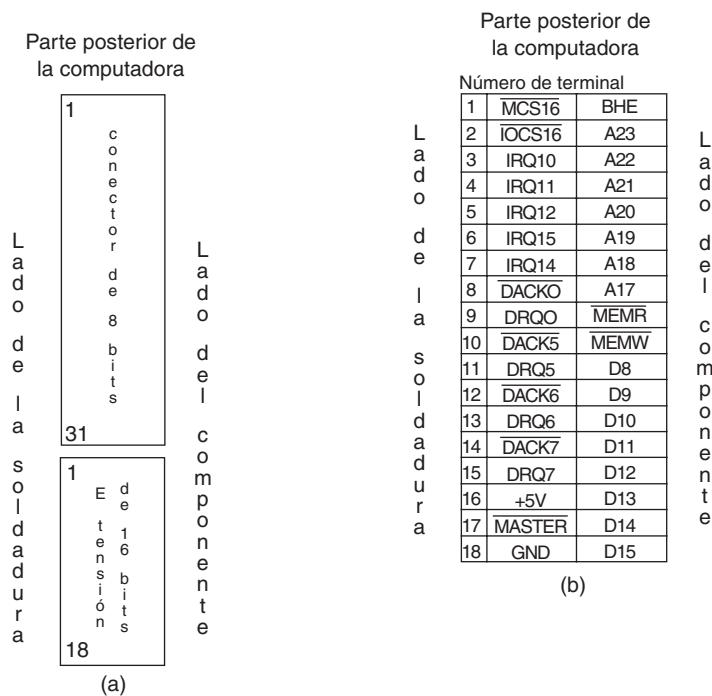
char ADC(int direccion)
{
    char temp = 1;
    if ( direccion )
        direccion = 2;
    direccion += 0x300;
    __asm
    {
        ;inicia el convertidor
        mov    edx,direccion
        out    dx,al
    }
    while ( temp )           //espera si está ocupado
    {
        __asm
        {
            mov    edx,direccion
            inc    edx
            in     al,dx
            mov    temp,al
            and    al,1
        }
        __asm
        {
            ;obtiene los datos
            mov    edx,direccion
            in     al,dx
            mov    temp,al
        }
        return temp;
    }
}

```

El bus ISA de 16 bits

La única diferencia entre el bus ISA de 8 bits y el de 16 bits es que se adjunta un conector adicional detrás del conector de 8 bits. Una tarjeta ISA de 16 bits contiene dos conectores de borde: uno se inserta en el conector original de 8 bits y el otro se inserta en el nuevo conector de 16 bits. La figura 15-5 muestra

FIGURA 15-5 El bus ISA de 16 bits. (a) Ambos conectores de 8 y de 16 bits, y (b) el diagrama de terminales del conector de 16 bits.



el diagrama de terminales del conector adicional y su ubicación en la computadora, en relación con el conector de 8 bits. A menos que se agregue memoria adicional en la tarjeta ISA, las conexiones de dirección adicionales A₂₃-A₂₀ no tendrán ninguna función para las operaciones de E/S. Las características agregadas que se utilizan con más frecuencia son las entradas de petición de interrupción adicionales y las señales de petición de DMA. En algunos sistemas, las operaciones de E/S de 16 bits utilizan las ocho conexiones adicionales del bus de datos (D₈-D₁₅), pero es más común hoy que el bus PCI se utilice para periféricos de más de 8 bits. Las únicas interfaces recientes que hay para el bus ISA son unos cuantos módems y algunas tarjetas de sonido.

15-2

EL BUS DE INTERCONEXIÓN DE COMPONENTES PERIFÉRICOS (PCI)

El bus PCI (**interconexión de componentes periféricos**) es prácticamente el único bus que se encuentra en los sistemas Pentium 4 y en casi todos los sistemas Pentium. En todos los sistemas más recientes se consigue el bus ISA mediante un pedido especial, pero como interfaz para las tarjetas de interfaz antiguas de 8 y de 16 bits. Muchos sistemas nuevos contienen sólo dos ranuras de bus ISA, o ninguna. Con el tiempo tal vez desaparezca el bus ISA, pero aún es una interfaz importante para muchas aplicaciones industriales. El bus PCI ha sustituido al bus local VESA. Una razón es que el bus PCI tiene características de “conectar y usar” (*plug-and-play*) y la habilidad de funcionar con un bus de datos de 64 bits. Una interfaz PCI contiene una serie de registros, ubicados en un pequeño dispositivo de memoria en la interfaz PCI, que contienen información acerca del tablero. Esta misma memoria llega a proporcionar características de “conectar y usar” al bus ISA o a cualquier otro bus. La información en estos registros permite que la computadora configure la tarjeta PCI de manera automática. Esta característica, conocida como **conectar y usar (PnP)**, tal vez sea la razón principal por la que el bus PCI se haya vuelto tan popular en la mayoría de los sistemas.

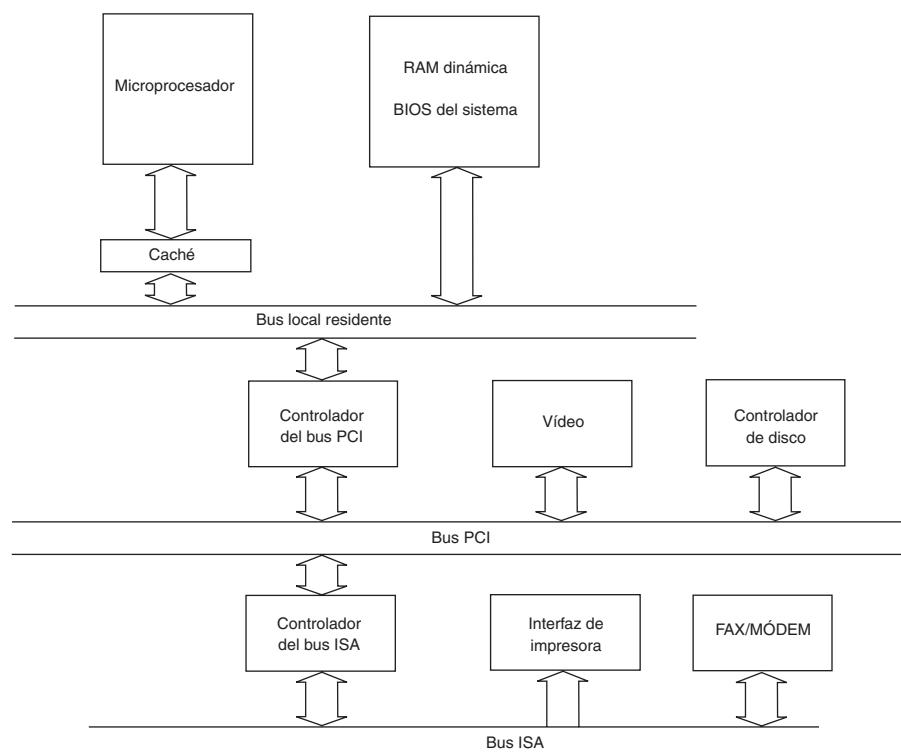


FIGURA 15-6 Diagrama de bloques del sistema para la computadora personal que contiene un bus PCI.

La figura 15-6 muestra la estructura del sistema para el bus PCI en un sistema de computadora personal. Observe que el bus del microprocesador está separado y es independiente del bus PCI. El microprocesador se conecta con el bus PCI a través de un circuito integrado conocido como **puente PCI**. Esto significa que casi cualquier microprocesador logra conectarse al bus PCI, siempre y cuando haya un controlador o un puente PCI diseñado para el sistema. En el futuro, tal vez todos los sistemas utilicen el mismo bus. Incluso hasta el sistema Apple Macintosh está cambiando al bus PCI. Por lo general, al bus local residente se le conoce como bus frontal.

Diagrama de terminales del bus PCI

Al igual que los otros buses que describimos en este capítulo, el bus PCI contiene todas las señales de control del sistema. A diferencia de los otros buses, el bus PCI funciona con un bus de datos de 32 bits o de 64 bits, y con un bus de direcciones completo de 32 bits. Otra diferencia es que los buses de direcciones y de datos se multiplexan para reducir el tamaño del conector de borde. Estas terminales multiplexadas se identifican como AD₀-AD₆₃ en el conector. La tarjeta de 32 bits (común en la mayoría de las computadoras) sólo tiene las conexiones 1 a 62, mientras que la tarjeta de 64 bits tiene las 94 conexiones. La tarjeta de 64 bits puede acomodar una dirección de 64 bits si se requiere en algún momento en el futuro. La figura 15-7 ilustra el diagrama de terminales del bus PCI.

Lo mismo que con los otros sistemas de bus, el bus PCI se utiliza más a menudo para conectar componentes de E/S al microprocesador. La memoria podría conectarse, pero operaría sólo a una velocidad de 33 MHz con el Pentium, que es la mitad de la velocidad del bus local residente de 66 MHz del sistema Pentium. Una versión más reciente del bus PCI (en conformidad con 2.1) opera a 66 MHz y a 33 MHz para las tarjetas de interfaz antiguas. Los sistemas Pentium 4 utilizan una velocidad de bus de sistema de 200 MHz (aunque a menudo se indique que es de 800 MHz), pero no hay una modificación planeada para la velocidad del bus PCI todavía.

Las conexiones de dirección/datos del bus PCI

La dirección PCI aparece en AD₀-AD₃₁ y está multiplexada con los datos. En algunos sistemas hay un bus de datos de 64 bits que utiliza AD₃₂-AD₆₃ sólo para la transferencia de datos. En el futuro estas terminales podrán utilizarse para extender la dirección hasta 64 bits. La figura 15-8 contiene el diagrama de sincronización para el bus PCI, el cual muestra la forma en que se multiplexa la dirección con los datos y también las señales de control que se utilizan para el multiplexeo.

Durante el primer periodo de reloj aparece la dirección de la memoria o la posición de E/S en las conexiones AD, en tanto que el comando para un periférico PCI aparece en las terminales C/B_{E3}-C/B_{E0}. La tabla 15-4 muestra los comandos de bus que se utilizan con el bus PCI.

Secuencia INTA Durante la secuencia de reconocimiento de interrupción, se direcciona un controlador de interrupciones (el controlador que produjo la interrupción) y se interroga para averiguar el vector de interrupción. Este vector de interrupción del tamaño de un byte se devuelve durante una operación de lectura de byte.

TABLA 15-4 Comandos del bus PCI.

C/B _{E3} -C/B _{E0}	Comando
0000	Secuencia INTA
0001	Ciclo especial
0010	Ciclo de lectura de E/S
0011	Ciclo de escritura de E/S
0100-1001	Reservado
1010	Lectura de configuración
1011	Escritura de configuración
1100	Acceso múltiple a memoria
1101	Ciclo de direccionamiento dual
1110	Direccionamiento de memoria en línea
1111	Escritura en memoria con invalidación

FIGURA 15-7 Diagrama de terminales del bus PCI.

Parte posterior de la computadora

Número de terminal		
1	-12V	TRST
2	TCK	+12V
3	GND	TM5
4	TDO	TD1
5	+5V	+5V
6	+5V	INTA
7	INTB	INTC
8	INTD	+5V
9	PRSNT 1	
10		+VI/O
11	PRSNT 2	
12	KEY	KEY
13	KEY	KEY
14		
15	GND	RST
16	CLK	VI/O
17	GND	VNT
18	REQ	GND
19	+V IO	
20	AD31	AD30
21	AD29	+3.3V
22	GND	AD28
23	AD27	AD26
24	AD25	GND
25	+3.3V	AD24
26	C/BE3	IDSEL
27	AD23	+3.3V
28	GND	AD22
29	AD21	AD20
30	AD19	GND
31	+3.3V	AD18
32	AD17	AD16
33	C/BE2	+3.3V
34	GND	FRAME
35	RDY	GND
36	+3.3V	RDY
37	DESEL	GND
38	GND	STOP
39	LOCK	+3.3V
40	PERR	SDONE
Lado de los componentes		
Lado de los componentes		
Notas: 1. Las terminales 63-94 sólo existen en la tarjeta PCI de 64 bits. 2. +VI/O es 3.3 V en una tarjeta de 3.3 V y +5 V en una tarjeta de 5 V. 3. Las terminales en blanco están reservadas.		
41	+3.3V	SBO
42	SERR	GND
43	+3.3V	PAR
44	C/BE1	AD15
45	AD14	+3.3V
46	GND	AD13
47	AD12	AD11
48	AD10	GND
49	GND	AD9
50	KEY	KEY
51	KEY	KEY
52	AD8	C/BE0
53	AD7	+3.3V
54	+3.3V	AD6
55	AD5	AD4
56	AD3	GND
57	GND	AD2
58	AD1	AD0
59	+V IO	+V IO
60	ACK64F	REQ64
61	+5V	+5V
62	+5V	+5V
63		GND
64	GND	C/BE7
65	C/BE6	C/BE5
66	C/BE4	+V IO
67	GND	PAR64
68	AD63	AD62
69	AD61	GND
70	+V IO	AD60
71	AD59	AD58
72	AD57	GND
73	GND	AD56
74	AD55	AD54
75	AD53	+V IO
76	GND	AD52
77	AD51	AD50
78	AD49	GND
79	+V IO	AD48
80	AD47	AD46
81	AD45	GND
82	GND	AD44
83	AD43	AD42
84	AD41	+VI/O
85	GND	AD40
86	AD39	AD38
87	AD37	GND
88	+VI/O	AD36
89	AD35	AD34
90	AD33	GND
91	GND	AD32
92		
93		GND
94	GND	

Ciclo especial

El ciclo especial se utiliza para transferir datos a todos los componentes PCI. Durante este ciclo, los 16 bits de más a la derecha del bus de datos contienen 0000H, lo cual significa que el procesador se ha apagado, 0001H para indicar que el procesador se detuvo, o 0002H para códigos o datos específicos del 80X86.

Ciclo de lectura de E/S

Los datos se leen desde un dispositivo de E/S mediante el uso de la dirección de E/S que aparece en AD₀-AD₁₅. Las lecturas en ráfaga no están soportadas para los dispositivos de E/S.

Ciclo de escritura de E/S

Al igual que con la lectura de E/S, este ciclo accede a un dispositivo de E/S, sólo que escribe datos.

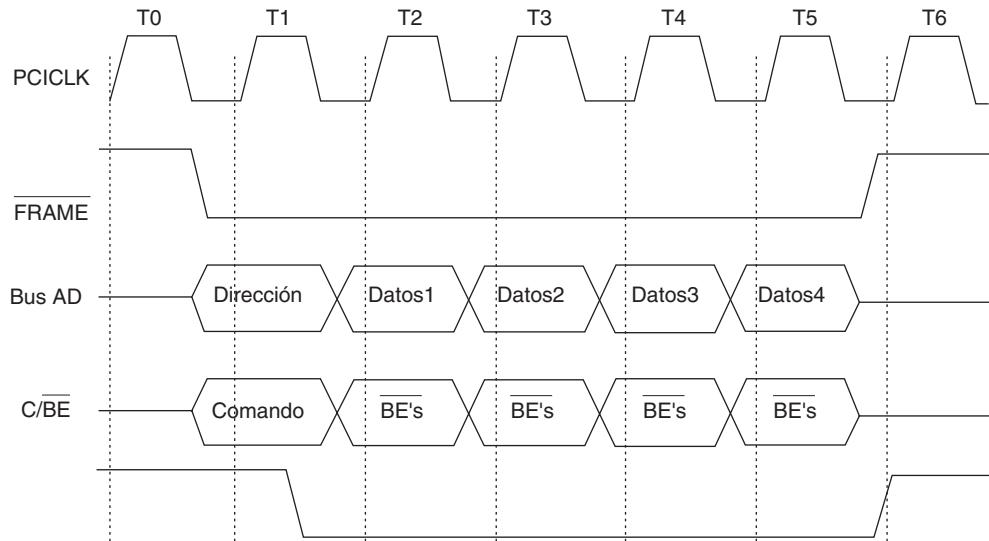


FIGURA 15-8 Sincronización básica del modo de ráfaga para el sistema de bus PCI. Observe que aquí se transfieren ya sea cuatro números de 32 bits (PCI de 32 bits) o cuatro números de 64 bits (PCI de 64 bits).

Ciclo de lectura de memoria Los datos se leen desde un dispositivo de memoria ubicado en el bus PCI.

Ciclo de escritura en memoria Como con la lectura de memoria, se accede a los datos en un dispositivo ubicado en el bus PCI. La ubicación se escribe.

Lectura de configuración La información de configuración se lee del dispositivo PCI mediante el uso del ciclo de lectura de configuración.

Escritura de configuración La escritura de configuración permite escribir datos en el área de configuración de un dispositivo PCI. La dirección se especifica mediante la lectura de configuración.

Acceso múltiple a memoria Es similar al acceso de lectura de memoria, sólo que por lo general se utiliza para acceder a muchos datos en vez de a uno.

Ciclo de direccionamiento dual Se utiliza para transferir la información de dirección a un dispositivo PCI de 64 bits, el cual sólo contiene una ruta de datos de 32 bits.

Direccionamiento de memoria en línea Se utiliza para leer más de dos números de 32 bits del bus PCI.

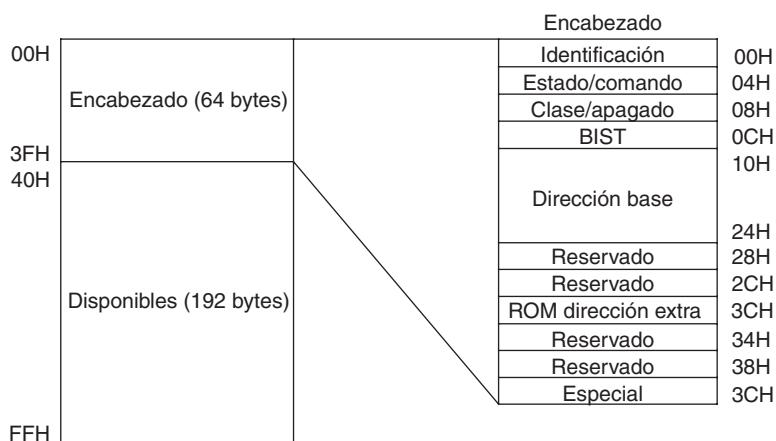
Escritura de memoria con invalidación Es igual que el acceso a memoria en línea, sólo que se utiliza con una escritura. Esta escritura pasa por alto la función de escritura aplazada (*write-back*) de la caché.

Espacio de configuración

La interfaz PCI contiene una memoria de configuración de 256 bytes que permite que la computadora interroge a la interfaz PCI. Esta característica permite que el sistema se configure a sí mismo de manera automática para la tarjeta PCI. Microsoft Corporation llama a tal característica “conectar y usar” (PnP). La figura 15-9 muestra la memoria de configuración y su contenido.

Los primeros 64 bytes de la memoria de configuración contienen el encabezado que guarda información acerca de la interfaz PCI. La primera doble palabra de 32 bits contiene el código de ID de unidad y el código de ID del fabricante. El código de ID de unidad es un número de 16 bits ($D_{31}-D_{16}$)

FIGURA 15-9 El contenido de la memoria de configuración en una tarjeta de expansión PCI.



que es FFFFH, si la unidad no está instalada, y un número entre 0000H y FFFEH, que identifica a la unidad si está instalada. Los códigos de clase identifican la clase de la interfaz PCI. El código de clase se encuentra en los bits D₃₁-D₁₆ de la memoria de configuración, en la posición 08H. Observe que los bits D₁₅-D₀ están definidos por el fabricante. Los códigos de clase actuales se muestran en la tabla 15-5, los cuales son asignados por el SIG de PCI, que es la entidad a cargo del estándar de la interfaz de bus PCI. El ID del fabricante (D₁₅-D₀) también es asignado por el SIG de PCI.

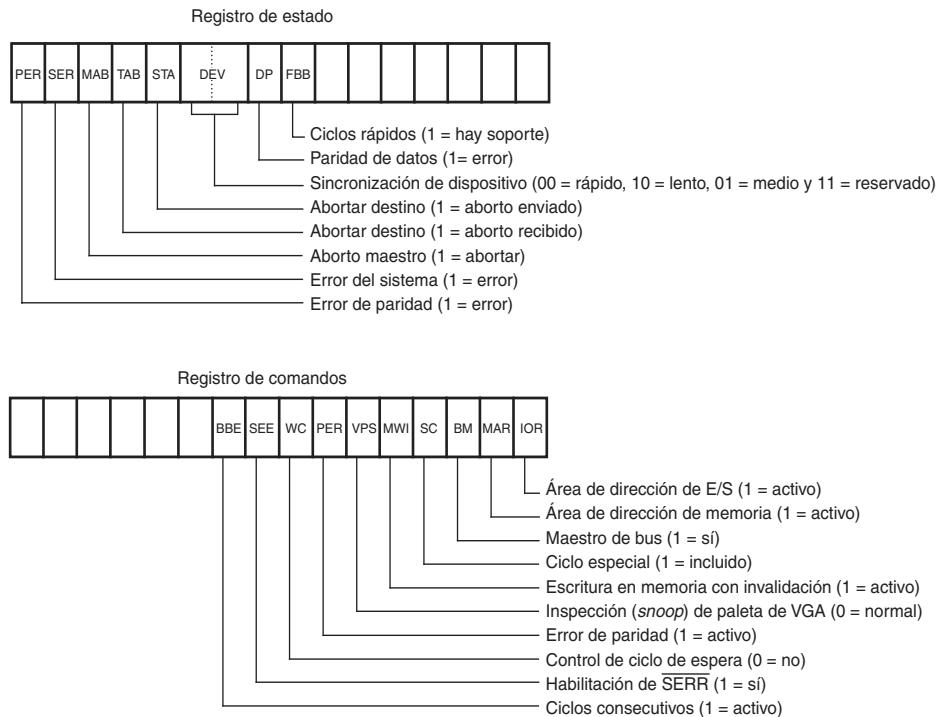
La palabra de estado se carga en los bits D₃₁-D₁₆ de la posición 04H de la memoria de configuración y el comando se encuentra en los bits D₁₅-D₀ de la posición 04H. La figura 15-10 muestra el formato de los registros de estado y de comandos.

TABLA 15-5
Los códigos de clase.

Código de clase	Función
0000H	Dispositivo no VGA antiguo (no PnP)
0001H	Dispositivo VGA antiguo (no PnP)
0100H	Controlador SCSI
0101H	Controlador IDE
0102H	Controlador de disco flexible
0103H	Controlador IPI
0180H	Otro controlador de disco duro/flexible
0200H	Controlador de Ethernet
0201H	Controlador de Token Ring
0202H	FDDI
0280H	Otro controlador de red
0300H	Controlador VGA
0301H	Controlador XGA
0380H	Otro controlador de video
0400H	Multimedia de video
0480H	Otro controlador multimedia
0500H	Controlador de RAM
0580H	Otro controlador de puente de memoria
0600H	Controlador de anfitrión
0601H	Puente ISA
0602H	Puente EISA
0603H	Puente MCA
0604H	Puente PCI-PCI
0605H	Puente PCMCIA
0680H	Otro puente
0700H-FFFEH	Reservado
FFFFH	No instalado

FIGURA 15-10

El contenido de las palabras de estado y de control en la memoria de configuración.



El espacio de dirección base consiste en una dirección base para la memoria, una segunda para el espacio de E/S y una tercera para la ROM de expansión. Las primeras dos dobles palabras del espacio de dirección base contienen la dirección base de 32 o de 64 bits para la memoria presente en la interfaz PCI. La siguiente doble palabra contiene la dirección base del espacio de E/S. Aún y cuando los microprocesadores Intel sólo utilizan una dirección de E/S de 16 bits, hay espacio para expandir la dirección de E/S a 32 bits. Lo anterior permite a los sistemas que utilizan las familias 680X0 y PowerPC el acceso al bus PCI, ya que tiene espacio de E/S, al cual se accede a través de una dirección de 32 bits. Los microprocesadores 680X0 y PowerPC utilizan E/S por asignación de memoria, como vimos al principio del capítulo 11.

BIOS para PCI

La mayoría de las computadoras personales modernas basadas en el Pentium-Pentium 4 contienen el bus PCI y una extensión al BIOS normal del sistema que soporta el bus PCI. Tales sistemas más recientes contienen el acceso al bus PCI en el vector de interrupción 1AH. La tabla 15-6 muestra las funciones disponibles hasta este momento a través de la instrucción INT 1AH del DOS, con AH = 0B1H para el bus PCI.

El ejemplo 15-5 muestra cómo se utiliza el BIOS para determinar si está disponible la extensión del bus PCI. Una vez que se establece la presencia del BIOS, es posible leer el contenido de la memoria de configuración mediante el uso de las funciones del BIOS. Hay que tomar en cuenta que el BIOS no soporta las transferencias de datos entre la computadora y la interfaz PCI. Estas transferencias de datos se manejan a través de controladores que se proporcionan con la interfaz. Estos controladores determinan el flujo de datos entre el microprocesador y el componente que se encuentre en la interfaz PCI.

EJEMPLO 15-5

```
; Programa de DOS que determina si existe el bus PCI
.MODEL SMALL
.DATA
    MENS1 DB      "EL BUS PCI ESTÁ PRESENTE"
    MENS1 DB      "EL BUS PCI NO SE ENCONTRÓ"
.CODE
```

TABLA 15-6 Funciones INT 1AH del BIOS para el bus PCI.

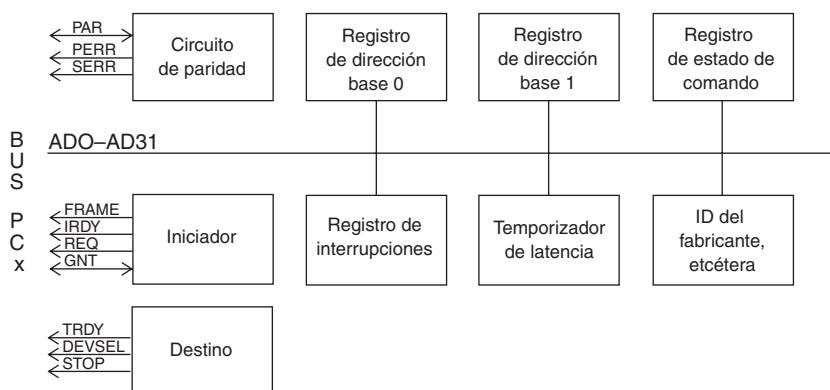
01H ¿Está disponible el BIOS?	
Entrada	AH = 0B1H AL = 01H
Salida	AH = 00H si está disponible la extensión PCI del BIOS BX = número de versión EDX = Cadena ASCII 'PCI' Acarreo = 1 si no está presente la extensión PCI
02H Búsqueda de unidad PCI	
Entrada	AH = 0B1H AL = 02H CX = Unidad DX = Fabricante SI = Índice
Salida	AH = código de resultado (vea las observaciones) BX = número de bus y de unidad Acarreo = 1 si hay error
Observaciones	Los códigos de resultado son: 00H = búsqueda con éxito 81H = la función no se soporta 83H = código de ID de fabricante inválido 86H = no se encontró la unidad 87H = número de registro inválido
03H Búsqueda de código de clase PCI	
Entrada	AH = 0B1H AL = 03H ECX = código de clase SI = índice
Salida	AH = código de resultado (vea las observaciones para la función 02H) BX = número de bus y de unidad Acarreo = 1 si hay error
06H Inicio de ciclo especial	
Entrada	AH = 0B1H AL = 06H BX = número de bus y de unidad EDX = datos
Salida	AH = código de resultado (vea las observaciones para la función 02H) Acarreo = 1 si hay error
Observaciones	El valor que se pasa en EDX se envía al bus PCI durante la fase de dirección.
08H Lectura de configuración tamaño byte	
Entrada	AH = 0B1H AL = 08H BX = número de bus y de unidad DI = número de registro
Salida	AH = código de resultado (vea las observaciones para la función 02H) CL = datos del registro de configuración Acarreo = 1 si hay error

(continúa en la siguiente página)

TABLA 15-6 (*continuación*)

09H	Lectura de configuración tamaño palabra
Entrada	AH = 0B1H AL = 08H BX = número de bus y de unidad DI = número de registro
Salida	AH = código de resultado (vea las observaciones para la función 02H) CX = datos del registro de configuración Acarreo = 1 si hay error
0AH	Lectura de configuración tamaño doble palabra
Entrada	AH = 0B1H AL = 08H BX = número de bus y de unidad DI = número de registro
Salida	AH = código de resultado (vea las observaciones para la función 02H) ECX = datos del registro de configuración Acarreo = 1 si hay error
0BH	Escritura de configuración tamaño byte
Entrada	AH = 0B1H AL = 08H BX = número de bus y de unidad CL = datos que se van a escribir en el registro de configuración DI = número de registro
Salida	AH = código de resultado (vea las observaciones para la función 02H) Acarreo = 1 si hay error
0CH	Escritura de configuración tamaño palabra
Entrada	AH = 0B1H AL = 08H BX = número de bus y de unidad CX = datos que se van a escribir en el registro de configuración DI = número de registro
Salida	AH = código de resultado (vea las observaciones para la función 02H) Acarreo = 1 si hay error
0DH	Escritura de configuración tamaño doble palabra
Entrada	AH = 0B1H AL = 08H BX = número de bus y de unidad ECX = datos que se van a escribir en el registro de configuración DI = número de registro
Salida	AH = código de resultado (vea las observaciones para la función 02H) Acarreo = 1 si hay error

FIGURA 15-11 Diagrama de bloques de la interfaz PCI.



```

.STARTUP
    MOV     AH,0B1H      ; acceso al BIOS de PCI
    MOV     AL,1
    INT     1AH
    MOV     DX,OFFSET MENS2
    .IF CARRY?           ; si el bus PCI está presente
        MOV     DX,OFFSET MENS1
    .ENDIF
    MOV     AH,9          ; muestra MENS1 o MENS2
    INT     21H
    .EXIT
END

```

Interfaz PCI

La interfaz PCI es compleja; por lo general se utiliza un controlador de bus PCI integrado para conectarla con el bus PCI. Se requiere memoria (EPROM) para almacenar la información del fabricante y demás información, como ya explicamos en esta sección del capítulo. La estructura básica de la interfaz PCI se ilustra en la figura 15-11. El contenido de este diagrama de bloques muestra los componentes requeridos para una interfaz PCI funcional; no muestra a la interfaz en sí. Los registros, el bloque de paridad, el iniciador, el destino y la EPROM del ID del fabricante son componentes requeridos de cualquier interfaz PCI. Si se construye una interfaz de este tipo, por lo general se utiliza un controlador PCI, debido a su complejidad. El controlador PCI proporciona las estructuras que se muestran en la figura 15-11.

Bus PCI Express

El bus PCI Express, aunque todavía no está disponible en las PC, transfiere datos en serie a la velocidad de 2.5 GHz para las aplicaciones PCI heredadas, lo cual aumenta la velocidad del vínculo de datos de 500 MBps a 16 GBps. El bus PCI estándar transmite datos a una velocidad aproximada de 133 MBps, en comparación. La gran mejora está en la tarjeta madre, en donde las interconexiones son en serie y operan a 2.5 GHz.

Cuando esté disponible llegará a reemplazar las tarjetas de vídeo actuales en el puerto AGP con una versión de mayor velocidad en el bus PCI Express (PCI-X). Esta tecnología en serie permite que los fabricantes de tarjetas principales utilicen menos espacio para la interconexión, lo cual reduce el costo.

La interfaz de impresora en paralelo (LPT, que en inglés significa impresora en línea) se encuentra en la parte posterior de la computadora personal. Mientras siga formando parte de la PC, podrá utilizarse como interfaz. Esta interfaz de impresora proporciona acceso al usuario a ocho líneas que pueden programarse para recibir o enviar datos en paralelo.

TABLA 15-7 Descripciones de las señales para los conectores de la figura 15-12.

<i>Señal</i>	<i>Descripción</i>	<i>25 terminales</i>	<i>36 terminales</i>
#STR	Estrobo a impresora	1	1
D0	Bit de datos 0	2	2
D1	Bit de datos 1	3	3
D2	Bit de datos 2	4	4
D3	Bit de datos 3	5	5
D4	Bit de datos 4	6	6
D5	Bit de datos 5	7	7
D6	Bit de datos 6	8	8
D7	Bit de datos 7	9	9
#ACK	Reconocimiento de la impresora	10	10
BUSY	Señal de ocupado de la impresora	11	11
PAPER	Sin papel	12	12
ONLINE	La impresora está en línea	13	13
#ALF	Bajo si la impresora envía LF después de CR	14	14
#ERROR	Error de impresora	15	32
#RESET	Reinicia la impresora	16	31
#SEL	Selecciona la impresora	17	36
+5 V	5 V de la impresora	—	18
Tierra de protección	Tierra física	—	17
Tierra de señal	Tierra de señal	Todas las demás terminales	Todas las demás terminales

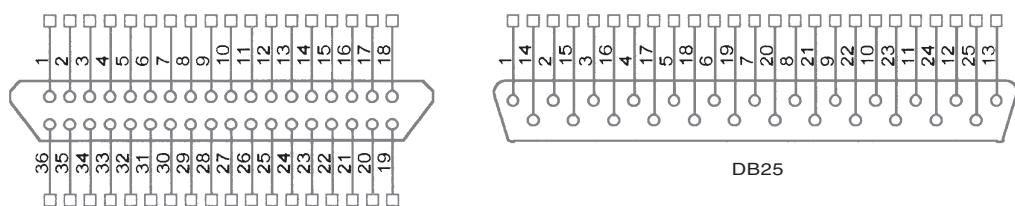
Nota: # indica una señal activa en nivel bajo.

Detalles del puerto

Por lo general, el puerto paralelo (LPT_1) se encuentra en las direcciones de puerto de E/S 378H, 379H y 37AH desde el DOS o mediante el uso de un controlador en Windows. El puerto secundario (LPT_2), si está presente, se encuentra en las direcciones de puerto de E/S 278H, 279H y 27AH. La siguiente información se aplica a ambos puertos, pero se utilizan las direcciones del puerto LPT_1 .

La interfaz Centronics, que implementa el puerto paralelo, utiliza dos conectores, uno tipo D de 25 terminales en la parte posterior de la PC y un Centronics de 36 terminales en la parte posterior de la impresora. En la tabla 15-7 se muestra el diagrama de terminales de estos conectores; en la figura 15-12 están los conectores.

El puerto paralelo puede trabajar como receptor y como transmisor en sus terminales de datos (D_0-D_7). Lo anterior permite la conexión de otros dispositivos que no sean impresoras, como CD-ROMs, para usarlos con la PC a través del puerto paralelo. Cualquier cosa que reciba y/o envíe datos a través de una interfaz de 8 bits tendrá la posibilidad de conectarse (y con frecuencia lo hará) al puerto paralelo (LPT_1) de una PC.



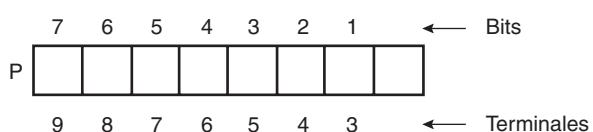
CENTRONICS DE 36

FIGURA 15-12 Los conectores utilizados para el puerto paralelo.

FIGURA 15-13 Los puertos 378H, 379H y 37AH que utilizan el puerto paralelo.

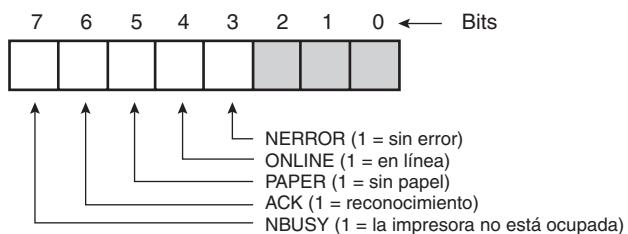
Puerto 378H

El puerto de datos que se conecta a los bits D₀-D₇ (terminales 2-9)

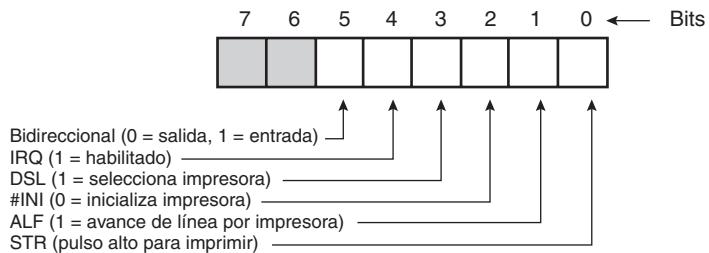


Puerto 379H

Es un puerto de sólo lectura que devuelve la información de la impresora a través de señales tales como BUSY, #ERROR y así sucesivamente. (Tenga cuidado; algunos de los bits están invertidos.)



Puerto 37AH



La figura 15-13 muestra el contenido del puerto de datos (378H), del registro de estado (379H) y de un puerto de estado adicional (37AH). Algunos de los bits de estado son verdadero cuando son un cero lógico.

Uso del puerto paralelo sin soporte para ECP

Para la mayoría de los sistemas desde que IBM lanzó la PS/2 al mercado, podemos seguir la información que se presenta en la figura 15-13 para utilizar el puerto paralelo sin ECP. Para leer el puerto, primero hay que inicializarlo mediante el envío del valor 20H al registro 37AH, como se muestra en el ejemplo 15-6. Como lo indica la figura 15-13, esto activa el bit bidireccional que selecciona la operación de entrada para el puerto paralelo. Si el bit está desactivado, se selecciona la operación de salida.

EJEMPLO 15-6

```
MOV     AL, 20H
MOV     DX, 37AH
OUT    DX, AL
```

Una vez que se programa el puerto paralelo como entrada, se lee como se indica en el ejemplo 15-7. La lectura se lleva a cabo mediante el acceso al puerto de datos en la dirección 378H.

EJEMPLO 15-7

```
MOV      DX, 378H  
IN       AL, DX
```

Para escribir datos al puerto paralelo, se reprograma el registro de comandos en la dirección 37A mediante la escritura del valor 00H para programar el bit bidireccional con un cero. Una vez que se programa el bit bidireccional, se envían datos al puerto paralelo a través del puerto de datos en la dirección 378H. El ejemplo 15-8 muestra cómo se envían los datos al puerto paralelo.

EJEMPLO 15-8

```
MOV      DX, 378H  
MOV      AL, ESCRIBE_DATOS  
OUT     DX, AL
```

En equipos más antiguos (basados en el 80286) el bit direccional no está en la interfaz. Para leer información del puerto paralelo se escribe 0FFH en el puerto (378H) y después se lee. Estos sistemas antiguos no tienen un registro en la posición 37AH.

Es difícil acceder al puerto de la impresora desde Windows, ya que se debe escribir un controlador para ello si se utiliza Windows 2000 o Windows XP. En Windows 98 o Windows ME, el acceso al puerto se realiza según lo que explicamos en esta sección.

Hay una forma de acceder al puerto paralelo a través de Windows 2000 y Windows XP sin escribir un controlador. Un controlador llamado UserPort (disponible a través de Internet) abre los puertos de E/S protegidos en Windows y permite el acceso directo al puerto paralelo a través de bloques de ensamblador en Visual C++, mediante el uso del puerto 378H. También permite el acceso a cualquier puerto de E/S entre 0000H y 03FFH. Hay otra herramienta útil disponible para una prueba de 30 días en www.jungo.com. La herramienta Jungo sirve para desarrollo de controladores, con muchos controladores de ejemplo para la mayoría de los subsistemas.

15-4**LOS PUERTOS SERIALES COM**

Los puertos de comunicaciones seriales son COM₁-COM₈, pero la mayoría de las computadoras sólo tienen instalados COM₁ y COM₂. Algunas tienen un solo puerto de comunicaciones (COM₁). En el capítulo 11 describimos cómo controlar y acceder a estos puertos en el entorno DOS mediante el componente de interfaz serial 16550, por lo que no volveremos a hablar de eso. En vez de ello trataremos las funciones de la API de Windows para operar los puertos COM para la interfaz de comunicaciones 16550.

Control de las comunicaciones

Los puertos seriales se acceden a través de cualquier versión de Windows y Visual C++ mediante el uso de unas cuantas funciones de la interfaz de aplicaciones del sistema (API). En el ejemplo 15-9 se muestra una función corta en C++ que accede a los puertos seriales. La función se llama EscribePuertoCom y contiene dos parámetros. El primer parámetro es el puerto, como en COM₁, COM₂ y así sucesivamente, en tanto que el segundo parámetro es el carácter que se va a enviar a través del puerto. Un valor de retorno verdadero indica que el carácter se envío y un valor de retorno falso muestra que hay un problema. Para usar la función para enviar la letra A a través del puerto COM₁, el puerto se atrae con EscribePuertoCom ("COM₁", "A"). Tal función está escrita para enviar sólo un byte individual a través del puerto serial COM, pero podría modificarse para enviar cadenas de caracteres. Para enviar 00H (ningún otro número es posible enviar de esta manera) a través de COM₂ se utiliza EscribePuertoCom ("COM₂", 0x00). El puerto COM fue configurado para funcionar a 9600 baudios, pero llega a modificarse con facilidad si cambia CBR_9600 a otro valor aceptable. En la tabla 15-8 consulte las velocidades en baudios permitidas.

TABLA 15-8 Velocidades en baudios permitidas para los puertos COM.

<i>Palabra clave</i>	<i>Velocidad en bits por segundo</i>
CBR_110	110
CBR_300	300
CBR_600	600
CBR_1200	1200
CBR_2400	2400
CBR_4800	4800
CBR_9600	9600
CBR_14400	14400
CBR_19200	19200
CBR_38400	38400
CBR_56000	56000
CBR_57600	57600
CBR_115200	115200
CBR_128000	128000
CBR_256000	256000

EJEMPLO 15-9

```
bool EscribePuertoCom(CString EspecificadorPuerto, CString datos)
{
    DCB dcb;
    DWORD bytesescritos;

    HANDLE hPuerto = CreateFile(EspecificadorPuerto,
        GENERIC_WRITE,
        0
        NULL,
        OPEN_EXISTING,
        0,
        NULL);

    If (!GetCommState(hPuerto,&dcb)) {
        return false;
    }

    dcb.BaudRate = CBR_9600;           //9600 baudios
    dcb.ByteSize = 8;                 //8 bits de datos
    dcb.Parity = NOPARITY;            //sin paridad
    dcb.StopBits = ONESTOPBIT;        //1 bit de paro

    if (!SetCommState(hPuerto,&dcb))
        return false;

    bool valRet = WriteFile(hPuerto,datos,1,&bytesescritos,NULL);
    CloseHandle(hPuerto);             //cierra el manejador
    return valRet;
}
```

La estructura CreateFile crea un manejador para los puertos COM, el cual puede usarse para escribir datos en el puerto. Después de obtener y modificar el estado del puerto para que cumpla con los requerimientos de velocidad en baudios, la función WriteFile envía los datos al puerto. Los parámetros que se utilizan con la función WriteFile son: el manejador del archivo (hPuerto), los datos que se van a escribir en forma de cadena de caracteres, el número de bytes que se van a escribir (1 en este ejemplo) y un lugar para almacenar el número de bytes que en realidad se escribieron en el puerto.

La recepción de datos a través del puerto COM es un proceso un poco más complicado, ya que es más frecuente que ocurran errores que en la transmisión. También hay muchos tipos de errores que pueden detectarse y deberían reportarse con frecuencia al usuario. El ejemplo 15-10 muestra una función en C++ que se utiliza para leer un carácter desde el puerto serial, la cual se llama LeeByte. Esta

función devuelve ya sea el carácter que se lee desde el puerto, un código de error de 0×100 , si no se abrió el puerto o un código de error de 0×101 si el receptor detectó un error. Si no se recibieron datos, esta función se paralizará debido a que no se establecieron tiempos de espera.

EJEMPLO 15-10

```

int LeeByte(CString EspecificadorPuerto)
{
    DCB dcb;
    int valRet;
    BYTE Byte;
    DWORD dwBytesTransferidos;
    DWORD dwEstadoCommsModem

    HANDLE hPuerto = CreateFile(EspecificadorPuerto,
        GENERIC_READ,
        0,
        NULL,
        OPEN_EXISTING,
        0,
        NULL);

    if (!GetCommState(hPuerto,&dcb))
        return 0x100;

    dcb.BaudRate = CBR_9600;                      //9600 baudios
    dcb.ByteSize = 8;                             //8 bits de datos
    dcb.Parity = NOPARITY;                        //sin paridad
    dcb.StopBits = ONESTOPBIT;                     //1 de paro

    if (!SetCommState(hPuerto,&dcb))
        return 0x100;

    SetCommMask (hPuerto, EV_RXCHAR | EV_ERR);   //evento de recibir carácter
    WaitCommEvent (hPuerto, &dwEstadoCommsModem, 0); //espera el carácter

    if (dwEstadoCommsModem & EV_RXCHAR)
        ReadFile (hPuerto, &Byte, 1, &dwBytesTransferidos, 0); //lee 1
    else if (dwEstadoCommsModem & EV_ERR)
        valRet = 0x101;
    valRet = Byte;
    CloseHandle(hPuerto);
    return valRet;
}

```

15-5

EL BUS SERIAL UNIVERSAL (USB)

Este bus ha resuelto un problema con el sistema de computadora personal. Las tarjetas de sonido PCI actuales utilizan la fuente de energía interna de la PC, la cual genera una gran cantidad de ruido. Como el USB deja que la tarjeta de sonido tenga su propia fuente de energía llega a eliminarse el ruido asociado con la fuente de energía de la PC, lo cual permite un sonido de alta fidelidad sin el zumbido de 60 Hz. Otros beneficios son la facilidad de conexión para el usuario y el acceso hasta a 127 conexiones distintas mediante un cable serial de cuatro conexiones. Dicha interfaz es ideal para los teclados, las tarjetas de sonido, los dispositivos de recuperación de vídeo simples y los módems. Las velocidades de transferencia de datos son de 480 Mbps para la operación con el USB 2.0, de 11 Mbps para las transferencias en conformidad con el USB 1.1 y de 1.5 Mbps para la operación en baja velocidad.

La longitud máxima de los cables está limitada a 5 metros para la interfaz de máxima velocidad y a 3 metros para la interfaz de baja velocidad. La energía máxima disponible a través de estos cables se clasifica en 100 mA, con una corriente máxima a 5.0 V. Si la cantidad de corriente se excede de 100 mA, Windows mostrará un signo de admiración amarillo al lado del dispositivo, con lo cual se indica una condición de sobrecarga.

FIGURA 15-14 Vista frontal de los dos tipos comunes de conectores USB.



TABLA 15-9 Configuración de terminales del USB.

Número de terminal	Señal
1	+5.0 V
2	-Datos
3	+Datos
4	Tierra

El conector

La figura 15-14 muestra el diagrama de terminales del conector USB. Existen dos tipos de conectores especificados, ambos en uso. En cualquier caso hay cuatro terminales en cada conector, las cuales contienen las señales que se indican en la tabla 15-9. Como mencionamos, las señales de +5.0 V y de tierra pueden usarse para operar los dispositivos de poder conectados al bus, siempre y cuando la cantidad de corriente no sea mayor de 100 mA por dispositivo. Las señales de datos son señales bifásicas. Cuando la terminal +datos está a 5.0 V, -datos está a cero voltios y viceversa.

Datos USB

Las señales de datos son señales bifásicas, que se generan mediante el uso de un circuito tal como el que se muestra en la figura 15-15. El receptor de línea también se muestra en esta figura. En el par de transmisión está colocado un circuito de supresión de ruido disponible a través de Texas Instruments (SN75240). Una vez que el transceptor está en su lugar, la interfaz para el USB está completa. El circuito integrado 75773 de Texas Instruments funciona como el controlador de línea diferencial y como el receptor para este diagrama esquemático.

La siguiente fase es aprender cómo interactúan las señales en el USB. Estas señales permiten enviar y recibir datos desde el sistema de computadora anfitrión. El USB utiliza la codificación de datos NRZI (sin retorno a cero invertido) para transmitir paquetes. Este método de codificación no modifica el nivel de la señal para la transmisión de un 1 lógico, pero el nivel de la señal se invierte cada vez que se cambia a 0 lógico. La figura 15-16 muestra un flujo de datos digitales y la señal USB que se produce mediante el uso de este método de codificación.

FIGURA 15-15 La interfaz para el USB, en la que se utiliza un par de búferes CMOS.

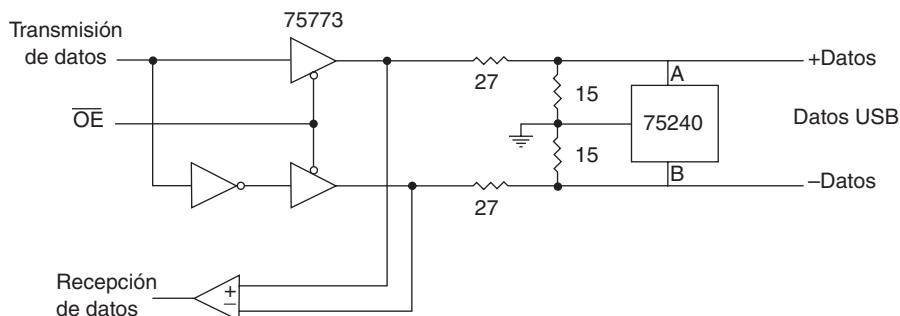
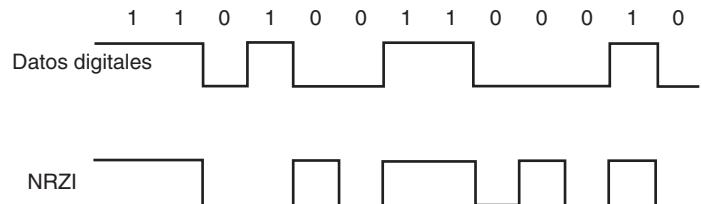


FIGURA 15-16

La codificación NRZI que se utiliza con el USB.



Los datos que se transmiten incluyen bits de sincronización mediante el uso de un método llamado *relleno de bits*. Si se transmite un 1 lógico durante más de seis bits seguidos, la técnica de relleno de bits agrega un bit adicional (0 lógico) después de seis 1s continuos en una fila. Como esta técnica extiende el flujo de datos, se le conoce como relleno de bits. La figura 15-17 muestra un flujo de datos serial con relleno de bits y el algoritmo que se utiliza para crearlo a partir de datos seriales puros. El relleno de bits asegura que el receptor mantenga la sincronización para cadenas extensas de 1s. Los datos siempre se transmiten empezando desde el bit menos significativo y después los bits siguientes.

Comandos USB

Ahora que entendemos el formato de datos del USB, hablaremos sobre los comandos que se utilizan para transferir datos y seleccionar el receptor. Para iniciar las comunicaciones se transmite primero el byte de sincronización (80H), seguido del byte de identificación de paquete (PID). El PID contiene ocho bits, pero sólo los cuatro bits de más a la derecha traen el tipo de paquete que sigue, en caso de que lo haya. Los cuatro bits de más a la izquierda del PID son los que complementan a los cuatro bits de más a la derecha. Por ejemplo, si se envía un comando de 1000, el byte que se envía para el PID es 0111

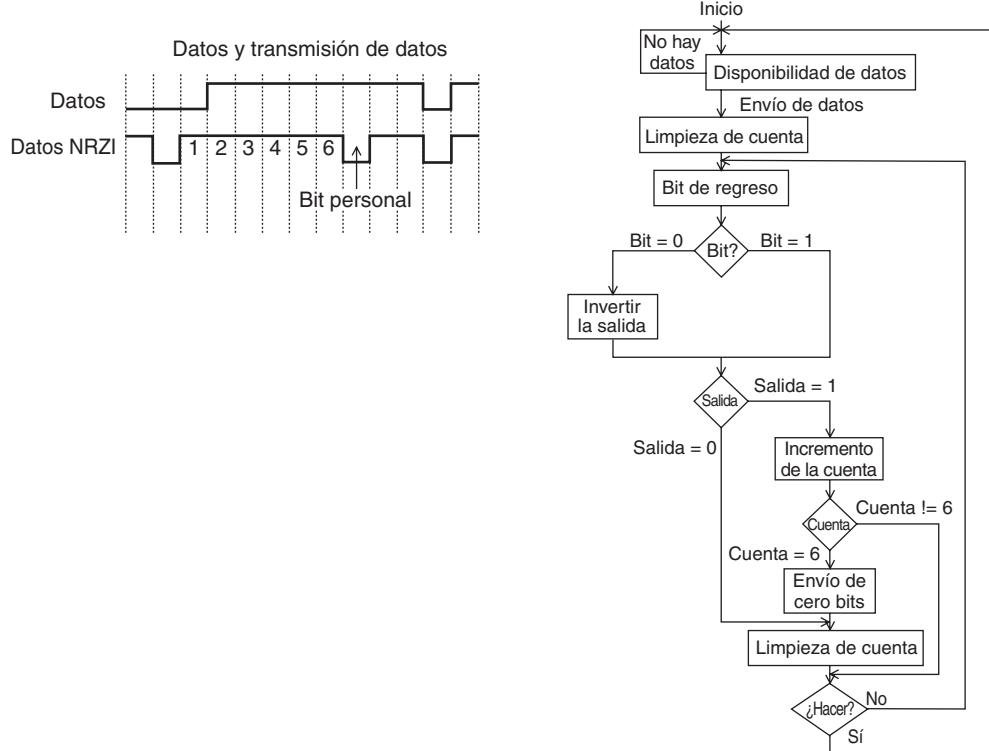


FIGURA 15-17 El flujo de datos y el diagrama de flujo que se utilizan para generar los datos del USB.

TABLA 15-10 Códigos del PID.

PID	Nombre	Tipo	Descripción
E ₁	OUT	Token	Anfitrión → transacción de función.
D ₂	ACK	Protocolo de intercambio (handshake)	El receptor acepta el paquete.
C ₃	Datos0	Datos	Paquete de datos (PID par).
A ₅	SOF	Token	Inicio de trama.
69	IN	Token	Función → transacción de anfitrión.
5A	NAK	Protocolo de intercambio (handshake)	El receptor no acepta el paquete.
4B	Datos1	Datos	Paquete de datos (PID impar).
3C	PRE	Especial	Preámbulo del anfitrión.
2D	Setup	Token	Comando de configuración.
1E	Stall	Token	Paralizado.

1000. La tabla 15-10 muestra los PIDs de cuatro bits disponibles y sus códigos de ocho bits. Observe que los PIDs se utilizan como indicadores de token, como indicadores de datos y para el protocolo de intercambio.

La figura 15-18 lista los formatos de los paquetes de datos, de token, de protocolo de intercambio y de inicio de trama que se utilizan en el USB. En el paquete de token, el ADDR (campo de dirección) contiene la dirección de siete bits del dispositivo USB. Como ya se dijo, puede haber hasta 127 dispositivos presentes en el USB en un momento dado. El ENDP (punto final) es un número de cuatro bits utilizado por el USB. El punto final 0000 se utiliza para la inicialización; otros números de punto final son únicos para cada dispositivo USB.

Hay dos tipos de CRC (**comprobaciones de redundancia cíclica**) que se utilizan en el USB: la CRC de cinco bits y la CRC de 16 bits (la que se utiliza para los paquetes de datos). La CRC de cinco bits se genera mediante el polinomio $X_5 + X_2 + 1$; la CRC de 16 bits se genera mediante el polinomio $X_{16} + X_{15} + X_2 + 1$. Cuando se construyen circuitos para generar o detectar la CRC, el signo positivo representa circuitos OR exclusivos. El circuito CRC o programa es un mecanismo de comprobación serial. Cuando se utiliza la CRC de cinco bits se recibe un residuo de 01100 si no hay error en todos los cinco bits de la CRC y en los bits de datos. Con la CRC de 16 bits, el residuo es 1000000000001101 cuando no hay error.

El USB utiliza los tokens ACK y NAK para coordinar la transferencia de paquetes de datos entre el sistema anfitrión y el dispositivo USB. Una vez que se transfiere un paquete de datos, desde el

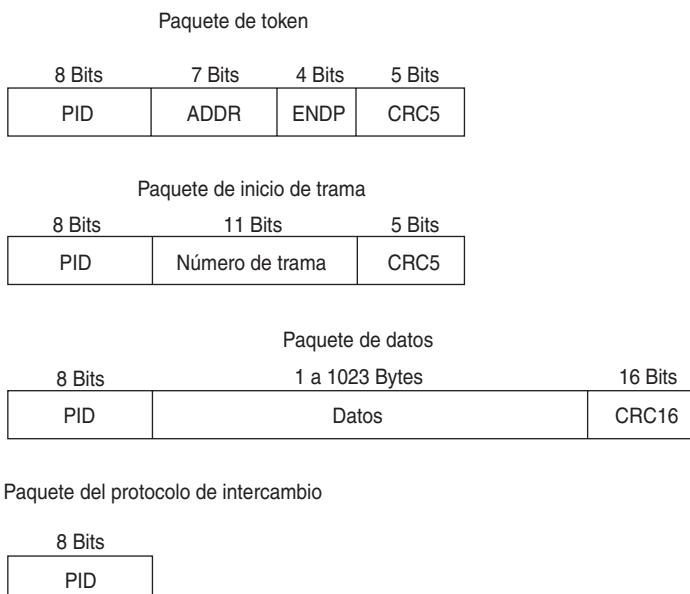
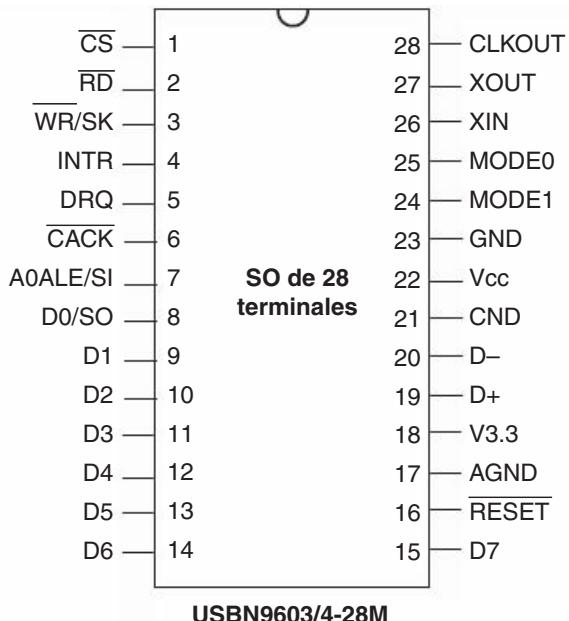
FIGURA 15-18 Los tipos de paquetes y el contenido que se encuentra en el USB.

FIGURA 15-19 El nodo del bus USB, de National Semiconductor.



anfitrión hacia el dispositivo USB, éste transmite un token ACK (aceptación) o NAK (no aceptación) de vuelta al anfitrión. Si los datos y la CRC se reciben en forma correcta, se envía el ACK; si no, lo hace el NAK. Si el anfitrión recibe un token NAK, vuelve a transmitir el paquete de datos hasta que el receptor final lo reciba en forma correcta. A menudo, a este método de transferencia de datos se le conoce como **control de flujo con paro y espera**. El anfitrión debe esperar a que el cliente envíe un token ACK o uno NAK antes de transferir paquetes de datos adicionales.

El nodo del bus USB

National Semiconductor produce una interfaz de bus USB que es muy fácil de conectar al microprocesador. La figura 15-19 ilustra el nodo USB USBN9604. Para conectar este dispositivo a un sistema que utilice acceso sin DMA se conecta el bus de datos a D₀-D₇, las entradas de control RD, WR y CS, así como un cristal fundamental de 24 MHz a través de las terminales X_{IN} y X_{OUT}. La conexión del bus USB se encuentra en las terminales D- y D+. La interfaz más simple se obtiene mediante la conexión de las dos entradas de modo a tierra. Esto coloca al dispositivo en un modo paralelo no multiplexado. En este modo, la terminal A₀ se utiliza para seleccionar la dirección (1) o los datos (0). La figura 15-20 muestra esta conexión con el microprocesador en las direcciones de los puertos de E/S 0300H (datos) y 0301H (dirección).

El USBN9604 es un transceptor de bus USB que puede recibir y transmitir datos por el USB. Esto proporciona un punto de interfaz para el USB por un costo mínimo aproximado de dos dólares.

Software para el USBN9604/3

El software que se presenta aquí funciona con la interfaz en la figura 15-20. Lo que no se proporciona es el software controlador para el sistema anfitrión. El ejemplo 15-11 muestra el código requerido para inicializar el controlador USB. El procedimiento USBINT configura el controlador USB de manera que utilice el punto final cero para las transferencias de datos.

EJEMPLO 15-11

```

ENVIA    MACRO  DIREC,DATOS
        MOV     DX,301H
        MOV     AL,DIREC
        OUT    DX,AL

```

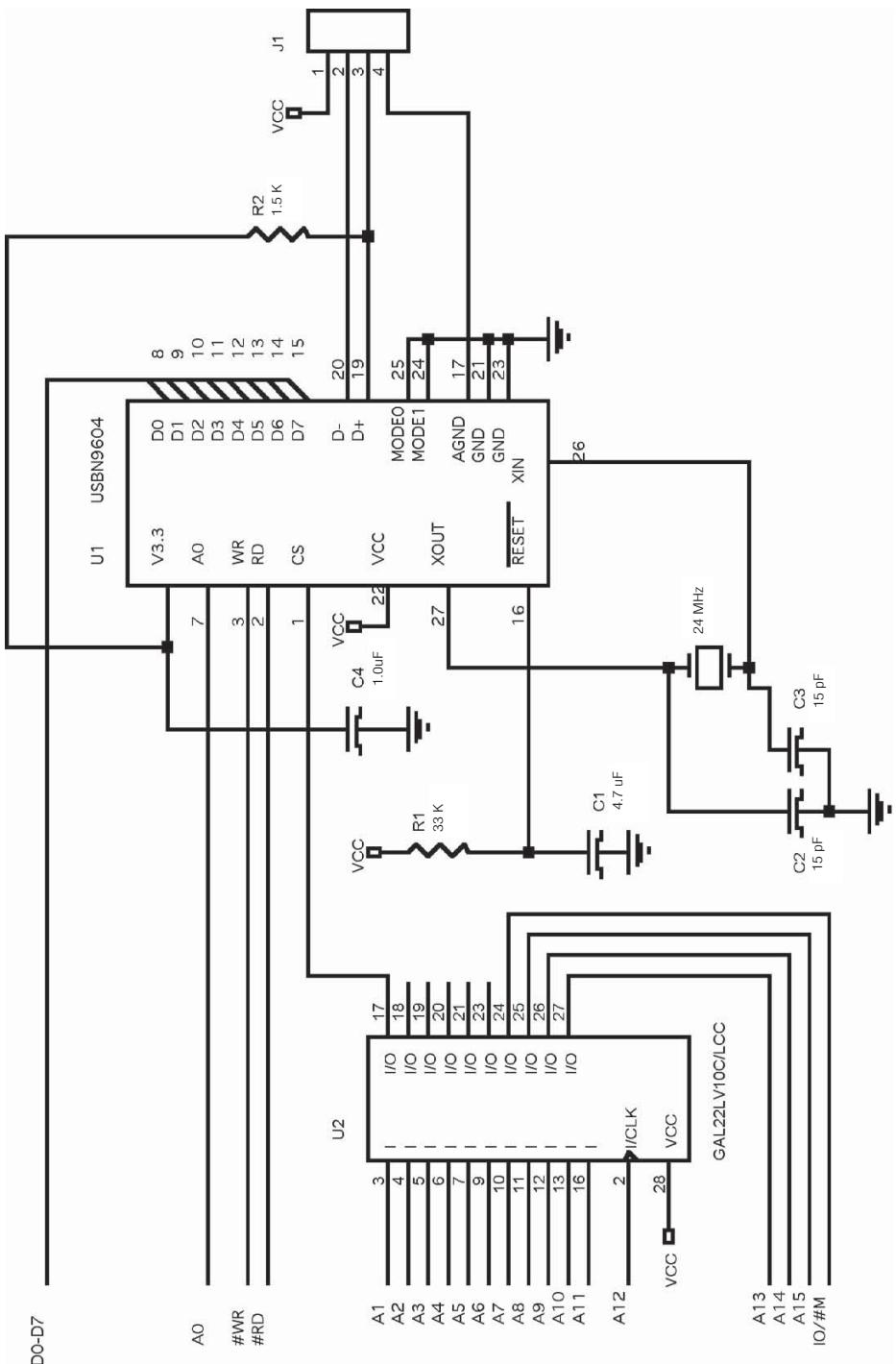


FIGURA 15-20 El USBN9604 conectado a un microprocesador en las direcciones de E/S 300H y 301H.

```

        MOV    DX, 300H
        MOV    AL, DATOS
        OUT   DX, AL
        ENDM

USBINT PROC NEAR

        SEND  0,5      ;interrupciones desactivadas, reinicio del USB por
                      ;software
        SEND  0,4      ;desactiva reinicio
        CALL  DELAY1   ;espera 1 ms
        SEND  9,40H    ;habilita comprobación de reinicio
        SEND  0DH,3    ;habilita EPO para recibir datos
        SEND  0BH,3    ;habilita EPO para transmitir datos
        SEND  20H,0    ;control EPO para dirección no predeterminada
        SEND  4,80H    ;establece FAR para aceptar dirección predeterminada
        SEND  0,8CH    ;USB está listo para enviar o recibir datos

USBINT ENDP

```

Una vez que se inicializa el controlador USB, es posible enviar o recibir datos hacia/desde el sistema anfitrión a través del USB. Para realizar la transmisión de datos se hace una llamada al procedimiento que se muestra en el ejemplo 15-12 para enviar un paquete de un byte mediante la FIFO TXD0. Este procedimiento utiliza la macro ENVIA que se muestra en el ejemplo 15-11 para transferir el byte en BL a través del USB y hacia el sistema anfitrión.

EJEMPLO 15-12

```

TRANS PROC NEAR

ENVIA 21H, BL          ;envía BL a FIFO
ENVIA 23H, 3           ;envía el byte

TRANS ENDP

```

Para recibir datos del USB se requieren dos funciones. Una verifica si hay datos disponibles y la otra lee un byte del USB, para colocarlo en el registro BL. Ambos procedimientos se listan en el ejemplo 15-13. El procedimiento ESTADO verifica si hay datos en la FIFO receptora. Si hay datos se activa el bit de acarreo al momento del retorno, pero si no se reciben datos se borra el bit de acarreo. El procedimiento LEES extrae un byte para la FIFO del receptor y lo devuelve en BL.

EJEMPLO 15-13

```

LEE     MACRO DIREC

        MOV    DX, 301H
        MOV    AL, DIREC
        OUT   DX, AL
        MOV    DX, 300H
        IN    AL, DX
        ENDM

ESTADO PROC NEAR
        ENVIA 6
        ENVIA 6
        SHL   AL, 2
        RET

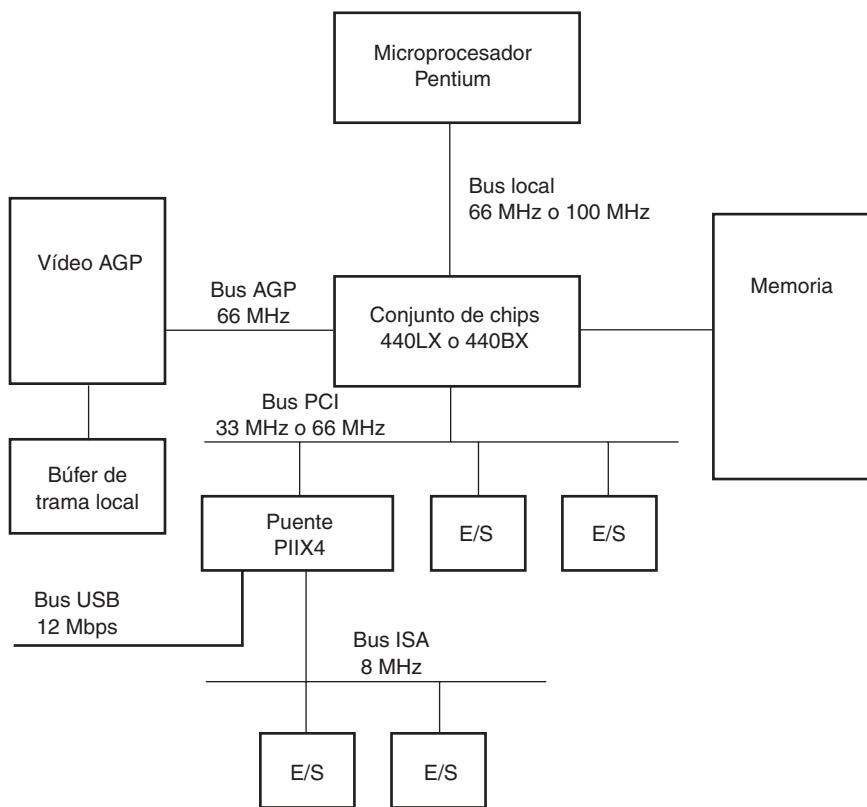
ESTADO ENDP

LEES    PROC NEAR
        LEE    25H
        RET

LEES    ENDP

```

FIGURA 15-21 Estructura de una computadora moderna, en la que se ilustran todos los buses.



15-6

EL PUERTO DE GRÁFICOS ACCELERADOS (AGP)

La más reciente adición en la mayoría de los sistemas computacionales es la inclusión del **puerto de gráficos acelerados** (AGP). El AGP opera a la frecuencia del reloj del bus del microprocesador. Fue diseñado de forma que una transferencia entre la tarjeta de vídeo y la memoria del sistema progresen a una velocidad máxima. El AGP puede transferir datos a una velocidad máxima de 2 Gbytes por segundo. Es probable que este puerto nunca se utilice para dispositivos que no sean tarjetas de vídeo, por lo que no le dedicaremos mucho espacio.

La figura 15-21 muestra la interfaz del AGP con un sistema Pentium 4 y la distribución de otros buses en el sistema. La principal ventaja del AGP sobre el bus PCI es que el AGP puede realizar transferencias (usando el sistema compatible con 8X) a velocidades de hasta 2 Gbytes por segundo. El sistema de 4X transfiere datos a velocidades de más de 1 Gbyte por segundo. El bus PCI tiene una velocidad de transferencia máxima aproximada de 133 Mbytes por segundo. El AGP fue diseñado de manera específica para permitir transferencias de alta velocidad entre el búfer de trama de la tarjeta de vídeo y la memoria del sistema, a través del conjunto de chips.

Es probable que el futuro traiga la desaparición del bus PCI y la incorporación del USB en el conjunto de chips. Tal vez hasta se incluya el conjunto de chips en el microprocesador. Hasta hoy, el sistema requiere el conjunto de chips 865 u 875 y el puente ISA, si se ordena como pedido especial para la tarjeta principal.

15-7

RESUMEN

- Los sistemas de bus (ISA, PCI y USB) permiten conectar los sistemas de E/S y de memoria a la computadora personal.

2. El bus ISA puede ser de ocho o de 16 bits, y soporta transferencias de memoria o de E/S a velocidades de 8 MHz.
3. El bus PCI (interconexión de componentes periféricos) soporta transferencias de 32 o de 64 bits entre la computadora personal y la memoria o la E/S a velocidades de 33 MHz. Este bus también permite conectar casi cualquier microprocesador con el bus PCI a través del uso de una interfaz de puente.
4. Una interfaz tipo “conectar y usar” (PnP) contiene una memoria que guarda la información de configuración para el sistema.
5. El puerto paralelo que se llama LPT₁ se utiliza para transferir datos de 8 bits en paralelo para las impresoras y otros dispositivos.
6. Los puertos COM seriales se utilizan para la transferencia de datos en serie. La API de Windows se utiliza en la aplicación Windows Visual C++ para efectuar la transferencia de datos en serie a través de los puertos COM.
7. El bus serial universal (USB) ha sustituido casi por completo al bus ISA en los sistemas más avanzados. El USB tiene tres velocidades de transferencia de datos: 1.5 Mbps, 12 Mbps y 480 Mbps.
8. El USB utiliza el sistema NRZI para codificar datos y utiliza el relleno de bits para las transmisiones de 1s lógicos que sean de más de seis bits.
9. El puerto de gráficos acelerados (AGP) es una conexión de alta velocidad entre el sistema de memoria y la tarjeta de gráficos de vídeo.

15-8**PREGUNTAS Y PROBLEMAS**

1. ¿A qué frase corresponden las letras ISA como acrónimo?
2. ¿Qué tamaños de transferencias de datos soporta el sistema de bus ISA?
3. ¿Se utiliza la interfaz de bus ISA con frecuencia para la expansión de memoria?
4. Desarrolle una interfaz de bus ISA que se decodifique en las direcciones 310H-313H. Esta interfaz deberá contener un 8255 al cual se acceda mediante estas direcciones de puerto. (No olvide usar búfer con todas las entradas para la tarjeta de bus ISA.)
5. Desarrolle una interfaz de bus ISA que decodifique los puertos 0340H-0343H para controlar un solo temporizador 8254.
6. Desarrolle una interfaz de bus PCI de 32 bits que agregue una EPROM 27C256 en las direcciones de memoria FFFF0000H-FFF7FFFH.
7. Suponiendo que cuenta con un búfer 74LS244 y un enclavamiento 74LS374, desarrolle una interfaz de bus ISA que contenga un puerto de entrada de 8 bits en la dirección de E/S 308H y un puerto de salida de 8 bits en la dirección de E/S 30AH.
8. Cree una interfaz de bus ISA que permita cuatro canales de señales de salida analógicas, de 0 a 5.0 V cada una. Estos cuatro canales deberán decodificarse en las direcciones de E/S 300H, 310H, 320H y 330H. También desarrolle software que soporte los cuatro canales.
9. Haga de nuevo la pregunta 8, sólo que en vez de cuatro canales de salida utilice cuatro ADCs para crear cuatro canales de entrada analógicos en las mismas direcciones.
10. Mediante el uso de un temporizador o de varios temporizadores 8254, desarrolle un temporizador de cuarto oscuro en una tarjeta de bus ISA. Su temporizador deberá generar un 0 lógico para intervalos de 1/100 segundos, desde 1/100 segundos hasta cinco minutos. Use el reloj del sistema de 8 MHz como fuente de sincronización. El software que desarrolle deberá permitir al usuario seleccionar el tiempo desde el teclado. La señal de salida del temporizador deberá ser un 0 lógico durante el tiempo seleccionado y pasar a través de un inversor para habilitar un relevador de estado sólido que controle el amplificador fotográfico.
11. Conecte un UART 16550 a la computadora personal a través de la interfaz de bus PCI. Desarrolle software para transmitir y recibir datos a velocidades de 300, 1200, 9600 y 19,200 baudios. El UART deberá responder a los puertos de E/S 1E3XH.
12. El bus ISA puede transferir datos de _____ bits a la velocidad de 8 MHz.
13. Describa cómo puede capturarse la dirección desde el bus PCI.

14. ¿Cuál es el propósito de la memoria de configuración que se encuentra en la interfaz de bus PCI?
15. Defina el término “conectar y usar”.
16. ¿Cuál es el propósito de la conexión C/ \overline{BE} en el sistema de bus PCI?
17. ¿Cómo se comprueba el BIOS para la extensión PCI del BIOS?
18. Desarrolle un programa corto que interrogué al bus PCI mediante el uso de la extensión para el BIOS, y que lea el contenido de 32 bits del registro de configuración 08H. Para este problema, considere que los números del bus y de la unidad son 0000H.
19. ¿Qué ventaja ofrece el bus PCI en comparación con el bus ISA?
20. ¿Qué tan rápido puede transferir el bus PCI Express datos en serie?
21. ¿En qué direcciones de E/S se decodifica el puerto paralelo en una computadora personal?
22. ¿Pueden leerse datos desde el puerto paralelo?
23. El conector de puerto paralelo que se encuentra en la parte posterior de la computadora tiene _____ terminales.
24. La mayoría de las computadoras contienen cuando menos un puerto de comunicaciones en serie. ¿Cómo se llama este puerto?
25. Desarrolle una función en C++ que envíe las letras ABC a través del puerto serial y continúe haciéndolo hasta que se devuelvan las letras ABC a través del mismo puerto serial. Muestre todas las funciones necesarias para lograr esto, incluyendo cualquier inicialización.
26. Modifique el ejemplo 15-9 de manera que envíe una cadena de caracteres de cualquier longitud.
27. Busque en Internet y explique con detalles (en un informe corto) cómo se utilizan las variantes en el entorno de programación Visual.
28. ¿Qué velocidades de datos se pueden usar con el USB?
29. ¿Cómo se codifican los datos en el USB?
30. ¿Cuál es la máxima longitud de cable que se puede usar con el USB?
31. ¿Sustituirá alguna vez el USB al bus ISA?
32. ¿Cuántas direcciones de dispositivos hay disponibles en el USB?
33. ¿Qué es la codificación NRZI?
34. ¿Qué es un bit relleno?
35. Si se envían los siguientes datos puros en el USB, dibuje la forma de onda de la señal que se encuentra en el USB: (1100110000110011011010).
36. ¿Qué tan extenso puede ser un paquete de datos en el USB?
37. ¿Cuál es el propósito de los tokens NAK y ACK en el USB?
38. Describa la diferencia en las velocidades de transferencia de datos en el bus PCI, en comparación con el AGP.
39. ¿Cuál es la velocidad de transferencia de datos en un sistema que utiliza una tarjeta de video AGP 8X?
40. Localice unos cuantos fabricantes de tarjetas de vídeo en Internet y averigüe cuánta memoria hay disponible en las tarjetas de vídeo AGP. Liste los fabricantes y la cantidad de memoria en las tarjetas.
41. Utilice Internet para escribir un informe en el que proporcione los detalles sobre cualquier controlador USB.

CAPÍTULO 16

Los microprocesadores 80186, 80188 y 80286

INTRODUCCIÓN

Los microprocesadores Intel 80186/80188 y 80286 son versiones mejoradas de la familia 80X86. El 80186/80188 y el 80286 son microprocesadores de 16 bits que tienen compatibilidad ascendente con el 8086/8088. Incluso, el hardware de estos microprocesadores es similar al que traen las versiones anteriores. En este capítulo presentaremos una descripción general de cada microprocesador, y señalaremos las diferencias o mejoras en cada versión. En la primera parte describiremos los microprocesadores 80186/80188; en la última, mostraremos el microprocesador 80286.

Una novedad en las ediciones recientes de este libro es la cobertura de la familia 80186/80188. Intel ha agregado cuatro nuevas versiones de cada uno de estos controladores integrados a su línea de microprocesadores. Cada uno es una versión CMOS y se identifica con un sufijo de dos letras: XL, EA, EB y EC. Los modelos 80C186XL y 80C188XL son similares a los modelos 80186/80188 anteriores.

OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

1. Describir las mejoras de hardware y software de los microprocesadores 80186/80188 y 80286, en comparación con los microprocesadores 8086/8088.
2. Detallar las diferencias entre las diversas versiones de los controladores integrados 80186 y 80188.
3. Conectar el 80186/80188 y el 80286 a la memoria y la E/S.
4. Desarrollar software que utilice las mejoras que se proporcionan en estos microprocesadores.
5. Describir la operación de la unidad de administración de memoria (MMU) dentro del microprocesador 80286.
6. Definir y detallar la operación de un sistema operativo en tiempo real (RTOS).

16-1

ARQUITECTURA 80186/80188

Los microprocesadores 8086 y 8088 y los microprocesadores 80186 y 80188 son casi idénticos. La única diferencia entre el 80186 y el 80188 es la anchura de sus buses de datos. El 80186 (al igual que el 8086) contiene un bus de datos de 16 bits, mientras que el 80188 (al igual que el 8088) trae un bus de datos de 8 bits. La estructura de registros interna del 80186/80188 es casi idéntica a la del 8086/8088. La única diferencia conocida es que el 80186/80188 contiene vectores de interrupción reservados adicionales y algunas características de E/S integradas muy poderosas. Por lo general, el 80186 y el 80188

se conocen como **controladores integrados** debido a su aplicación como controlador, no como computadora basada en microprocesador.

Versiones del 80186/80188

Como ya se mencionó, el 80186 y el 80188 están disponibles en cuatro versiones distintas; las cuatro son microprocesadores CMOS. La tabla 16-1 muestra cada versión y las principales características que se proporcionan. El 80C186XL y el 80C188XL son las versiones más básicas del 80186/80188; el 80C186EC y el 80C188EC son los más avanzados. En este libro hablaremos con detalle sobre el 80C186XL/80C188XL, después describiremos las características y mejoras adicionales que se proporcionan en las otras versiones.

Diagrama de bloques básico del 80186

La figura 16-1 proporciona el diagrama de bloques del microprocesador 80188, el cual representa en forma genérica a todas las versiones, con la excepción de las mejoras y las características adicionales que se describen en la tabla 16-1. Este microprocesador tiene muchos más circuitos internos que el 8088. Los diagramas de bloques del 80186 y del 80188 son idénticos, excepto por la cola de búsqueda previa (*prefetch*), que es de cuatro bytes en el 80188 y de seis bytes en el 80186. Al igual que el 8088, el 80188 contiene una unidad de interfaz de bus (BIU) y una unidad de ejecución (ED).

Además de la BIU y la ED, la familia 80186/80188 contiene un generador de reloj, un controlador de interrupciones programable, temporizadores programables, un controlador de DMA programable y una unidad de selección de chip programable. Estas mejoras aumentan en forma considerable la utilidad del 80186/80188 y reducen el número de componentes periféricos requeridos para implementar un sistema. Muchos subsistemas populares para la computadora personal utilizan los microprocesadores

TABLA 16-1 Las cuatro versiones del controlador integrado 80186/80188.

Característica	80C186XL 80C188XL	80C186EA 80C188EA	80C186EB 80C188EB	80C186EC 80C188EC
Conjunto de instrucciones similar al 80286	✓	✓	✓	✓
Ahorro de energía (modo verde)	✓	✓		✓
Modo de apagado		✓	✓	✓
Interfaz con el 80C187	✓	✓	✓	✓
Modo único	✓	✓	✓	✓
Controlador de interrupciones	✓	✓	✓	✓ similar al 8259
Unidad de temporizador	✓	✓	✓	✓
Unidad de selección de chip	✓	✓	✓ mejorada	✓ mejorada
Controlador de DMA	✓ 2 canales	✓ 2 canales		✓ 4 canales
Unidad de comunicaciones en serie			✓	✓
Controlador de refresco	✓	✓	✓ mejorado	✓ mejorado
Temporizador Watchdog				✓
Puertos de E/S			✓ 16 bits	✓ 22 bits

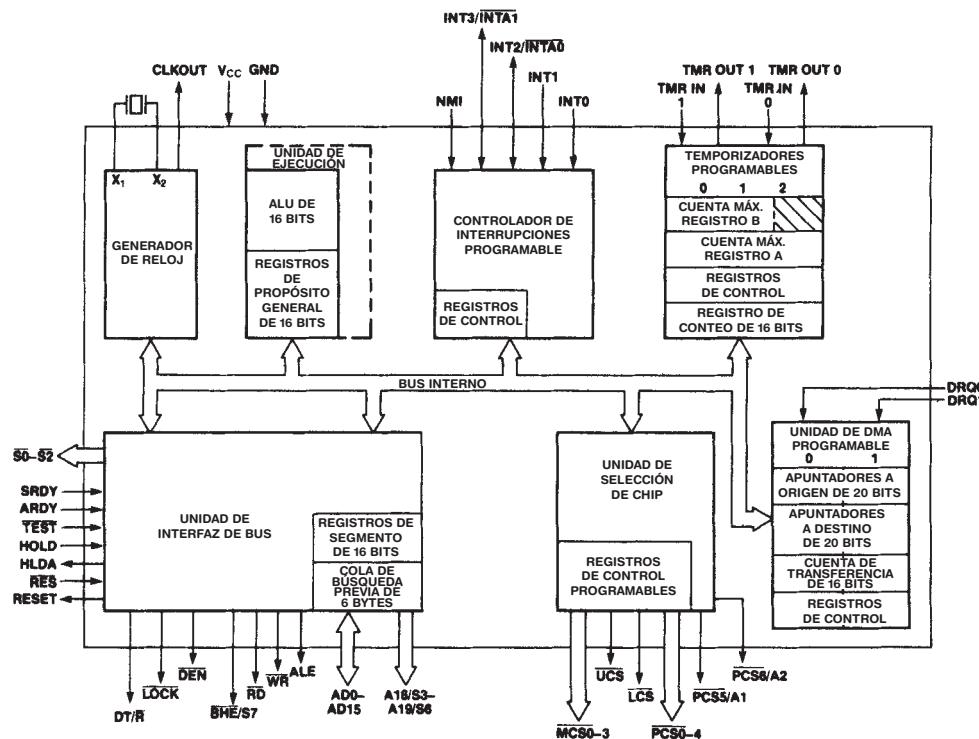


FIGURA 16-1 Diagrama de bloques del microprocesador 80186. El diagrama de bloques del 80188 es idéntico, sólo falta BHE/S7 y las terminales AD₁₅-AD₈ están identificadas como A₁₅-A₈. (Cortesía de Intel Corporation.)

80186/80188 como controladores de disco de caché, controladores de red de área local (LAN), etcétera. El 80186/80188 también tiene aplicación en la red de telefonía celular como commutador.

El software para el 80186/80188 es idéntico al destinado para el microprocesador 80286, pero sin las instrucciones de administración de memoria. Esto significa que las instrucciones similares a las del 80286 para multiplicación inmediata, conteos de desplazamiento inmediato, E/S de cadenas de caracteres, PUSH, POP, BOUND, ENTER y LEAVE funcionan en los microprocesadores 80186/80188.

Características básicas del 80186/80188

En este segmento del libro presentaremos las mejoras de los microprocesadores o controladores integrados 80186/80188, que se aplican a todas las versiones, excepto en donde se indica, aunque no proporcionaremos una cobertura exclusiva. Más adelante proporcionaremos más detalles sobre la operación de cada una de las mejoras y los detalles sobre cada versión avanzada.

Generador de reloj. El generador de reloj interno sustituye al generador de reloj 8284A que se utiliza con los microprocesadores 8086/8088. Esto reduce la cuenta de componentes en un sistema.

El generador de reloj interno tiene tres conexiones terminales: X₁, X₂ y CLKOUT (o, en algunas versiones: CLKIN, OSCOUT y CLKOUT). Las terminales X₁ (CLKIN) y X₂ (OSCOUT) se conectan a un cristal que resuena al doble de la frecuencia de operación del microprocesador. En la versión de 8 MHz del 80186/80188 se conecta un cristal de 16 MHz a X₁ (CLKIN) y a X₂ (OSCOUT). El 80186/80188 está disponible en versiones de 6 MHz, 8 MHz, 12 MHz, 16 MHz o 25 MHz.

La terminal CLKOUT proporciona una señal de reloj de sistema cuya frecuencia es equivalente a la mitad de la frecuencia del cristal, con un ciclo de trabajo del 50%. La terminal CLKOUT controla otros dispositivos en un sistema y proporciona una fuente de sincronización para microprocesadores adicionales en el sistema.

Además de estas terminales externas, el generador de reloj proporciona la temporización interna para sincronizar la terminal de entrada READY, mientras que en el sistema 8086/8088 el generador de reloj 8284A es el que proporciona la sincronización de READY.

Controlador de interrupciones programable. El controlador de interrupciones programable (PIC) controla las interrupciones internas y externas, así como hasta dos PICs 8259A externos. Cuando se conecta un 8259 externo, los microprocesadores 80186/80188 funcionan como maestro y los 8259 funcionan como esclavo. Los modelos 80C186EC y 80C188EC contienen un controlador de interrupciones compatible con el 8259A, en vez del que se describe aquí para las otras versiones (XL, EA y EB).

Si el PIC se opera sin el 8259 externo, tiene cinco entradas de interrupción: INT0 a INT3 y NMI. El número de interrupciones disponibles depende de la versión: la versión EB tiene seis entradas de interrupción y la versión EC tiene 16. Ésta es una expansión de las dos entradas de interrupción disponibles en los microprocesadores 8086/8088. En muchos sistemas las cinco entradas de interrupción son adecuadas.

Temporizadores. La sección de temporizadores contiene tres temporizadores de 16 bits 100% programables. Los temporizadores 0 y 1 generan formas de onda para uso externo y se controlan mediante el reloj maestro del 80186/80188, o mediante un reloj externo. También se utilizan para contar eventos externos. El tercer temporizador (2) es interno y se controla mediante el reloj maestro. La salida del temporizador 2 genera una interrupción después de cierto número específico de ciclos de reloj y llega a proporcionar un reloj a los otros temporizadores. El temporizador 2 también puede utilizarse como temporizador *watchdog*, ya que es posible programarlo para interrumpir el microprocesador después de cierta cantidad de tiempo.

Los modelos 80C186EC y 80C188EC tienen un temporizador adicional, al cual se le conoce como *watchdog* (perro vigía). Tal temporizador es un contador de 32 bits que se controla en forma interna mediante la señal CLKOUT (la mitad de la frecuencia del cristal). Cada vez que el contador llega a cero, se vuelve a cargar y genera un pulso en la terminal WDTOUT, equivalente a cuatro períodos CLKOUT. Dicha salida puede usarse para cualquier propósito: puede conectarse a la entrada de reinicio para producir un reinicio o a la entrada NMI para producir una interrupción. Si se conecta a las entradas de reinicio o NMI, se reprograma en forma periódica, de manera que su conteo nunca llegue a cero. El propósito de un temporizador *watchdog* es reiniciar o interrumpir el sistema, en caso de que el software empiece a fallar.

Unidad de DMA programable. Esta unidad contiene dos canales de DMA o cuatro canales de DMA en los modelos 80C186EC/80C188EC. Cada canal puede transferir datos entre posiciones de memoria, entre la memoria y un dispositivo de E/S o entre dispositivos de E/S. Este controlador de DMA es similar al controlador de DMA 8237 que vimos en el capítulo 13. La principal diferencia es que el controlador de DMA 8237 tiene cuatro canales de DMA, igual que el modelo EC.

Unidad de selección de chip programable. La unidad de selección de chip es una memoria programable y un decodificador de E/S integrados. Tiene seis líneas de salida para seleccionar memoria, siete líneas para seleccionar la E/S en los modelos XL y EA, así como 10 líneas para seleccionar memoria o E/S en los modelos EB y EC.

En los modelos XL y EA, las líneas de selección de memoria se dividen en tres grupos que seleccionan memoria para las principales secciones del mapa de memoria del 80186/80188. La señal de selección de memoria inferior habilita la memoria para los vectores de interrupción, la señal de selección de memoria superior habilita hasta cuatro dispositivos de memoria medios. El límite de la memoria inferior comienza en la posición 00000H y el límite de la memoria superior termina en la posición FFFFFH. Los tamaños de las áreas de memoria son programables y llegan a insertarse estados de espera (0-3 estados) de manera automática con la selección de un área de memoria.

En los modelos XL y EA, cada señal de selección de E/S programable direcciona un bloque de 128 bytes de espacio de E/S. El área de E/S programable empieza en una dirección de E/S base programada por el usuario, en tanto que los siete bloques de 128 bytes son contiguos.

En los modelos EB y EC hay una terminal de selección de chip de memoria superior e inferior, así como ocho terminales de selección de memoria o de chip de E/S de propósito general. Otra diferencia es que pueden programarse de 0 a 15 estados de espera en estas dos versiones de los controladores integrados 80186/80188.

Característica de ahorro de energía/apagado. La característica de ahorro de energía permite dividir el reloj del sistema entre 4, 8 o 16 para reducir el consumo de energía. La característica de ahorro de energía se inicia mediante software y termina con un evento de hardware tal como una interrupción. La característica de apagado detiene el reloj por completo, pero no está disponible en la versión XL. El modo de apagado se activa mediante la ejecución de una instrucción HLT y se desactiva mediante una interrupción.

Unidad de control de refresco. Esta unidad genera la dirección de fila de refresco en el intervalo programado. La unidad de control de refresco no multiplexea la dirección para la DRAM, lo cual sigue siendo responsabilidad del diseñador del sistema. La dirección de refresco se proporciona al sistema de memoria al final del intervalo de refresco programado, junto con la señal de control \overline{RFSH} . El sistema de memoria debe efectuar un ciclo de refresco durante el tiempo activo de la señal de control \overline{RFSH} . En la sección que explica el funcionamiento de la unidad de selección de chip veremos más sobre la memoria y el refresco.

Diagrama de terminales

La figura 16-2 muestra el diagrama de terminales del microprocesador 80C186XL. Este microprocesador viene empaquetado en un portador de chip sin plomo (LCC) de 68 terminales o en una matriz de rejilla de terminales (PGA). En la figura 16-3 se muestran los paquetes LCC y PGA.

Definiciones de terminales. La siguiente lista define cada una de las terminales del 80C186XL y señala cualquier diferencia entre los microprocesadores 80C186XL y 80C188XL. Más adelante, en este capítulo, veremos las versiones mejoradas.

V_{CC}	Ésta es la conexión de la fuente de energía del sistema para $\pm 10\%$, +5.0 V.
V_{SS}	Ésta es la conexión a la tierra del sistema.
X₁ y X₂	Las terminales de reloj se conectan por lo general a un cristal resonante paralelo de modo fundamental, el cual opera un oscilador de cristal interno. Puede conectarse una señal externa de reloj a la terminal X ₁ . El reloj maestro interno opera a la mitad de la señal de entrada de reloj o del cristal interno. En algunas versiones del 80186/80188, estas etiquetas se identifican como CLKIN (X ₁) y OSCOUT(X ₂).

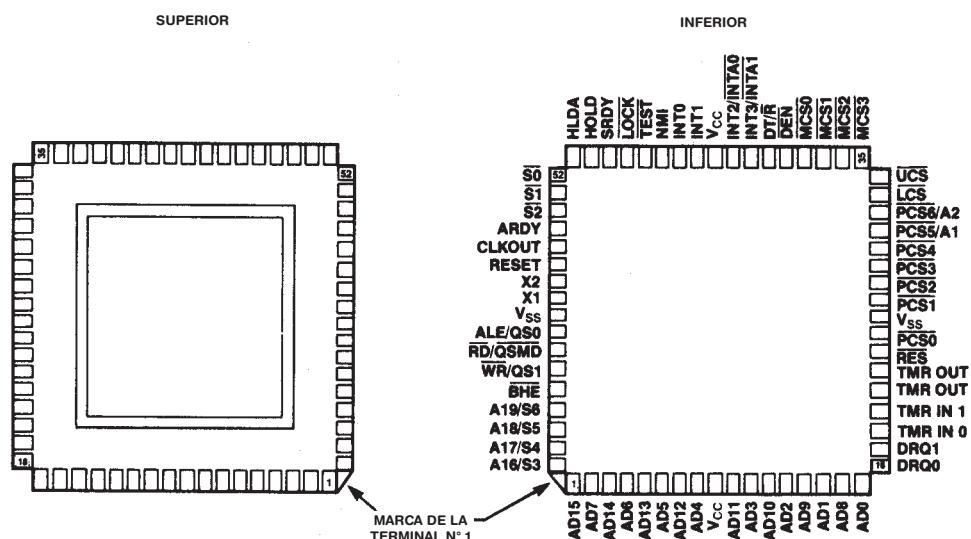
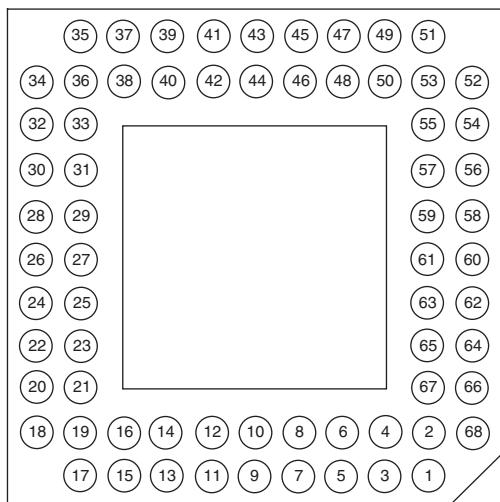
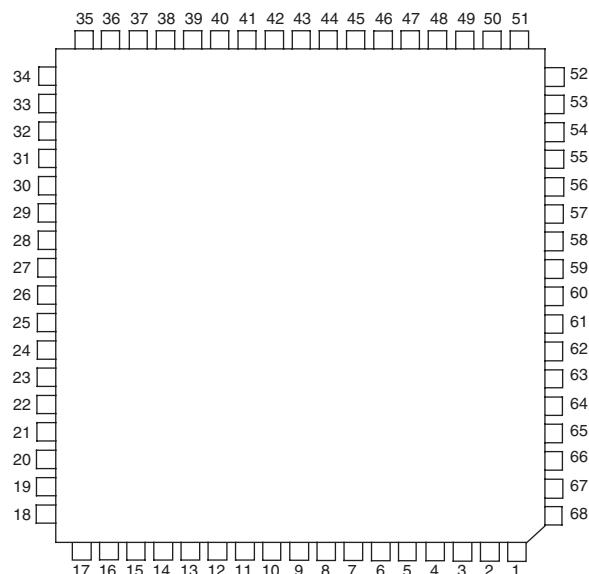


FIGURA 16-2 Diagrama de terminales del microprocesador 80186.
(Cortesía de Intel Corporation.)

Vista inferior PGA



Vista inferior LCC

**FIGURA 16-3** Vistas inferiores de las versiones estilo PGA y LCC del microprocesador 80C188XL.**CLKOUT**

La salida de reloj proporciona una **señal de sincronización** para los periféricos del sistema, a una frecuencia equivalente a la mitad de la frecuencia de entrada de reloj con un ciclo de trabajo del 50%.

RES

La terminal **entrada de reinicio** reinicia el 80186/80188. Para un reinicio apropiado, RES debe mantenerse en nivel bajo durante por lo menos 50 ms después de aplicar energía. Esta terminal se conecta a menudo con un circuito RC que genera una señal de reinicio después de que se aplica la energía. La posición de reinicio es idéntica a la del microprocesador 8086/8088: FFFF0H.

RESET

La terminal **salida de reinicio complementaria** (se va a nivel alto durante un reinicio) se conecta a los periféricos del sistema para inicializarlos cada vez que la entrada RES se va al nivel bajo.

TEST

Esta terminal de prueba se conecta a la salida BUSY del coprocesador numérico 80187. La terminal TEST se interroga con la instrucción FWAIT o WAIT.

T_{in0} y T_{in1}

Estas terminales se utilizan como **fuentes de reloj externas** para los temporizadores 0 y 1.

T_{out0} y T_{out1}

Estas terminales proporcionan las **señales de salida** de los temporizadores 0 y 1, que pueden programarse para proporcionar ondas cuadradas o pulsos.

DRQ0 y DRQ1

Estas terminales son líneas de **p petición de DMA** que disparan y están activas en nivel alto para los canales de DMA 0 y 1.

NMI

Ésta es una entrada de **interrupción no enmascarable**. Se dispara con el borde positivo y siempre está activa. Cuando se activa NMI, utiliza el vector de interrupción 2.

**INT₀, INT₁, INT₂/
INTA₀ e INT₃/INTA₁**

Son **entradas de interrupción enmascarable**. Están activas a nivel alto y se programan para dispararse por nivel o por borde. Tales terminales se configuran como cuatro entradas de interrupción si no hay un 8259 externo presente, o como dos entradas de interrupción, si está presente el controlador de interrupciones 8259A.

A₁₉/ONCE, A₁₈, A₁₇ y A₁₆	Éstas son conexiones de dirección/estado multiplexadas que proporcionan la dirección (A ₁₉ -A ₁₆) y el estado (S ₆ -S ₃). Los bits de estado que se encuentran en las terminales de dirección A ₁₈ -A ₁₆ no tienen función del sistema y se utilizan durante la <u>fabricación</u> para realizar pruebas. La terminal A ₁₉ es una entrada para la función ONCE en un reinicio. Si ONCE se mantiene en nivel bajo durante un reinicio, el microprocesador entra a un modo de prueba.
AD₁₅-AD₀	Éstas son conexiones de bus de dirección/datos multiplexadas . Durante T ₁ , el 80186 coloca A ₁₅ -A ₀ en estas terminales; durante T ₂ , T ₃ y T ₄ , el 80186 utiliza estas terminales como bus de datos para las señales D ₁₅ -D ₀ . Tenga en cuenta que el 80188 tiene las terminales AD ₇ -AD ₀ y A ₁₅ -A ₈ .
BHE	La terminal habilitación de bus superior indica (cuando es un 0 lógico) que se van a transferir datos válidos a través de las conexiones D ₁₅ -D ₈ del bus de datos.
ALE	Habilitación de enclavamiento de dirección es una terminal de salida que contiene la ALE medio ciclo de reloj antes que en el 8086. Se utiliza para demultiplexar los buses de dirección/datos y dirección/estado. (Aún y cuando los bits de estado en A ₁₉ -A ₁₆ no se utilizan en el sistema, de todas formas deben demultiplexarse.)
WR	La terminal escritura hace que se escriban datos en la memoria o E/S.
RD	La terminal lectura hace que se lean datos de la memoria o E/S.
ARDY	La entrada READY asíncrona informa al 80186/80188 que la memoria o la E/S están listas para que el 80186/80188 lea o escriba datos. Si esta terminal se conecta de manera permanente a +5.0 V, el microprocesador funciona en forma normal; si se aterraza, el microprocesador entra a estados de espera.
SRDY	La entrada READY sincrónica se sincroniza con el reloj del sistema para proporcionar una sincronización relajada para la entrada READY. Al igual que ARDY, SRDY se conecta de manera permanente a +5.0 V para que no haya estados de espera.
LOCK	La terminal bloqueo es una salida que se controla mediante el prefijo LOCK. Si una instrucción contiene este prefijo, la terminal LOCK se convierte en un 0 lógico durante la instrucción bloqueada.
S₂, S₁ y S₀	Éstos son bits de estado que proporcionan al sistema el tipo de transferencia de bus que se lleva a cabo. En la tabla 16-2 podrán consultarse los estados de los bits de estado. La terminal selección de chip de memoria superior selecciona la memoria que está en la porción superior del mapa de memoria.
UCS	La salida selección de chip de memoria superior puede programarse para permitir tamaños de memoria de 1 K-256 Kbytes, los cuales terminan en la posición FFFFH. La terminal se programa de manera distinta en las versiones EB y EC; además, habilita la memoria entre 1 K y 1 M.

TABLA 16-2 Los bits de estado S₂, S₁ y S₀.

S ₂	S ₁	S ₀	Función
0	0	0	Reconocimiento de interrupción
0	0	1	Lectura de E/S
0	1	0	Escritura de E/S
0	1	1	Alto
1	0	0	Búsqueda de código de operación
1	0	1	Lectura de memoria
1	1	0	Escritura en memoria
1	1	1	Pasivo

LCS	La terminal selección de chip de memoria inferior habilita la memoria que empieza en la posición 00000H. La terminal fue programada para seleccionar tamaños de memoria desde 1 K hasta 256 Kbytes. Esta terminal funciona de manera distinta para las versiones EB y EC, además de que habilita la memoria entre 1 K y 1 Mbytes.
MCS0-MCS3	Las terminales selección de chip de memoria intermedia habilitan cuatro dispositivos de memoria intermedia. Estas terminales se pueden programar para seleccionar un bloque de memoria de 8 K-512 Kbytes, que contiene cuatro dispositivos. Las terminales no están presentes en las versiones EB y EC.
PCS0-PCS4	Éstas son cinco líneas de selección de periférico distintas. Las líneas no están presentes en las versiones EB y EC.
PCS5/A₁ y PCS6/A₂	Éstas se programan como líneas de selección de periférico o como los bits de dirección A ₂ y A ₁ con enclavamiento interno. Las líneas no están presentes en las versiones EB y EC.
DT/R	La terminal transmisión/recepción de datos controla la dirección de los búferes del bus de datos si se conecta al sistema.
DEN	La terminal habilitación del bus de datos habilita los búferes externos del bus de datos.

Características de operación en DC

Es necesario conocer las características de operación en DC antes de tratar de conectar o de operar el microprocesador. Los microprocesadores 80C186/80C188 requieren entre 42 mA y 63 mA de corriente de la fuente de energía. Cada terminal de salida proporciona 3.0 mA de corriente para el nivel de 0 lógico y -2 mA de corriente para el nivel 1 lógico.

Sincronización del 80186/80188

En la figura 16-4 se muestra el diagrama de sincronización para el 80186. La sincronización para el 80188 es idéntica excepto por las conexiones de dirección multiplexadas, que son AD₇-AD₀, en vez de AD₁₅-AD₀, y por BHE, que no existe en el 80188.

La sincronización básica para el 80186/80188 está compuesta por cuatro períodos de reloj, de igual forma que en el 8086/8088. Un ciclo de bus para la versión de 8 MHz requiere 500 ns, mientras la versión de 16 MHz requiere 250 ns.

Hay muy pocas diferencias entre la sincronización para el 80186/80188 y el 8086/8088. La diferencia más notable es que ALE aparece medio ciclo de reloj antes en el 80186/80188.

Tiempo de acceso a memoria. Uno de los puntos más importantes en el diagrama de sincronización de cualquier microprocesador es el tiempo de acceso a memoria. Los cálculos del tiempo de acceso para el 80186/80188 son idénticos que para el 8086/8088. Recuerde que el tiempo de acceso es el que se asigna a la memoria y a la E/S, para que proporcionen datos al microprocesador, una vez que éste envía su dirección a la memoria o a la E/S.

Un análisis detallado del diagrama de sincronización revela que la dirección aparece en el bus de dirección T_{CLAV} después del inicio de T₁. T_{CLAV} se lista como en 44 ns para la versión de 8 MHz (vea la figura 16-5). Los datos se muestran desde el bus de datos al final de T₃, pero se requiere un tiempo de preparación antes de que el reloj se defina como T_{DVCL}. Los tiempos que se listan para T_{DVCL} son de 20 ns para ambas versiones del microprocesador. El tiempo de acceso es igual a tres períodos de reloj menos T_{CLAV} y T_{DVCL}. El tiempo de acceso para el microprocesador de 8 MHz es de 375 ns - 4 ns - 20 ns, o 311 ns. El tiempo de acceso para la versión de 16 MHz se calcula de la misma forma, sólo que T_{CLAV} es de 25 ns y T_{DVCL} de 15 ns.

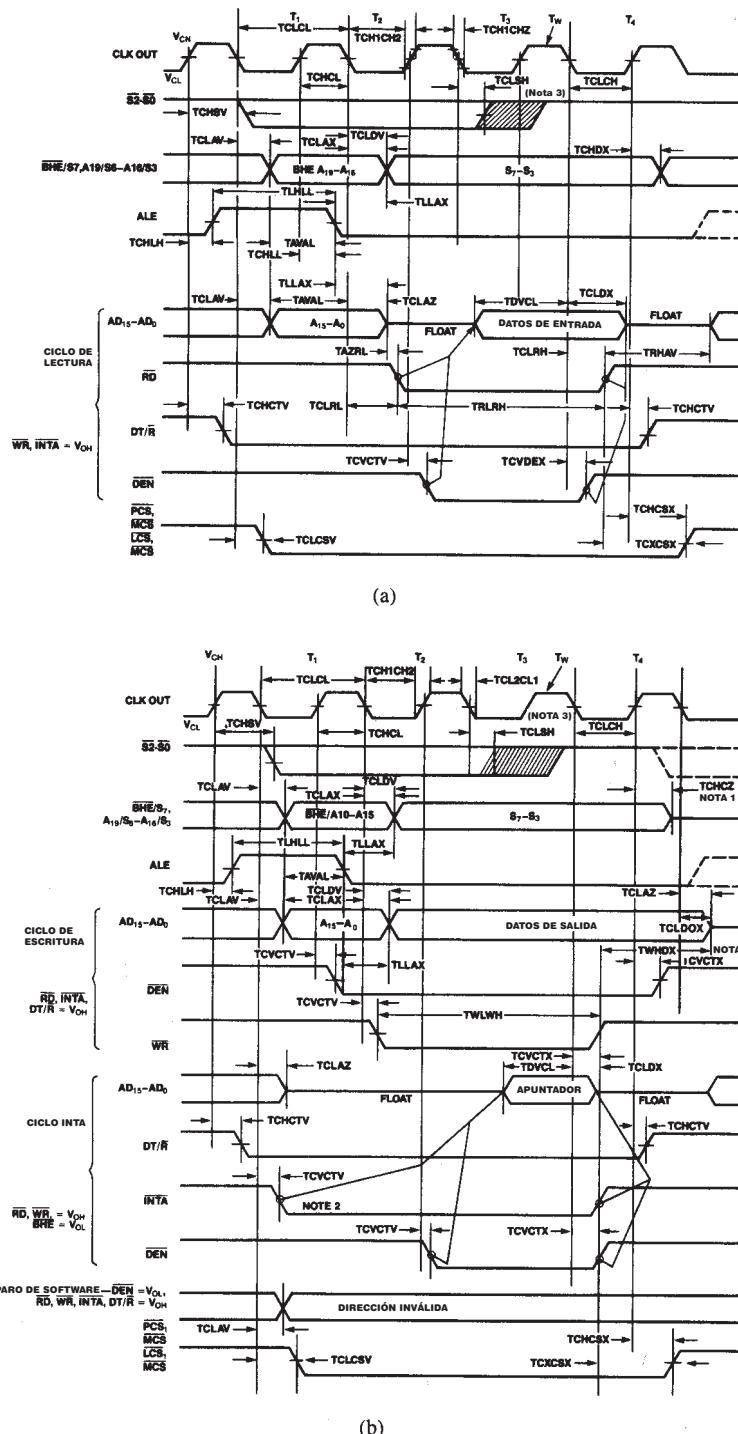


FIGURA 16-4 Sincronización del 80186/80188. (a) Sincronización del ciclo de lectura y (b) sincronización del ciclo de escritura. (Cortesía de Intel Corporation.)

Respuestas de sincronización de la interfaz del maestro 80186

Símbolo	Parámetros	80188 (8 MHz)		80188-6 (6 MHz)		Unidades	Condiciones de prueba
		Mín.	Máx.	Mín.	Máx.		
T _{CLAV}	Retraso de dirección válida	5	44	5	63	ns	C _L = 20-200 pF todas las salidas
T _{CLAX}	Retención de dirección	10		10		ns	
T _{CLAZ}	Retraso por flotación de dirección	T _{CLAX}	35	T _{CLAX}	44	ns	
T _{CHCZ}	Retraso por flotación de líneas de comando		45		56	ns	
T _{CHCV}	Retraso líneas de dirección válidas (después de flotación)		55		76	ns	
T _{LHLL}	Anchura ALE	T _{CLCL-35}		T _{CLCL-35}		ns	
T _{CHLH}	Retraso ALE activa		35		44	ns	
T _{CHLL}	Retraso ALE inactiva		35		44	ns	
T _{LLAX}	Retención de dirección para ALE inactiva	T _{CHCL-25}		T _{CHCL-30}		ns	
T _{CLDV}	Retraso de dirección válida	10	44	10	55	ns	
T _{CLDOX}	Tiempo de retención de datos	10		10		ns	
T _{WHDX}	Retención de datos después de WR	T _{CLCL-40}		T _{CLCL-50}		ns	
T _{CVCTV}	Retraso de control activo 1	5	70	5	87	ns	
T _{CHCTV}	Retraso control activo 2	10	55	10	76	ns	
T _{CVCTX}	Retraso control inactivo	5	55	5	76	ns	
T _{CVDEX}	Retraso DEN inactivo (ciclo de no escritura)		70		87	ns	
T _{AZRL}	Flotación de dirección para RD activa	0		0		ns	
T _{CLRL}	Retraso DL activa	10	70	10	87	ns	
T _{CLRH}	Retraso DL inactiva	10	55	10	76	ns	
T _{RHAV}	RD inactiva a dirección activa	T _{CLCL-40}		T _{CLCL-50}		ns	
T _{CLHAV}	Retraso válido HLDA	10	50	10	67	ns	
T _{RLRH}	Anchura RD	2T _{CLCL-50}		2T _{CLCL-50}		ns	
T _{WLWH}	Anchura WR	2T _{CLCL-40}		2T _{CLCL-40}		ns	
T _{AVAL}	Dirección válida para ALE baja	T _{CLCH-25}		T _{CLCH-45}		ns	
T _{CHSV}	Retraso estado activo	10	55	10	76	ns	
T _{CLSH}	Retraso estado inactivo	10	55	10	76	ns	
T _{CLTMV}	Retraso salida temporizador		60		75	ns	100 pF máximo
T _{CLRO}	Retraso reinicio		60		75	ns	
T _{CHQSV}	Retraso estado cola		35		44	ns	

Respuestas de sincronización de selección de chip del 80186

Símbolo	Parámetros	Mín.	Máx.	Mín.	Máx.	Unidades	Condiciones de prueba
T _{CLSV}	Retraso selección de chip activa		66		80	ns	
T _{CXCSX}	Retención de selección de chip desde comando inactiva	35		35		ns	
T _{CHCSX}	Retraso selección de chip inactiva	5	35	5	47	ns	

Símbolo	Parámetro	Mín.	Máx.	Unidades	Condiciones de prueba
TDVCL	Datos en preparación (A/D)	20		ns	
TCLDX	Datos en retención (A/D)	10		ns	
TARYHCH	Tiempo* de preparación señal READY asíncrona (READY)	20		ns	
TARYLCL	Tiempo de preparación READY inactiva	35		ns	
TCHARYX	Tiempo retención READY	15		ns	
TSRYCL	Tiempo de preparación transición SREADY	35		ns	
TCLSRY	Tiempo de retención transición SREADY	15		ns	
THVCL	INTR, NMI, TEST, TIMERIN, Preparación*	25		ns	
TINVCL	DRQ0, DRQ1, Preparación*	25		ns	

* Para garantizar el reconocimiento en el siguiente ciclo de reloj.

FIGURA 16-5 Características de AC del 80186. (Cortesía de Intel Corporation.)

16-2

PROGRAMACIÓN DE LAS MEJORAS DEL 80186/80188

En esta sección proporcionaremos los detalles acerca de la programación y operación de las mejoras en todas las versiones (XL, EA, EB y EC) de los microprocesadores 80186/80188. En la siguiente sección detallaremos el uso del 80C188EB en un sistema que utiliza muchas de las mejoras que veremos aquí. La única característica nueva que no veremos aquí es el generador de reloj, el cual se describió en la sección anterior sobre la arquitectura.

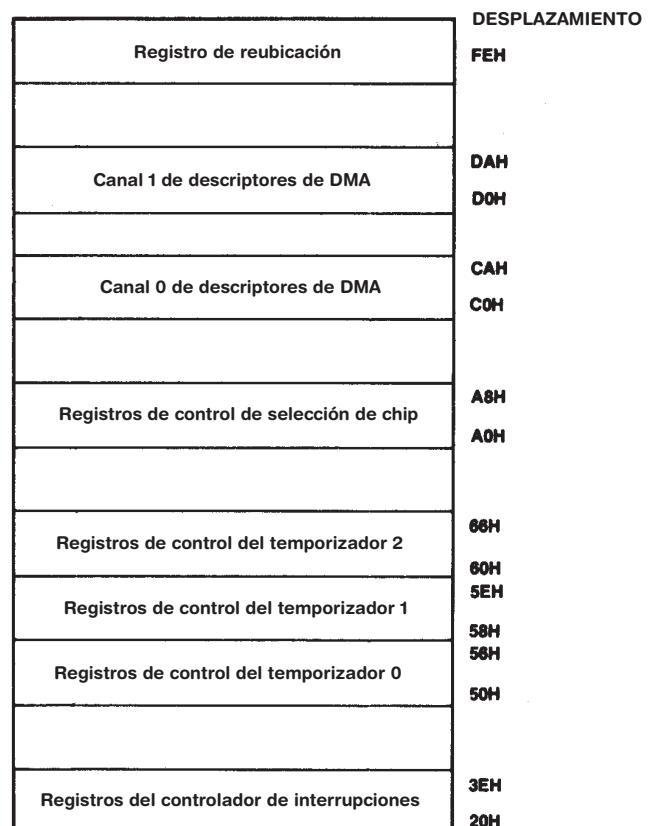
Bloque de control de periféricos

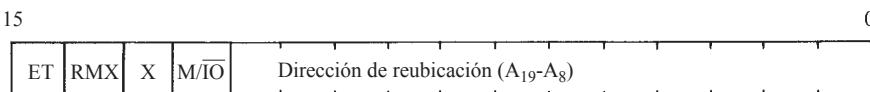
Todos los periféricos internos se controlan mediante un conjunto de registros de 16 bits que se encuentran en el bloque de control de periféricos (PCB). El PCB (vea la figura 16-6) es un conjunto de 256 registros que se encuentran en la E/S o en el espacio de memoria. Tenga en cuenta que este conjunto se aplica en las versiones XL y EA. Más adelante en esta sección definiremos y describiremos las versiones EB y EC del PCB.

Cada vez que se reinicia el 80186/80188, el bloque de control de periféricos se ubica de manera automática en la parte superior del mapa de E/S (direcciones de E/S FF00H-FFFFH). En la mayoría de los casos permanece en esta área de espacio de E/S, pero el PCB puede reubicarse en cualquier momento en otra área de memoria o de E/S. La reubicación se lleva a cabo mediante la modificación del contenido del registro de reubicación (vea la figura 16-7) que se encuentra en las direcciones de desplazamiento FEH y FFH.

El registro de reubicación se establece en 20FFH cuando se reinicia el 80186/80188. Esto hace que el PCB se ubique en las direcciones de E/S FF00H-FFFFH. Para reubicar el PCB, el usuario sólo tiene que enviar una palabra OUT a la dirección de E/S FFFEH con un nuevo patrón de bits. Por ejemplo, para reubicar el PCB hacia las posiciones de memoria 20000H-200FFH, se envía un 1200H a

FIGURA 16-6 El bloque de control de periféricos (PCB) del 80186/80188. (Cortesía de Intel Corporation.)





ET = ESC/ NO ESC TRAP

RMX = modo/modo maestro iRM × 86

M/IO = Espacio de memoria/ES

X = Sin utilizar

FIGURA 16-7 El registro de control de periféricos.

la dirección de E/S FFFEH. Observe que M/IO es un 1 lógico para seleccionar la memoria, así como que 200H selecciona la dirección de memoria 20000H como la dirección base del PCB. Todos los accesos al PCB deben ser tipo palabra, ya que está organizado en forma de registros de 16 bits. El ejemplo 16-1 muestra el software requerido para reubicar el PCB hacia las posiciones de memoria 20000H-200FFH. Puede utilizarse una salida de 8 o de 16 bits para programar el 80186; en el 80188 nunca hay que utilizar la instrucción OUT DX,AX, debido a que requieren períodos de reloj adicionales para ejecutarse.

EJEMPLO 16-1

```

MOV      DX,0FFEH          ;registro de reubicación de direcciones
MOV      AX,1200H          ;nueva ubicación del PCB
OUT      DX,AL             ;también puede ser OUT DX,AX

```

Las versiones EB y EC utilizan una dirección distinta para programar la ubicación del PCB. Ambas versiones almacenan el registro de reubicación del PCB en la dirección de desplazamiento XXA8H, en vez de la dirección de desplazamiento XXFEH en las versiones XL y EA. El patrón de bits de estas versiones es el mismo que para las versiones XL y EA, sólo que falta el bit RMX.

Interrupciones en el 80186/80188

Las interrupciones en el 80186/80188 son idénticas a las del 8086/8088, sólo que se definen vectores de interrupción adicionales para algunos de los dispositivos internos. En la tabla 16-3 aparece una lista completa de los vectores de interrupción reservados. Los primeros cinco son idénticos a los del 8086/8088.

La interrupción por instrucción BOUND de arreglo se produce si el límite de un registro índice se encuentra fuera de los valores establecidos en la memoria. La interrupción por código de operación no utilizado se produce cada vez que el 80186/80188 ejecuta un código de operación no definido. Esto es importante si un programa empieza a ejecutarse de manera extraña. Una instrucción puede acceder a la interrupción por código de operación no utilizado, pero el ensamblador no la incluye en el conjunto de instrucciones. En los microprocesadores del Pentium Pro al Pentium 4 y en algunos de los primeros microprocesadores Intel la instrucción 0F0BH o la 0FB9H harán que el programa llame al procedimiento cuya dirección está almacenada en el vector de interrupción por código de operación no utilizado.

La interrupción por código de operación ESC se produce si se ejecutan los códigos de operación de ESCape D8H-DFH. Esto ocurre sólo si está activo el bit ET (trampa de escape) del registro de reubicación. Si se produce una interrupción ESC, la dirección almacenada en la pila por la interrupción apuntará a la instrucción ESC o a su prefijo de sustitución de segmento, si es que se utiliza uno.

Las interrupciones de hardware interno deben habilitarse mediante el bit de bandera I y desencadenarse para funcionar. El bit de bandera I se activa (habilita) con STI y se borra (deshabilita) con CLI. Más adelante, en esta sección, hablaremos sobre el resto de las interrupciones decodificadas en forma interna, junto con los temporizadores y el controlador DMA.

Controlador de interrupciones

El controlador de interrupciones dentro del 80186/80188 es un dispositivo sofisticado. Tiene muchas entradas de interrupción, que llegan desde las cinco entradas de interrupción externas, el controlador de DMA y los tres temporizadores. La figura 16-8 proporciona un diagrama de bloques de la estructura de interrupciones del controlador de interrupciones 80186/80188. Este controlador aparece en las

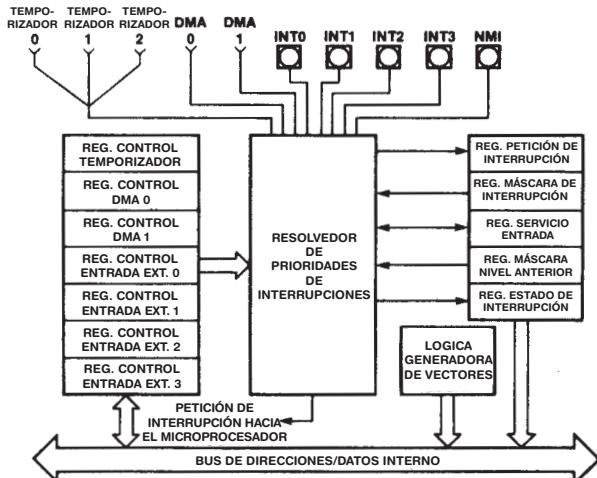
TABLA 16-3

Los vectores de interrupción del 80186/80188.

Nombre	Tipo	Dirección	Prioridad
Error de división	0	00000-00003	1
Un solo paso	1	00004-00007	1A
Terminal NMI	2	00008-0000B	1
Punto de interrupción	3	0000C-0000F	1
Desbordamiento	4	00010-00013	1
Instrucción BOUND	5	00014-00017	1
Código de operación no utilizado	6	00018-0001B	1
Código de operación de ESCape	7	0001C-0001F	1
Temporizador 0	8	00020-00023	2A
Reservado	9	00024-00027	—
DMA 0	A	00028-0002B	4
DMA 1	B	0002C-0002F	5
INT ₀	C	00030-00033	6
INT ₁	D	00034-00037	7
INT ₂	E	00038-0003B	8
INT ₃	F	0003C-0003F	9
Coprocessador 80187	10	00040-00043	1
Reservado	11	00044-00047	—
Temporizador 1	12	00048-0004B	2B
Temporizador 2	13	0004C-0004F	2C
Receptor serial	14	00050-00053	3A
Transmisor serial	15	00054-00057	3B

Nota: En esta tabla, el nivel 1 es la prioridad más alta, mientras que el nivel 9 es la más baja. Observe que algunos interruptores tienen la misma prioridad, sólo los modelos EB y EC contienen el puerto serial.

FIGURA 16-8 El controlador de interrupciones programable 80186/80188. (Cortesía de Intel Corporation.)



versiones XL, EA y EB, pero la versión EC contiene el equivalente exacto a un par de controladores 8259A, como se vio en el capítulo 12. En la versión EB, las entradas de DMA se sustituyen con entradas de la unidad serial para la recepción y la transmisión.

El controlador de interrupciones opera en dos modos: maestro y esclavo. El modo se selecciona mediante un bit en el registro de control de interrupción (versiones EB y EC), al cual se le conoce como *bit CAS*. Si el bit CAS es un 1 lógico, el controlador de interrupción se conecta a los controladores de interrupciones programables 8259A externos (vea la figura 16-9); si CAS es un 0 lógico, se selecciona el controlador de interrupciones interno. En muchos casos hay suficientes interrupciones dentro del 80186/80188, por lo que no se utiliza mucho el modo esclavo. En las versiones XL y EA, los modos maestro y esclavo se seleccionan en el registro de control de periféricos, en la dirección de desplazamiento FEH.

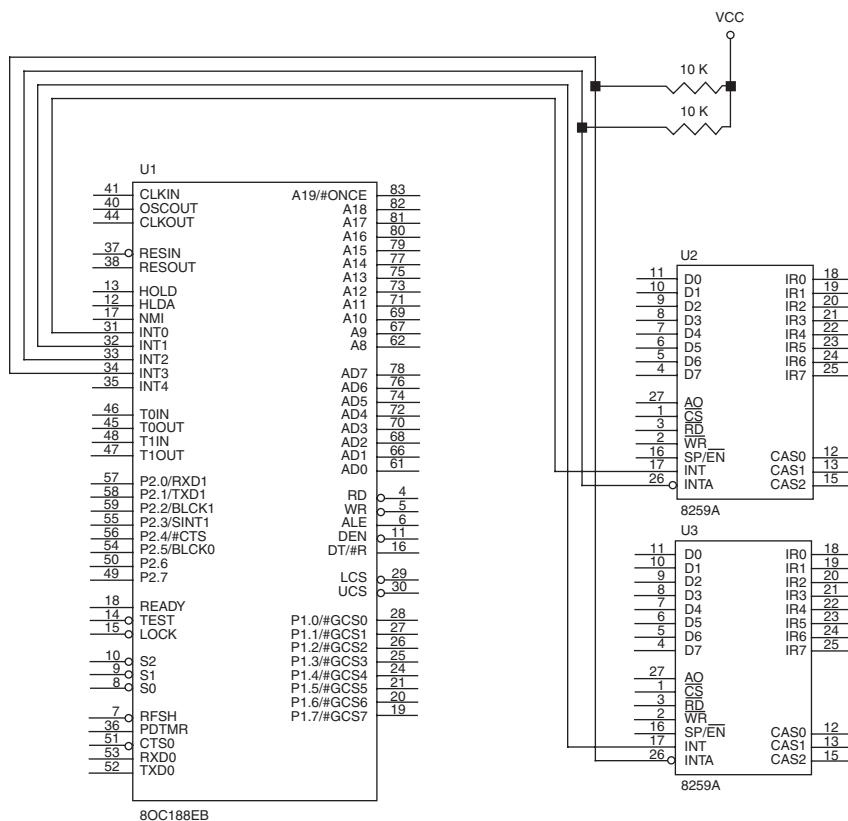


FIGURA 16-9 La interconexión entre el 80C188EB y dos controladores de interrupciones programables 8259A. *Nota:* Sólo se muestran las conexiones que son vitales para esta interfaz.

En esta porción del libro no detallaremos la programación del controlador de interrupciones. En vez de ello, nos limitaremos a una discusión sobre su estructura interna. Veremos la programación y la aplicación del controlador de interrupciones en las secciones en las que se describen el temporizador y el controlador de DMA.

Registros del controlador de interrupciones. La figura 16-10 muestra los registros del controlador de interrupciones. Tales registros se encuentran en el bloque de control de periféricos que empieza en la dirección de desplazamiento 22H. Para la versión EC, compatible con el 8259A, los puertos del controlador de interrupciones se encuentran en las direcciones de desplazamiento 00H y 02H para el maestro, así como en los puertos 04H y 06H para el esclavo. En la versión EB, el controlador de interrupciones se programa en la dirección de desplazamiento 02H. La versión EB tiene una entrada de interrupción adicional (INT4).

Modo esclavo. Cuando el controlador de interrupciones opera en el modo esclavo, utiliza hasta dos controladores de interrupciones programables 8259A para la expansión de las entradas de interrupción. La figura 16-9 muestra cómo se conectan los controladores de interrupciones externos a las terminales de entrada de interrupción del 80186/80188 para la operación en modo esclavo. Aquí, las entradas INT₀ e INT₁ se utilizan como conexiones externas para las salidas de petición de interrupción de los controladores 8259A, en tanto que las terminales INTA₀ (INT₂) e INTA₁ (INT₃) se utilizan como señales de reconocimiento de interrupción para los controladores externos.

Registros de control de interrupciones. Hay registros de control de interrupciones en ambos modos de operación, cada uno de los cuales controla una sola fuente de interrupción. La figura 16-11 muestra el patrón de bits binario de cada uno de estos registros de control de interrupciones. El bit de máscara habilita (0) o deshabilita (1) la entrada de interrupción representada por la palabra de control, y los bits de prioridad establecen el nivel de prioridad de la fuente de interrupción. El nivel de prioridad más alto

	Versiones XL y EA	Versión EB
3EH	Registro de control INT3	1EH
3CH	Registro de control INT2	1CH
3AH	Registro de control INT1	1AH
38H	Registro de control INT0	18H
36H	Registro de control DMA1	16H
34H	Registro de control DMA0	14H
32H	Registro de control temporizador	12H
30H	Estado de la interrupción	10H
2EH	Petición	0EH
2CH	En servicio	0CH
2AH	PRIMSK	0AH
28H	Máscaras de interrupción	08H
26H	Estado de POLL	06H
24H	POLL	04H
22H	EOI	02H

FIGURA 16-10 La asignación de puertos de desplazamiento de E/S para la unidad de control de interrupciones.

es 000 y el más bajo es 111. El bit CAS se utiliza para habilitar el modo esclavo o modo cascada (0 habilita el modo esclavo) y el bit SFNM selecciona el **modo anidado completo especial**. El SFNM permite mantener la estructura de prioridades del 8259A.

Registro de petición de interrupciones. Este registro contiene una imagen de las fuentes de interrupción en cada modo de operación. Cada vez que se solicita una interrupción, el bit de petición de interrupción correspondiente se vuelve un 1 lógico, incluso aunque la interrupción esté enmascarada. La petición se borra cada vez que el 80186/80188 reconoce la interrupción. La figura 16-12 muestra el patrón de bits binario del registro de petición de interrupciones para los modos maestro y esclavo.

Registros de máscara y máscara de prioridad. El registro de máscara de interrupción tiene el mismo formato que el registro de interrupciones que se muestra en la figura 16-12. Si una fuente se enmascara (deshabilita), el bit correspondiente del registro de máscara de interrupción contiene un 1 lógico; si se habilita, contiene un 0 lógico. El registro de máscara de interrupción se lee para determinar cuáles fuentes de interrupción están enmascaradas y cuáles están habilitadas. Para enmascarar una fuente se activa el bit de máscara de la fuente en su registro de control de interrupciones.

El registro de máscara de prioridad, que se muestra en la figura 16-13, indica la prioridad de la interrupción que se está atendiendo en ese momento por el 80186/80188. El nivel de la interrupción se indica mediante los bits de prioridad P₂-P₀. Estos bits previenen una interrupción en forma interna debido a una fuente de menor prioridad. Estos bits se ajustan de manera automática al siguiente nivel más bajo al final de una interrupción, según lo que envíe el 80186/80188.

Registro de servicio de entrada. Este registro tiene el mismo patrón de bits binario que el registro de petición de la figura 16-12. El bit que corresponde a la fuente de interrupción se activa si el 80186/80188 está reconociendo la entrada en ese momento. El bit se reinicia al final de una interrupción.

Los registros de sondeo y de estado de sondeo. Ambos registros comparten los mismos patrones de bits binarios que se muestran en la figura 16-14. Estos registros tienen un bit (INT REQ) que indica que hay una interrupción pendiente. Este bit se activa si se recibe una interrupción con suficiente prioridad

Registros de control serial y de temporizador

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
															MASC P2 P1 P0

Registros de control de INT2, INT3 e INT4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
													LVL MASC P2 P1 P0		

Registros de control de INT0 e INT1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										SFNM CAS LVL MASC P2 P1 P0					

P2-P0 = Nivel de prioridad

MASC (Máscara) = 0 habilita interrupción

LVL = disparo por 0 = borde y 1 = nivel

CAS = 1 selecciona el modo esclavo

SFNM = 1 selecciona modo anidado completo especial

FIGURA 16-11 Los registros de control de interrupciones.**FIGURA 16-12** El registro de petición de interrupción.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									I N T 3 I N T 2 I N T 1 I N T 0 I N T 4 SER T IM						

Registro de petición de interrupción (versión EB)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									I N T 3 I N T 2 I N T 1 I N T 0 D M A 1 D M A 0 T IM						

FIGURA 16-13 El registro de máscara de prioridad.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
													P2 P1 P0		

P2-P0 = Nivel de prioridad

FIGURA 16-14

Los registros de sondeo y estado de sondeo.

Registros de sondeo y estado de sondeo															
15											V	V	V	V	V
I											T	T	T	T	T
R											4	3	2	1	0
E															
Q															

$\text{IREQ} = 1 = \text{Interrupción pendiente}$
 $\text{VT4-VT0} = \text{Interrupción pendiente del número de tipo de interrupción más alto}$

FIGURA 16-15 El registro de fin de interrupción (EOI).

Registro de fin de interrupción															
15											V	V	V	V	V
N											T	T	T	T	T
S											4	3	2	1	0
P															
E															
C															

y se borra cuando se reconoce una interrupción. Los bits S indican el número de tipo de vector de interrupción de la interrupción pendiente con mayor prioridad.

Los registros de sondeo y estado de sondeo parecen idénticos debido a que contienen la misma información. No obstante, su función es distinta. Cuando se lee el registro de sondeo de interrupción, se reconoce la interrupción. Cuando se lee el registro de estado de sondeo de interrupción, no se envía un reconocimiento. Estos registros se utilizan sólo en el modo maestro, no en el esclavo.

Registro de fin de interrupción. El registro de fin de interrupción (EOI) produce la terminación de una interrupción cuando se escribe mediante un programa. La figura 16-15 muestra el contenido del registro EOI para los modos esclavo y maestro.

En el modo maestro, la escritura en el registro EOI concluye un nivel de interrupción específico (número de vector) o cualquier nivel que esté activo en ese momento (no específico). En el modo no específico, debe activarse el bit NSPEC antes de que se escriba en el registro EOI para terminar una interrupción no específica. El EOI no específico borra el bit de interrupción de nivel más alto en el registro de servicio de entrada. El EOI específico borra el bit seleccionado en el registro de servicio de entrada, el cual informa al microprocesador que se ha atendido la interrupción y que puede aceptarse otra interrupción del mismo tipo. El modo no específico se utiliza a menos que haya una circunstancia especial que requiera un distinto orden para los reconocimientos de interrupciones. Si se requiere un EOI específico, el número de vector se coloca en el comando EOI. Por ejemplo, para borrar la interrupción del temporizador 2 el comando EOI es 13H (el vector para el temporizador 2).

En el modo esclavo, se escribe el nivel de la interrupción que finalizará en el registro EOI. El modo esclavo no permite un EOI no específico.

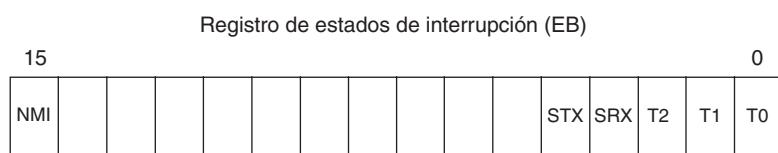
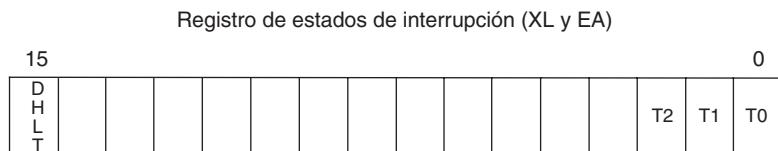
Registro de estado de interrupción. En la figura 16-16 se muestra el formato del registro de estado de interrupción. En el modo maestro, T_2-T_0 indica cuál temporizador (0, 1 o 2) está produciendo una interrupción. Esto es necesario porque los tres temporizadores tienen el mismo nivel de prioridad de interrupción. Estos bits se activan cuando el temporizador solicita una interrupción y se borran cuando se reconoce la interrupción. El bit DHLT (alto de DMA) se utiliza sólo en el modo maestro; cuando se activa, detiene una acción de DMA. Tome en cuenta que el registro de estado de interrupción es distinto para la versión EB.

Registro del vector de interrupción. El registro de vectores de interrupción está presente sólo en el modo esclavo y en las versiones XL y EA, en la dirección de desplazamiento 20H. Se utiliza para especificar los cinco bits más significativos del número de tipo de vector de interrupción. La figura 16-17 ilustra el formato de este registro.

Temporizadores

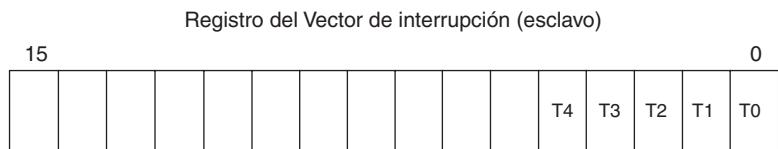
Los microprocesadores 80186/80188 contienen tres temporizadores de 16 bits 100% programables, cada uno totalmente independiente de los otros. Dos de los temporizadores (temporizador 0 y temporizador 1) tienen terminales de entrada y de salida que les permiten contar eventos externos o generar formas de

FIGURA 16-16 El registro de estados de interrupción.



SRX = receptor serial
STX = transmisor serial

FIGURA 16-17 El registro del vector de interrupción.



onda. El tercer temporizador (temporizador 2) se conecta al reloj del 80186/80188. El temporizador 2 se utiliza como fuente de petición de DMA, como preescalador para otros temporizadores o como temporizador watchdog.

La figura 16-18 muestra la estructura interna de la unidad de temporizadores. Esta unidad contiene un elemento de conteo responsable de la actualización de los tres contadores. Cada temporizador es en realidad un registro que se reescribe a partir del elemento de conteo (un circuito que lee un valor de un registro de temporizador y lo incrementa antes de regresarlo). El elemento contador también es responsable de generar las salidas en las terminales T₀_{OUT} y T₁_{OUT}, de leer las terminales T₀_{IN} y T₁_{IN}, así como de producir una petición de DMA desde la cuenta terminal (TC) del temporizador 2, si éste se programa para solicitar una acción de DMA.

Operación del registro de temporizadores. Los temporizadores se controlan mediante un bloque de registros en el bloque de control de periféricos (vea la figura 16-19). Cada temporizador tiene un

FIGURA 16-18 Estructura interna de los temporizadores del 80186/80188. (Cortesía de Intel Corporation.)

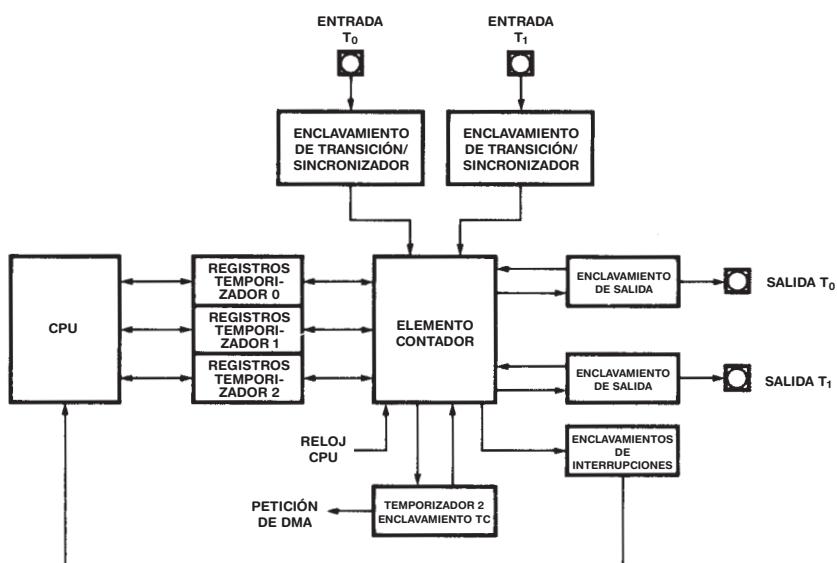
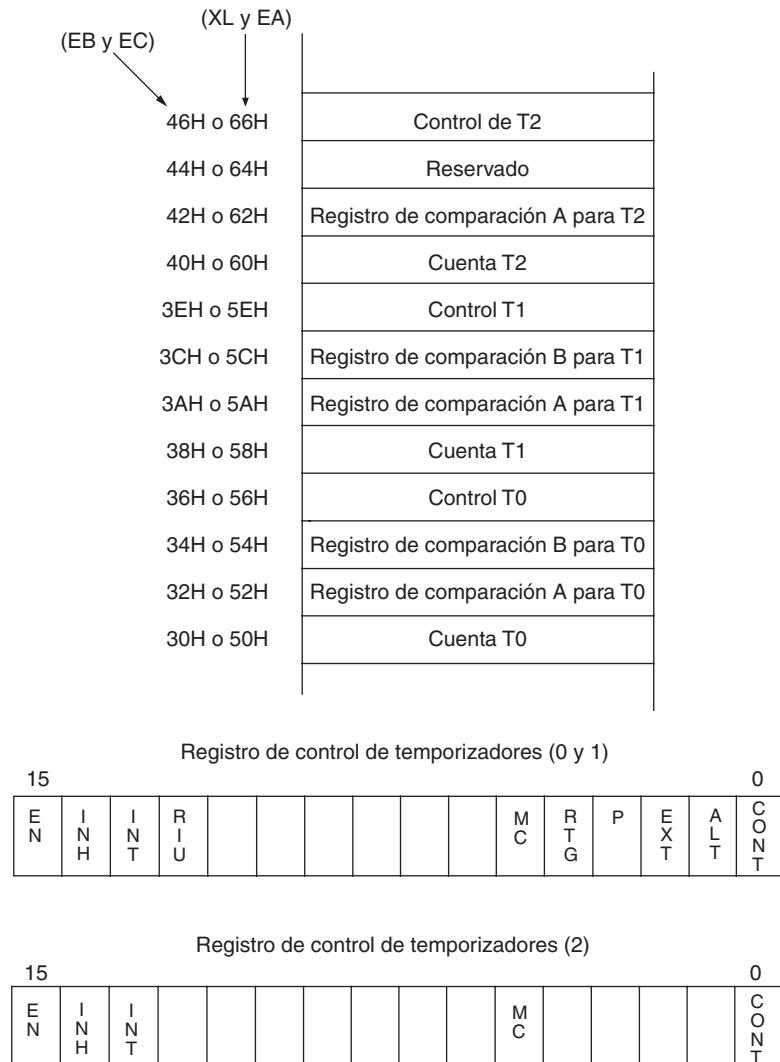


FIGURA 16-19

Las posiciones de desplazamiento y el contenido de los registros que se utilizan para controlar los temporizadores.



registro de conteo, un registro o registros de conteo máximo y un registro de control. Se puede leer y escribir en todos estos registros en cualquier momento, ya que los microprocesadores 80186/80188 aseguran que el contenido nunca cambiará durante su lectura o escritura.

El registro de conteo de temporizador contiene un número de 16 bits que se incrementa cada vez que ocurre una entrada para el temporizador. Los temporizadores 0 y 1 se incrementan en el borde positivo de una terminal de entrada externa, cada cuarto ciclo de reloj del 80186/80188, o mediante la salida del temporizador 2. Este temporizador recibe un pulso cada cuarto ciclo de reloj del 80186/80188, pero no tiene otra fuente de sincronización. Esto significa que en la versión de 8 MHz del 80186/80188, el temporizador 2 opera a 2 MHz, en tanto que la frecuencia máxima de conteo de los temporizadores 0 y 1 es de 2 MHz. La figura 16-20 muestra estos cuatro períodos de reloj, que no están relacionados con la sincronización del bus.

Cada temporizador tiene cuando menos un registro de conteo máximo, al cual se le conoce como registro de **comparación** (registro de comparación A para los temporizadores 0 y 1) y se carga con la cuenta máxima del registro de conteo para generar una salida. Tenga en cuenta que un temporizador es un contador ascendente. Cada vez que el registro de conteo es igual al registro de comparación de conteo máximo, se ajusta en 0. Con un conteo máximo de 0000H, el contador cuenta 65,536 veces. Para cualquier otro valor, el temporizador cuenta el valor real del conteo. Por ejemplo, si la cuenta máxima es de 0002H, el contador contará de 0 a 1 y después se borrará para quedar en 0; un contador módulo 2 tiene dos estados.

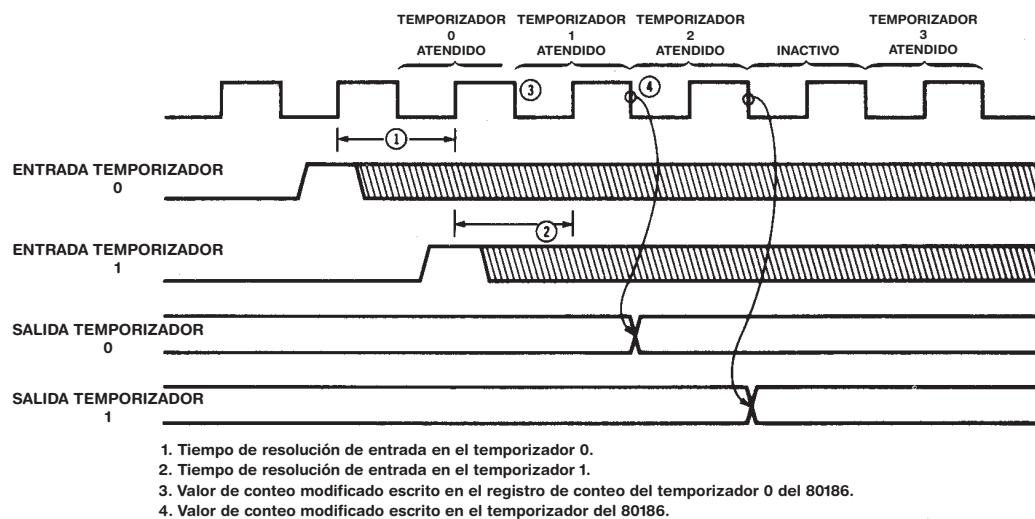


FIGURA 16-20 Sincronización para los temporizadores del 80186/80188. (Cortesía de Intel Corporation.)

Cada uno de los temporizadores 0 y 1 llega a tener un segundo registro de comparación de conteo máximo (registro de comparación B) que se seleccione mediante el registro de control para el temporizador. Pueden utilizarse cualquiera de los registros de comparación de conteo máximo A o ambos registros A y B con tales temporizadores, según lo programado por el bit ALT en el registro de control para el temporizador. Cuando se utilizan ambos registros de comparación de conteo máximo, el temporizador cuenta hasta el valor que se encuentra en el registro de comparación de conteo máximo A, se borra para quedar en 0 y después cuenta hasta el valor contenido en el registro de comparación de conteo máximo B. Luego se repite este proceso. El uso de ambos registros de conteo máximo permite al temporizador contar hasta 131,072.

El registro de control (vea la figura 16-19) de cada temporizador es de 16 bits y especifica la operación del temporizador. A continuación se muestra una definición de cada bit de control:

EN	El bit de habilitación permite que el temporizador empiece a contar. Si EN está en ceros, el temporizador no cuenta; si está activo, el temporizador cuenta.
INH	El bit de inhibición permite que una escritura en el registro de control de temporizadores afecte al bit de habilitación (EN). Si INH está activo, entonces el bit EN puede activarse o borrarse para controlar el conteo. Si INH está en cero, EN no se ve afectado por una escritura en el registro de control de temporizadores. Esto permite modificar otras características del temporizador sin habilitarlo o deshabilitarlo.
INT	El bit de interrupción permite que el temporizador genere una interrupción. Si INT está activo, se producirá una interrupción cada vez que se llegue al conteo máximo en cualquiera de los registros de comparación de conteo máximo. Si este bit está en cero, no se genera una interrupción. Cuando se genera la petición de interrupción, ésta permanece en efecto aun cuando el bit EN se borre después de la petición de interrupción.
RIU	El bit registro en uso indica cuál registro de comparación de conteo máximo está utilizando el temporizador. Si RIU es un 0 lógico, entonces se está utilizando el registro de comparación de conteo máximo A. Este bit es de sólo lectura, por lo que las escrituras no le afectan.
MC	El bit de conteo máximo indica que el temporizador ha llegado a su conteo máximo. Este bit se convierte en 1 lógico cuando el temporizador llega al conteo máximo y permanece en 1 lógico hasta que se borre el bit MC mediante la escritura de un 0 lógico. Esto permite que el software detecte el conteo máximo.
RTG	El bit de redisparo se activa sólo cuando se utiliza un pulso de reloj externo (EXT = 0). El bit RTG se utiliza sólo con los temporizadores 0 y 1 para seleccionar

la operación de las terminales de entrada del temporizador ($T0_{IN}$ y $T1_{IN}$). Si RTG es un 0 lógico, la entrada externa hará que el temporizador cuente si es un 1 lógico; el temporizador retendrá su conteo (dejará de contar) si es un 0 lógico. Si RTG es un 1 lógico, la terminal de entrada externa borra la cuenta del temporizador y la pone en 0000H cada vez que se produce un borde positivo.

P	El bit preescalador selecciona la fuente de pulsos de reloj para los temporizadores 0 y 1. Si $EXT = 0$ y $P = 0$, la frecuencia de la fuente equivale a una cuarta parte de la frecuencia de reloj del sistema. Si $P = 1$, la fuente es el temporizador 2.
EXT	El bit externo selecciona la sincronización interna ($EXT = 0$) o la sincronización externa ($EXT = 1$). Si $EXT = 1$, la fuente de sincronización se aplica a las terminales $T0_{IN}$ o $T1_{IN}$. En este modo, el temporizador se incrementa después de cada borde positivo en la terminal de entrada del temporizador. Si $EXT = 0$, la fuente de pulsos de reloj proviene de una de las fuentes internas.
ALT	El bit alternativo selecciona el modo de conteo máximo individual (registro de comparación de conteo máximo A), si es un 0 lógico, o el modo de conteo máximo alternativo (registros de comparación de conteo máximo A y B), si es un 1 lógico.
CONT	El bit continuo selecciona la operación continua si es un 1 lógico. En este tipo de operación, el contador sigue contando de manera automática hasta que llega al máximo. Si $CONT = 0$ lógico, el temporizador dejará de contar de manera automática y borrará el bit EN. Cada vez que se reinicia el 80186/80188, los temporizadores se deshabilitan de manera automática.

Terminal de salida de temporizador. Los temporizadores 0 y 1 tienen una terminal de salida que se utiliza para generar ondas cuadradas o pulsos. Para producir pulsos, el temporizador se opera en modo de conteo máximo individual ($ALT = 0$). En este modo, la terminal de salida se va al nivel bajo durante un periodo de reloj, cuando el contador llega al máximo. Mediante el control del bit $CONT$ en el registro de control, pueden generarse un pulso individual o pulsos continuos.

Para producir ondas cuadradas o ciclos de trabajo variables, se selecciona el modo alternativo ($ALT = 1$). En este modo, la terminal de salida es un 1 lógico, mientras el registro de comparación de conteo máximo A controla el temporizador; es un 0 lógico, mientras el registro de comparación de conteo máximo B controla el temporizador. Al igual que en el modo de conteo máximo individual, el temporizador puede generar una sola onda cuadrada o varias ondas cuadradas continuas. En la tabla 16-4 verá la función de los bits de control ALT y $CONT$.

En el modo alternativo puede generarse casi cualquier ciclo de trabajo. Por ejemplo, suponga que se requiere un ciclo de trabajo del 10% en la terminal de salida de un temporizador. El registro de conteo máximo A se carga con un 10 y el registro de conteo máximo B se carga con un 90 para producir una salida que sea un 1 lógico durante 10 ciclos de reloj y un 0 lógico durante 90 ciclos de reloj. Esto también divide la frecuencia de la fuente de sincronización entre un factor de 100.

Ejemplo de reloj en tiempo real. Muchos sistemas requieren la hora del día. A esto se le conoce como **reloj en tiempo real** (RTC). Un temporizador dentro del 80186/80188 puede proporcionar la fuente de sincronización para el software que mantiene la hora del día.

No mostramos el hardware requerido para esta aplicación, ya que todo lo que se requiere es conectar la terminal $T1_{IN}$ a +5.0 V a través de una resistencia elevadora para habilitar el temporizador 1. En el ejemplo, los temporizadores 1 y 2 se utilizan para generar una interrupción de un segundo que proporciona al software una fuente de sincronización.

TABLA 16-4 Los bits ALT y $CONT$ en el registro de control de temporizadores.

<i>ALT</i>	<i>CONT</i>	<i>Modo</i>
0	0	Pulso individual
0	1	Pulsos continuos
1	0	Onda cuadrada individual
1	1	Ondas cuadradas continuas

En los ejemplos 16-2 y 16-3 se muestra el software requerido para implementar un reloj en tiempo real. El ejemplo 16-2 muestra el software necesario para inicializar los temporizadores. El ejemplo 16-3 contiene un procedimiento de servicio de interrupciones, el cual mantiene la hora. Hay otro procedimiento en el ejemplo 16-3 que incrementa un contador módulo BCD. Tampoco se muestra el software utilizado para instalar el vector de interrupción y establecer o mostrar la hora del día.

EJEMPLO 16-2

```
;el software está escrito para la versión EB del 80186/80188 que
;inicializa y arranca los temporizadores 1 y 2.

;direcciónamiento de igualaciones

T2_CA    EQU    OFF42H          ;registro A de comparación, temporizador 2
T2_CON   EQU    OFF46H          ;registro de control, temporizador 2
T2_CNT   EQU    OFF40H          ;registro de conteo, temporizador 2
T1_CA    EQU    OFF3AH          ;registro A de comparación, temporizador 1
T1_CON   EQU    OFF38H          ;registro de control, temporizador 1
T1_CNT   EQU    OFF3EH          ;registro de conteo, temporizador 1

        MOV AX,20000          ;programa el temporizador 2 para 10 mseg
        MOV DX,T2_CA
        OUT DX,AX

        MOV AX,100            ;programa el temporizador 1 para 1 seg
        MOV DX,T1_CA
        OUT DX,AX

        MOV AX,0              ;borra los registros de conteo
        MOV DX,T2_CNT
        OUT DX,AX
        MOV DX,T1_CNT
        OUT DX,AX

        MOV AX,0C001H          ;habilita el temporizador 2 y lo inicia
        MOV DX,T2_CON
        OUT DX,AX

        MOV AX,0E009H          ;habilita el temporizador 1 con interrupción
        MOV DX,T1_CON
        OUT DX,AX
```

El temporizador 2 se programa para dividir por un factor de 20,000. Esto hace que el reloj (suponiendo que sea de 2 MHz en la versión del 80186/80188 de 8 MHz) se divida hasta llegar a un pulso cada 10 ms. El reloj para el temporizador 1 se deriva en forma interna de la salida del temporizador 2. El temporizador 1 se programa para dividir el reloj del temporizador 2 entre 100 y generar un pulso una vez por segundo. El registro de control del temporizador 1 se programa de tal forma que el pulso de un segundo genere una interrupción en forma interna.

El procedimiento de servicio de interrupciones se llama una vez por segundo para mantener la hora. Este procedimiento suma uno al contenido de la posición de memoria SEGUNDOS en cada interrupción. Una vez cada 60 segundos se incrementa el contenido de la siguiente posición de memoria (SEGUNDOS + 1). Por último, el contenido de la posición de memoria SEGUNDOS + 2 se incrementa una vez cada hora. El tiempo se almacena en estas tres posiciones de memoria consecutivas en formato BCD, para que el software del sistema tenga un fácil acceso a la hora del día.

EJEMPLO 16-3

```
SEGUNDOS DB    ?
MINUTOS   DB    ?
HORAS     DB    ?

SINTR     PROC   FAR   USES   DS   AX   SI
                    MOV    AX,DIRECCION_SEGMENTO
                    MOV    DS,AX
```

```

MOV AH, 60H ;carga módulo 60
MOV SI, OFFSET SEGUNDOS ;direcciona el reloj
CALL UPS ;incrementa los segundos
.JF ZERO? ;si segundos se hizo 0
    CALL UPS ;incrementa los minutos
    MOV AH, 24H ;carga módulo 24
    .IF ZERO? ;si minutos se hizo 0
        CALL UPS ;incrementa las horas
    .ENDIF
.ENDIF
MOV DX, OFF02H ;borra interrupción
MOV AX, 8000H
OUT DX, AX

RET

SINTR ENDP

UPS PROC NEAR

MOV AL, [SI]
ADD AL, 1 ;incrementa el contador
DAA ;lo convierte en BCD
INC SI
.JF AL == AH ;evalúa operación módulo
    MOV AL, 0
.ENDIF
MOV [SI-1], AL
RET

UPS ENDP

```

Controlador de DMA

El controlador de DMA, que está dentro del 80186/80188, tiene dos canales de DMA 100% independientes. Cada uno trae su propio conjunto de registros de direcciones de 20 bits, por lo que cualquier posición de memoria o de E/S es accesible para una transferencia por DMA. Además, cada canal puede programarse para incrementarse o decrementarse en forma automática para los registros de origen o

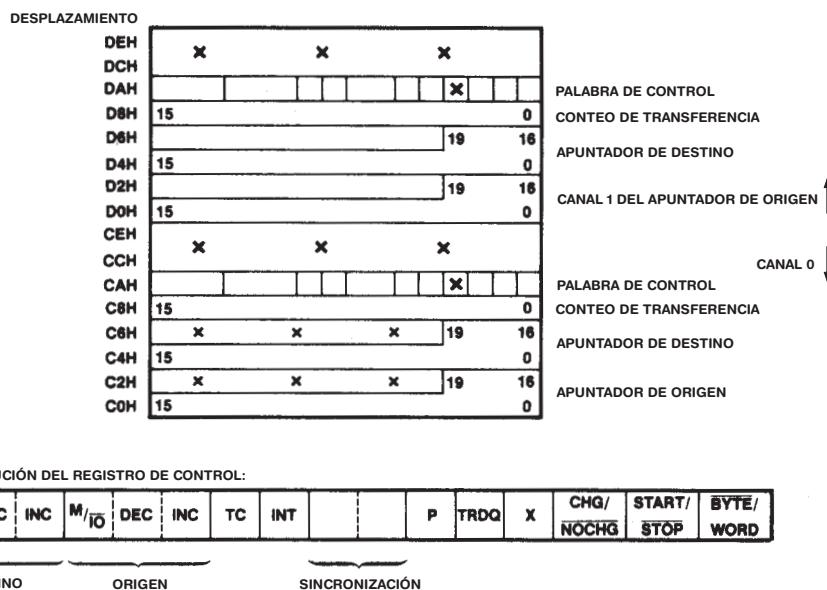


FIGURA 16-21 La estructura de los registros del controlador de DMA del 80186/80188.
(Cortesía de Intel Corporation.)

destino. El controlador no está disponible en las versiones EB o EC. La versión EC contiene un controlador de DMA de cuatro canales modificado; la versión EB no trae un controlador de DMA. En este texto no hablaremos sobre el controlador de DMA que está dentro de la versión EC.

La figura 16-21 muestra la estructura de registros internos del controlador de DMA. Estos registros se encuentran en el bloque de control de periféricos, en las direcciones de desplazamiento C0H-DFH.

Tome en cuenta que los conjuntos de registros de ambos canales de DMA son idénticos; cada canal contiene una palabra de control, un apuntador de origen y de destino, así como un conteo de transferencia. Este conteo, de 16 bits, permite transferencias por DMA desatendidas de bytes (80188/80186) y de palabras (sólo en el 80186). Cada vez que se transfieren un byte o una palabra, se reduce el conteo en uno hasta que llega a 0000H el conteo terminal.

Los apuntadores de origen y de destino son de 20 bits cada uno, por lo que es posible realizar transferencias por DMA hacia cualquier posición de memoria o dirección de E/S sin preocuparse por las direcciones de segmento y desplazamiento. Si la dirección de origen o de destino es un puerto de E/S, los bits A₁₉-A₁₆ deben ser 0000, o se producirá una falla.

Registro de control de canal. Cada canal de DMA contiene su propio registro de control de canal (vea la figura 16-21), el cual define su operación. Los seis bits de más a la izquierda especifican la operación de los registros de origen y de destino. El bit M/IO indica una posición de memoria o de E/S, DEC hace que se decremente el apuntador e INC, que se incremente. Si ambos bits INC y DEC son 1, entonces el apuntador permanece sin cambio después de cada transferencia por DMA. Es posible realizar transferencias de memoria a memoria con este controlador de DMA.

El bit TC (conteo terminal) hace que el canal DMA detenga las transferencias cuando el registro de conteo del canal se decrementa hasta 0000H. Si este bit es 1 lógico, el controlador de DMA sigue transfiriendo datos, aun cuando se llegue al conteo terminal.

El bit INT habilita las interrupciones para el controlador de interrupciones. Si se activa, el bit INT hace que se produzca una interrupción cuando se llega al conteo terminal del canal.

El bit SYN selecciona el tipo de sincronización para el canal: 00 = sin sincronización, 01 = sincronización de origen y 10 = sincronización de destino. Cuando se selecciona la sincronización de origen o destino, los datos se transfieren a la velocidad de 2 Mbytes por segundo. Estos dos tipos de sincronización permiten realizar transferencias sin interrupción. Si se selecciona la sincronización de destino, la velocidad de transferencia es menor (1.3 Mbytes por segundo) y el controlador delega el control al 80186/80188 después de cada transferencia de DMA.

El bit P selecciona la prioridad del canal. Si P = 1, el canal tiene la prioridad más alta. Si ambos canales tienen la misma prioridad, el control alterna las transferencias entre los canales.

El bit TRDQ habilita las transferencias por DMA del temporizador 2. Si este bit es un 1 lógico, la petición de DMA se origina del temporizador 2. Esto puede evitar que las transferencias por DMA utilicen todo el tiempo del microprocesador para la transferencia.

El bit CHG/NOCHG determina si START/STOP cambia para una escritura en el registro de control. El bit START/STOP inicia o detiene la transferencia por DMA. Para iniciar una transferencia por DMA, tanto CHG/NOCHG como START/STOP se colocan en nivel de 1 lógico.

El bit BYTE/WORD selecciona si el tamaño de la transferencia va a ser de byte o de palabra.

Ejemplo de transferencia de memoria a memoria. El controlador de DMA integrado es capaz de realizar transferencias de memoria a memoria. En el ejemplo 16-4 se muestra el procedimiento que se utiliza para programar el controlador e iniciar la transferencia.

EJEMPLO 16-4

```
.MODEL SMALL
.186
.CODE
;
;Transferencia de memoria a memoria mediante el uso de DMA
;
;Secuencia de llamada para MOVES
;
;      DS:DI = dirección de origen
;      ED:DI = dirección de destino
;      CX = Número de bytes
;
```

```

GETA    MACRO SEGA, OFFA, DMAA
        MOV AX,SEGA           ;;obtiene dirección de segmento
        SHL AX,4              ;;desplaza 4 posiciones a la izquierda
        ADD AX,OFFA            ;;agrega desplazamiento
        MOV DX,DMAA             ;;direcciona el controlador de DMA
        OUT DX,AX              ;;programa la dirección
        PUSHF
        MOV AX,SEGA
        SHR AX,12
        POPF
        ADC AX,0
        ADD DX,2
        OUT DX,AX
        ENDM

MOVES   PROC NEAR
        GETA DS,SI,0FFC0H      ;programa dirección de origen
        GETA ES,DI,0FFC4H      ;programa dirección de destino
        MOV DX,0FFC8H          ;programa el conteo
        MOV AX,CX
        OUT DX,AX
        MOV DX,0FFCAH          ;programa el control DMA
        MOV AX,0B606H
        OUT DX,AX              ;inicia la transferencia
        RET

MOVES   ENDP

```

El procedimiento del ejemplo 16-4 transfiere datos desde la posición del segmento de datos direccionala por SI, hasta la posición del segmento extra direccionala por DI. El número de bytes que se transfieren se guarda en el registro CX. Esta operación es idéntica a la instrucción REP MOVSB, sólo que la ejecución se realiza a una velocidad mucho mayor mediante el uso del controlador de DMA.

Unidad de selección de chip

La unidad de selección de chip simplifica la interfaz de la memoria y la E/S para el 80186/80188. Esta unidad contiene lógica programable de selección de chip. En sistemas pequeños y medianos no se requiere un decodificador externo para seleccionar la memoria y la E/S. En sistemas grandes, tal vez se requieran decodificadores externos. Existen dos formas de la unidad de selección de chip: una que se encuentra en las versiones XL, así como EA difiere de la unidad que viene en las versiones EB y EC.

Selecciones de chip de memoria. Se utilizan seis terminales (versiones XL y EA) o 10 terminales (versiones EB y EC) para seleccionar distintos componentes de memoria externa en un sistema pequeño o mediano basado en el 80186/80188. La terminal UCS (**selección de chip superior**) habilita el dispositivo de memoria que se encuentra en la porción superior del mapa de memoria que por lo general está poblado con ROM. Esta terminal programable permite especificar el tamaño de la ROM y el número de estados de espera requeridos. La dirección final de la ROM es FFFFFH. La terminal LCS (**selección de chip inferior**) selecciona el dispositivo de memoria (por lo general una RAM), que empieza en la posición de memoria 00000H. Al igual que con la terminal UCS, el tamaño de la memoria y el número de estados de espera son programables. Las cuatro u ocho terminales de chip de selección restantes eligen dispositivos de memoria media. Las cuatro terminales en las versiones XL y EA (MCS3-MCS0) se programan tanto para la dirección inicial (base) como para el tamaño de la memoria. Todos los dispositivos deben ser del mismo tamaño. Las ocho terminales (GCS7-GCS0) en las versiones EB y EC se programan con base en su tamaño y también por su dirección inicial; a la vez, llegan a representar a un dispositivo de memoria o de E/S.

Selecciones de chip de periférico. El 80186/80188 direcciona hasta siete dispositivos periféricos externos con las terminales PCS6-PCS0 (en las versiones XL y EA). Las terminales GCS se utilizan en las versiones EB y EC para seleccionar hasta ocho dispositivos de memoria o de E/S. La dirección de

TABLA 16-5 Los bits de control de estado de espera R₂, R₁ y R₀ (versiones XL y EA).

R ₂	R ₁	R ₀	Número de estados de espera	Se requiere READY
0	X	X	—	Sí
1	0	0	0	No
1	0	1	1	No
1	1	0	2	No
1	1	1	3	No

E/S base se programa en cualquier intervalo de 1 Kbyte con tamaños de bloque de direcciones de puerto de 128 bytes (64 bytes en las versiones EB y EC).

Programación de la unidad de selección de chip para las versiones XL y EA. El número de estados de espera en cada sección de la memoria y de la E/S son programables. Los microprocesadores 80186/80188 tienen un generador de estados de espera integrado que puede introducir entre 0 y 3 estados de espera (versiones XL y EA). La tabla 16-5 muestra los niveles lógicos requeridos en los bits R₂-R₀ en cada registro programable para seleccionar varios números de estados de espera. Estas tres líneas también determinan si se requiere una señal READY externa para generar estados de espera. Si se selecciona READY, la señal READY externa está en paralelo con el generador de estados de espera interno. Por ejemplo, si READY es un 0 lógico para tres períodos de reloj, pero el generador de estados de espera interno se programa para insertar dos estados de espera, se insertan tres estados de espera.

FIGURA 16-22 Los registros de selección de chip para las versiones XL y EA del 80186/80188.

Registro de control de memoria superior (desplazamiento = A0H)																	
15	0	A17	A16	A15	A14	A13	A12	A11	A10	0	0	0	R2	R1	R0	0	
Registro de control de memoria inferior (desplazamiento = A2H)																	
15	0	A17	A16	A15	A14	A13	A12	A11	A10	0	0	0	R2	R1	R0	0	
Dirección base de selección de chip de periférico (desplazamiento = A4H)																	
15	A19	A18	A17	A16	A15	A14	A13	0	0	0	0	0	R2	R1	R0	0	
Dirección base de memoria de medio rango (desplazamiento = A6H)																	
15	A19	A18	A17	A16	A15	A14	A13	0	0	0	0	0	R2	R1	R0	0	
Lado de memoria de medio rango (desplazamiento = A8H)																	
15	0	M6	M5	M4	M3	M2	M1	M0	EX	MS	0	0	0	R2	R1	R0	0

TABLA 16-6

Programación de la memoria superior para el registro A₀H (versiones XL y EA).

<i>Dirección inicial</i>	<i>Tamaño de bloque</i>	<i>Valor para estados de no espera, sin READY</i>
FFC00H	1 K	3FC4H
FF800H	2 K	3F84H
FF000H	4 K	3F04H
FE000H	8 K	3E04H
FC000H	16 K	3C04H
F8000H	32 K	3804H
F0000H	64 K	3004H
E0000H	128 K	1004H
C0000H	256 K	0004H

Suponga que una EPROM de 64 Kbytes se encuentra en la parte superior del sistema de memoria y requiere dos estados de espera para su operación apropiada. Para seleccionar el dispositivo para esta sección de memoria, se programa la terminal UCS para un rango de memoria de F0000H-FFFFFH con dos estados de espera. La figura 16-22 muestra un listado de los registros de control para todas las selecciones de memoria y de E/S en el bloque de control de periféricos en las direcciones de desplazamiento A₀-A₉H. Los tres bits de más a la derecha de estos registros de control vienen en la tabla 16-5. El registro de control para el área de memoria superior se encuentra en la dirección de desplazamiento A₀H del PCB. Este registro de 16 bits se programa con la dirección inicial del área de memoria (F0000H, en este caso) y el número de estados de espera. Tome en cuenta que los dos bits superiores de la dirección deben programarse como 00 y que sólo los bits de dirección A₁₇-A₁₀ se programan en el registro de control. En la tabla 16-6 verá ejemplos con los códigos para diversos tamaños de memoria. Como nuestro ejemplo requiere dos estados de espera, la dirección básica es la misma que en la tabla para un dispositivo de 64 K, sólo que los tres bits de más a la derecha son 110 en vez de 100. La información que se envía al registro de control de memoria superior es 3006H.

Suponga que una SRAM de 32 Kbytes que requiere estados de no espera y entrada no READY, se encuentra en la parte inferior del sistema de memoria. Con la finalidad de programar la terminal LCS para seleccionar este dispositivo, el registro A₂ se carga de la misma forma que el registro A₀H. En este ejemplo, se envía un 07FCH al registro A2H. La tabla 16-7 muestra los valores de programación para la salida de selección de chip inferior.

La parte central de la memoria se programa a través de dos registros: A₆H y A₈H. El registro A₆H programa la dirección inicial o base de las líneas de selección de memoria media (MCS3-MCS0) y el número de estados de espera. El registro A₈H define el tamaño del bloque de memoria y el de dispositivo de memoria individual (vea la tabla 16-8). Además del tamaño del bloque, el número de estados de espera de periférico se programa igual que otras áreas de la memoria. El EX (bit 7) y el MS (bit 6) especifican las líneas de selección de periférico; en breve hablaremos sobre esto.

TABLA 16-7 Programación de la memoria inferior para el registro A₂H (versiones XL y EA).

<i>Dirección final</i>	<i>Tamaño de bloque</i>	<i>Valor para estados de no espera, sin READY</i>
003FFFH	1 K	0004H
007FFFH	2 K	0044H
00FFFFH	4 K	00C4H
01FFFFH	8 K	01C4H
03FFFFH	16 K	03C4H
07FFFFH	32 K	07C4H
0xFFFFFH	64 K	0FC4H
1FFFFFFH	128 K	1FC4H
3FFFFFFH	256 K	3FC4H

TABLA 16-8 Programación de la memoria media para el registro A_{8H} (versiones XL y EA).

Tamaño de bloque	Tamaño de chip	Valor para estados de no espera, sin READY y EX = 0, MS = 1
8 K	2 K	0144H
16 K	4 K	0344H
32 K	8 K	0744H
64 K	16 K	0F44H
128 K	32 K	1F44H
256 K	64 K	3F44H
512 K	128 K	7F44H

Por ejemplo, suponga que se agregan cuatro SRAMs de 32 Kbytes al área de memoria media, comenzando en la posición 80000H y terminando en la posición 9FFFFH sin estados de espera. Para programar las líneas de selección de memoria media para esta área de memoria, colocamos los siete bits de dirección de más a la izquierda en el registro A_{6H}, en donde los bits 8-3 contienen 0s lógicos, y los tres bits de más a la derecha traen los bits de control de la señal listos (ready). Para este ejemplo, A_{6H} se carga con 8004H. El registro A_{8H} se programa con 1F44H, suponiendo que EX = 0 y MS = 1, por lo que no se requieren estados de espera ni READY para los periféricos.

El registro A_{4H} programa las terminales de selección de chip de periférico (PCS6-PCS0), junto con los bits EX y MS del registro A_{8H}. El registro A_{4H} almacena la dirección inicial o base de las líneas de selección del periférico. Los periféricos pueden colocarse en memoria o en el mapa de E/S. Si se ponen en el mapa de E/S, las terminales A₁₉-A₁₆ del número de puerto deben ser 0000. Una vez que se programa la dirección inicial en cualquier límite de direcciones de E/S de 1 Kbyte, las terminales PCS se separan en intervalos de 128 bytes.

Por ejemplo, si el registro A_{4H} se programa con un 0204H, sin estados de espera ni sincronización de READY, la dirección de memoria comienza en 02000H o el puerto de E/S inicia en 2000H. En este caso, los puertos de E/S son: PCS0 = 2000H, PCS1 = 2080H, PCS2 = 2100H, PCS3 = 2180H, PCS4 = 2200H, PCS5 = 2280H y PCS6 = 2300H.

El bit MS del registro A_{8H} selecciona la asignación de memoria o la asignación de E/S para las terminales de selección de periférico. Si MS es un 0 lógico, entonces las líneas PCS se decodifican en el mapa de memoria; si es un 1 lógico, entonces las líneas PCS están en el mapa de E/S.

El bit EX selecciona la función de las terminales PCS5 y PCS6. Si EX = 1, estas terminales PCS seleccionan dispositivos de E/S; si EX = 0, estas terminales proporcionan al sistema las líneas de dirección enclavadas A₁ y A₂. Algunos dispositivos de E/S utilizan las terminales A₁ y A₂ para seleccionar registros internos; es por ello que se proporcionan.

Programación de la unidad de selección de chip para las versiones EB y EC. Como ya mencionó, las versiones EB y EC tienen una unidad de selección de chip distinta. Tales versiones más recientes del 80186/80188 contienen una terminal de selección de chip de memoria superior e inferior como las versiones anteriores, sólo que no traen terminales de selección de memoria media ni de selección de periférico. En vez de estas terminales, las versiones EB y EC contienen ocho terminales de selección de chip generales (GCS7-GCS0), las cuales seleccionan un dispositivo de memoria o un dispositivo de E/S.

La programación también es distinta, ya que cada una de las terminales de selección de chip contiene un registro de dirección inicial y un registro de dirección final. En la figura 16-23 verá la dirección de desplazamiento de cada terminal y el contenido de los registros inicial y final.

La programación para las versiones EB y EC del 80186/80188 es mucho más sencilla que para las versiones XL y EA anteriores. Por ejemplo, para programar la terminal UCS para una dirección que comienza en la posición F0000H y termina en la posición 9FFFFH (64 Kbytes), se programa el registro de dirección inicial (desplazamiento = A_{4H}) con F002H para una dirección inicial de F0000H con dos estados de espera. El registro de dirección final (desplazamiento = A_{6H}) se programa con 000EH para una dirección final de 9FFFFH para la memoria sin sincronización externa de la señal ready (listo). Las demás terminales de selección de chip se programan de forma similar.

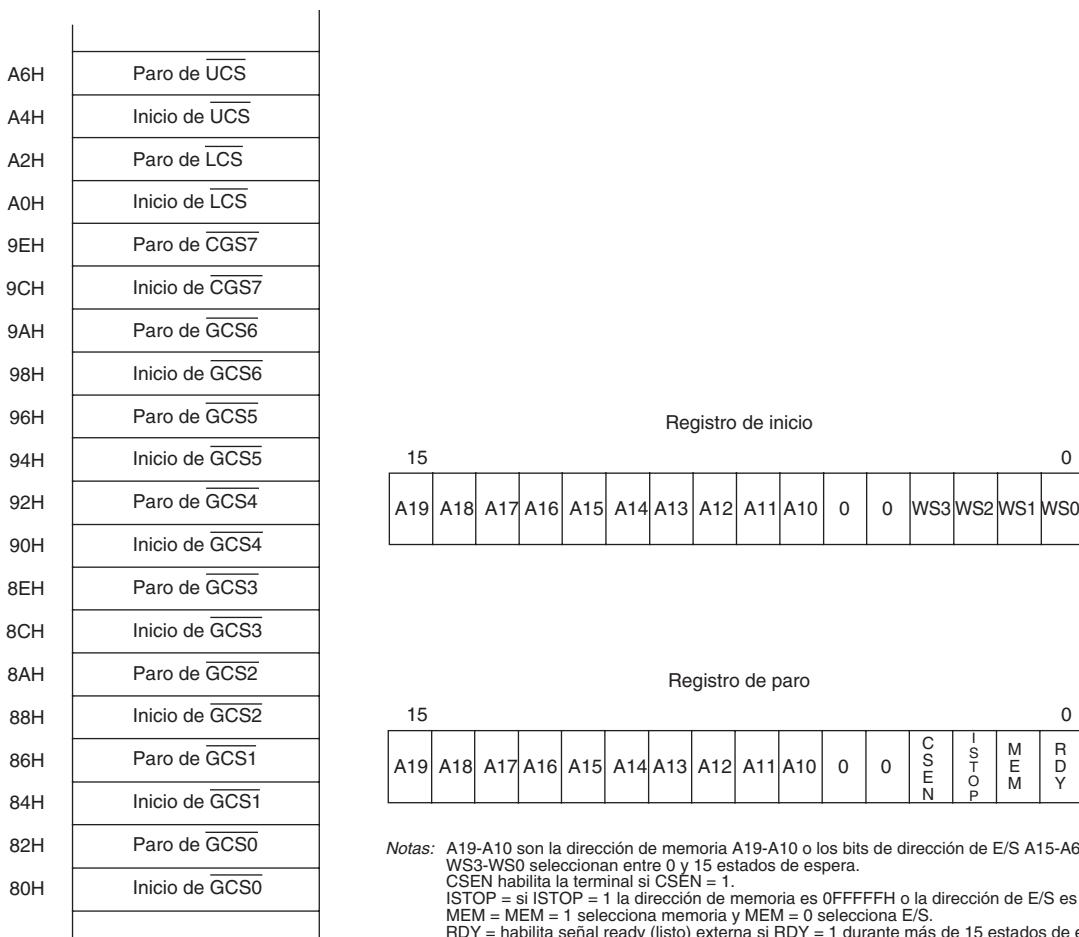


FIGURA 16-23 La unidad de selección de chip en las versiones EB y EC del 80186/80188.

16-3

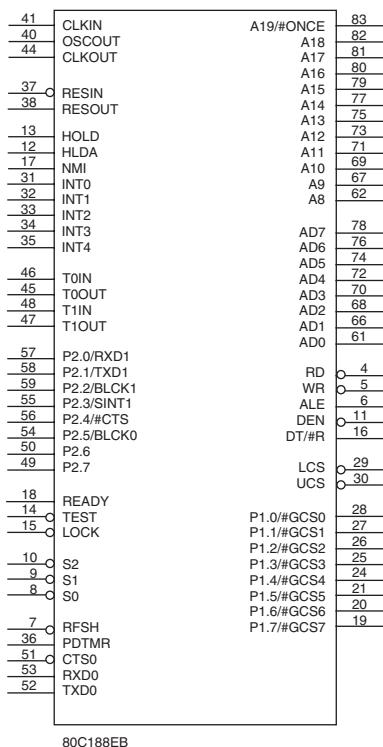
INTERFAZ DE EJEMPLO PARA EL 80C188EB

Como los microprocesadores 80186/80188 fueron diseñados como controladores integrados, en esta sección veremos un ejemplo de dicha aplicación. El ejemplo ilustra una memoria y dispositivos de E/S simples conectados al microprocesador 80C188EB. También lista el software requerido para programar el 80C188EB y sus registros internos después de un reinicio del sistema. La figura 16-24 muestra el diagrama de terminales de la versión 80C188EB del microprocesador 80188. Observe las diferencias entre esta versión y la versión XL que presentamos antes en este capítulo.

La versión 80C188EB contiene algunas nuevas características que no estaban en versiones anteriores. Estas características son: dos puertos de E/S (P_1 y P_2), que se comparten con otras funciones, y dos interfaces de comunicaciones en serie que están integradas en el procesador. A diferencia de la versión XL, esta versión no contiene un controlador de DMA.

El 80188EB puede conectarse con un pequeño sistema diseñado para usarse como entrenador de microprocesadores. El entrenador que se muestra en este texto utiliza una EPROM 27256 para almacenar programas, una SRAM 62256 para almacenar datos y un 8255 para una interfaz de teclado y pantalla LCD. La figura 16-25 muestra un pequeño entrenador de microprocesadores que se basa en el microprocesador 80C188EB.

FIGURA 16-24 Diagrama de terminales de la versión 80C188EB del microprocesador 80188.



La memoria se selecciona mediante la terminal UCS para la EPROM 27C256 y la terminal LCS para la SRAM 62256; la terminal GCS0 selecciona el 8255. El software del sistema coloca la EPROM en las direcciones de memoria F8000H-FFFFFH; la SRAM en 00000H-07FFFH, así como el 8255 en los puertos de E/S 0000H-003FH (el software utiliza los puertos 0, 1, 2 y 3). En este sistema, como es común, no modificamos la dirección del bloque de control de periféricos, el cual reside en los puertos de E/S FF00H-FFFFH.

El ejemplo 16-5 lista el software requerido para inicializar el microprocesador 80C188EB. Este ejemplo programa el 80C188EB por completo y también todo el sistema. En la siguiente sección describiremos el software.

EJEMPLO 16-5

;Sistema simple de prueba de operación en tiempo real para un 80188EB.

```

; INT 40H retrasa para BL milisegundos (rango de 1 a 99)
; INT 41H retrasa para BL segundos (rango de 1 a 59)
; Nota* los tiempos de retraso deben escribirse en hexadecimal !
; por ejemplo, 15 milisegundos es 15H.
; INT 42H muestra cadena de caracteres en LCD
;           ES:BX direcciona la cadena NULL
;           AL = posición (80H línea 1, COH línea 2)
; INT 43H borra la pantalla LCD
; INT 44H lee una tecla del teclado; AL = código de tecla

.MODEL TINY
.186          ;cambia al conjunto de instrucciones del 80186/80188
.CODE
.STARTUP

;programa para el entrenador del microprocesador 80188EB.
;USA MASM 6.11
;línea de comandos = ML /AT NOMBREARCH.ASM

```

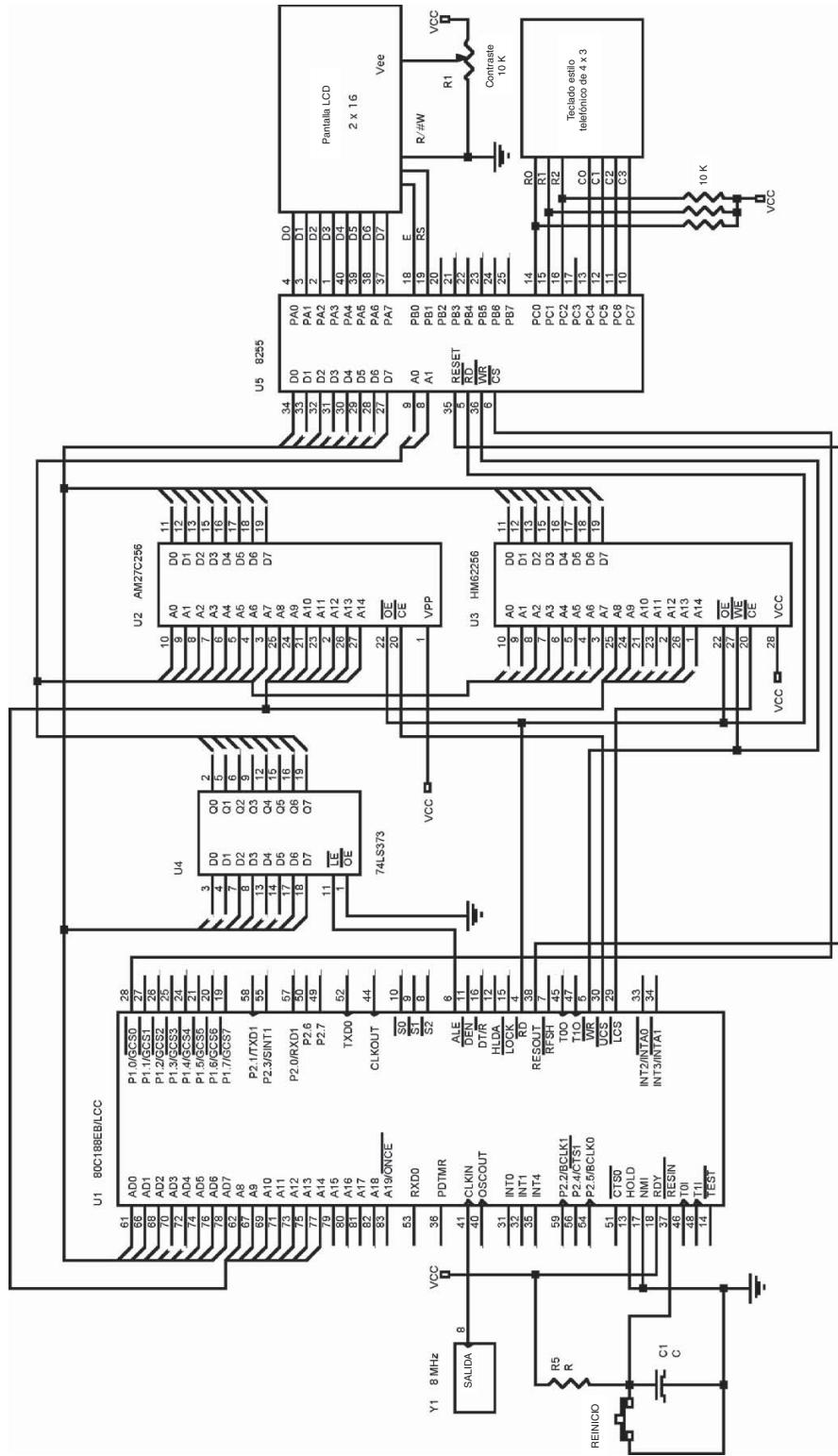


FIGURA 16-25 Un sistema de ejemplo basado en el microprocesador 80C188EB.

```

;:::::::::::;:::::::::::;:::::::::::;:::::::::::;:::::::::::;
;Aquí van las MACROS
;:::::::::::;:::::::::::;:::::::::::;:::::::::::;

IO      MACRO  PUERTO,DATOS
MOV    DX,PUERTO
MOV    AX,DATOS
OUT   DX,AL           ;Al es más eficiente
ENDM

CS_IO  MACRO  PUERTO,INICIO,PARO
IO    PUERTO,INICIO
IO    PUERTO+2,PARO
ENDM

ENVIA MACRO  VALOR,COMANDO,RETRASO
MOV   AL,VALOR
OUT  0,AL
MOV  AL,COMANDO
OUT  1,AL
OR   AL,1
OUT  1,AL
AND  AL,2
OUT  1,AL
PUSH BX
MOV  BL,RETRASO
INT  40H
POP  BX
ENDM

BUT   MACRO
IN   AL,2             ;comprueba si se oprime una tecla
OR   AL,0F8H
CMP  AL,0FFH
ENDM

;:::::::::::;:::::::::::;:::::::::::;:::::::::::;:::::::::::;
;Aquí se coloca la inicialización
;:::::::::::;:::::::::::;:::::::::::;:::::::::::;:::::::::::;

IO      OFFA6H,000EH          ;dirección de paro de UCS
CS_IO  OFFA0H,0,80AH         ;programa LCS
CS_IO  OFF80H,0,48H          ;programa GCS0
IO     OFF54H,1              ;control puerto 1
IO     OFF5CH,0              ;control puerto 2
IO     OFF58H,00FFH          ;dirección puerto 2
IO     3,81H                 ;programa el 8255
MOV   AX,0                  ;direcciona el segmento 0000
MOV   DS,AX                 ;para DS, ES y SS
MOV   ES,AX
MOV   SS,AX
MOV   SP,8000H   ;establece apuntador de la pila (0000:8000)

MOV   BX,OFFSET INTT-100H      ;instala vectores de interrupción
.WHILE WORD PTR CS:[BX] != 0
    MOV   AX,CS:[BX]
    MOV   DI,CS:[BX+2]
    MOV   DS:[DI],AX
    MOV   DS:[DI+2],CS
    ADD   BX,4
.ENDW

MOV   BYTE PTR DS:[40FH],0      ;no muestra la hora
IO    OFF40H,0                ;Conteo del temporizador 2
IO    OFF42H,1000              ;Comparación del temporizador 2
IO    OFF46H,0E001H              ;Control del temporizador 2
IO    OFF08H,00FCH              ;Máscara de interrupción
MOV   AL,0

```

```

        OUT    1,AL           ;coloca E en 0 para LCD
        STI               ;habilita interrupciones
        CALL   INIT          ;inicializa LCD

;:::::::::::::::::::::::::::;AQUÍ VA EL SOFTWARE DEL SISTEMA;::::::::::::::::::
;Las siguientes instrucciones son temporales para evaluar el sistema y
;se sustituyen cuando se crea un nuevo sistema.

        MOV    BYTE PTR DS:[40FH],0FFH      ;se prepara para mostrar la hora
        MOV    WORD PTR DS:[40CH],0          ;pone el reloj en 00:00:00 AM
        MOV    BYTE PTR DS:[40EH],0
        MOV    AX,CS                      ;mensaje de la línea 1
        MOV    ES,AX
        MOV    AL,80H
        MOV    BX,OFFSET MENS1 - 100H
        INT    42H

;Aquí se coloca el software del sistema

        .WHILE 1           ;fin de ciclo del sistema
        .ENDW

;::::::::::::::::::;los procedimientos y los datos van después del software del sistema;::::::::::::::::::
;::::::::::::::::::;MENS1           DB      'El 80188 es el mejor!',0
;::::::::::::::::::;Tabla de vectores de interrupción

INTT    DW      TEM2-100H          ;procedimiento de interrupción
        DW      13H * 4            ;dirección del vector
        DW      RETRASOM-100H
        DW      40H * 4
        DW      RETRASOS-100H
        DW      41H * 4
        DW      CADENA-100H
        DW      42H * 4
        DW      BORRA-100H
        DW      43H * 4
        DW      TECLA-100H
        DW      44H * 4
        DW      0                  ;fin de la tabla

;Procedimiento de servicio de interrupciones para la interrupción del
;TEMPORIZADOR 2 (Una vez por cada milisecondo)

TEM2    PROC   FAR USES ES DS AX BX SI DX

        MOV    AX,0
        MOV    DS,AX
        MOV    ES,AX
        MOV    BX,409H          ;direcciona reloj en tiempo real - 1
        MOV    SI,OFFSET MODU-101H ;direcciona tabla módulo - 1
        .REPEAT
                INC    SI           ;apunta al módulo
                INC    BX           ;apunta al contador
                MOV    AL,[BX]        ;obtiene el contador
                ADD    AL,1          ;le suma 1
                DAA
                .IF AL == BYTE PTR CS:[SI] ;comprueba módulo
                        MOV    AL,0
                .ENDIF
                MOV    [BX],AL         ;guarda nuevo conteo
        .UNTIL !ZERO? || BX = 40FH
        IO    OFF02H,8000H       ;fin de interrupción
        .IF    BYTE PTR DS:[40AH] == 0 && BYTE PTR DS:[40BH] == 0
                CALL  MUESTRA      ;inicia subprocesso MUESTRA

```

```

        .ENDIF
        IRET

TEM2    ENDP

MODU    DB      0          ;Mod 100
        DB      10H       ;Mod 10
        DB      60H       ;Mod 60
        DB      60H       ;Mod 60
        DB      24H       ;Mod 24

MUESTRA PROC   NEAR      ;muestra la hora del dia (una vez por segundo)
        .IF    BYTE PTR DS:[40FH] != 0      ;si el tiempo a mostrar no es cero
            STI
            MOV   BX,3F0H
            MOV   SI,40EH
            MOV   AL,[SI]
            .IF  AL > 12H           ;direcciona el reloj
                SUB  AL,12H
                DAS
            .ELSEIF AL == 0          ;obtiene las horas
                MOV  AL,12H
            .ENDIF
            CALL GUARDA
            MOV  BYTE PTR [BX],':'
            INC  BX
            MOV  AL,[SI]           ;obtiene los minutos
            CALL GUARDA
            MOV  BYTE PTR [BX],':'
            INC  BX
            MOV  AL,[SI]           ;obtiene los segundos
            CALL GUARDA
            MOV  DL,'A'            ;para AM / PM
            .IF  BYTE PTR DS:[40EH] > 11H
                MOV  DL,'P'
            .ENDIF
            MOV  BYTE PTR [BX],` '
            MOV  [BX+1],DL
            MOV  BYTE PTR [BX+2],'M'
            MOV  BYTE PTR [BX+3],0      ;fin de la cadena
            MOV  BX,3F0H
            MOV  AL,0C2H           ;muestra el búfer
            INT  42H               ;posición inicial de LCD
        .ENDIF
        RET

MUESTRA ENDP

GUARDA PROC   NEAR
        PUSH AX
        SHR  AL,4
        MOV  DL,AL
        POP  AX
        AND  AL,15
        MOV  DH,AL
        ADD  DX,3030H
        MOV  [BX],DX
        ADD  BX,2
        DEC  SI
        RET

GUARDA ENDP

RETRASOS PROC  FAR USES DS BX AX
        STI                  ;habilita interrupciones futuras
        MOV  AX,0
        MOV  DS,AX

```

```

MOV     AL, DS:[40H]           ;obtiene contador de milisegundos
ADD     AL, BL                 ;BL = núm. de milisegundos
DAA
.REPEAT
.UNTIL AL == DS:[40AH]
IRET

RETRASOS ENDP

RETRASOS PROC FAR USES DS BX AX

STI          ;habilita interrupciones futuras
MOV     AX, 0
MOV     DS, AX
MOV     AL, DS:[40CH]           ;obtiene los segundos
ADD     AL, BL
DAA
.IF AL >= 60H
    SUB     AL, 60H
    DAS
.ENDIF
.REPEAT
.UNTIL AL == DS:[40CH]
IRET

RETRASOS ENDP

INIT      PROC NEAR

MOV     BL, 30H                ;espera 30 milisegundos
INT     40H
MOV     CX, 4
.REPEAT
    ENVIA   38H, 0, 6
.UNTILCXZ
ENVIA   8, 0, 2
ENVIA   1, 0, 2
ENVIA   12, 0, 2
ENVIA   6, 0, 2
RET

INIT_LCD ENDP

CADENA   PROC FAR USES BX AX      ;muestra la cadena

STI          ;habilita interrupciones futuras
ENVIA   AL, 0, 1                ;envía posición inicial
.REPEAT
    ENVIA   BYTE PTR ES:[BX], 2, 1
    INC     BX
.UNTIL BYTE PTR ES:[BX] == 0
IRET

CADENA   ENDP

BORRA    PROC FAR USES AX BX      ;borra LCD

STI          ;habilita interrupciones futuras
ENVIA   1, 0, 2
IRET
BORRA    ENDP

TECLA    PROC FAR USES BX      ;lee tecla

STI
MOV     AL, 0                  ;borra de C0 a C3
OUT    2, AL
.REPEAT
    .REPEAT
        BUT
.UNTIL ZERO?

```

```

        MOV    BL,12H           ;retraso de tiempo
        INT    40H
        BUT
.UNTIL ZERO?
.REPEAT          ;espera a que se oprima una tecla
.REPEAT
        BUT
.UNTIL !ZERO?
MOV    BL,12H           ;retraso de tiempo
INT    40H
BUT
.UNTIL !ZERO?
MOV    BX,0FDEFH
.REPEAT
        MOV    AL,BL
        OUT   2,AL
        ADD   BH,3
        ROL   BL,1
        BUT
.UNTIL !ZERO?
.WHILE 1
        SHR   AL,1
        .BREAK .IF !CARRY?
        INC   BH
.ENDW
MOV    BX,OFFSET BUSCA-100H
XLAT   CS:BUSCA          ;especifica el segmento de código (EPROM)

IRET

TECLA  ENDP

BUSCA  DB    3,2,1
       DB    6,5,4
       DB    9,8,7
       DB    10,0,11

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;CUALQUIER OTRA SUBRUTINA O DATOS QUE SE REQUIERAN DEBEN IR AQUÍ
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

ORG    080F0H           ;llega a posición de reinicio
REINICIO:
        IO    OFFA4H,0F800H      ;dirección inicial UCS
        DB    0EAH
        DW    0000H,0F800H      ;JMP    F800:0000          (F8000H)
END

```

16-4**SISTEMAS OPERATIVOS EN TIEMPO REAL (RTOS)**

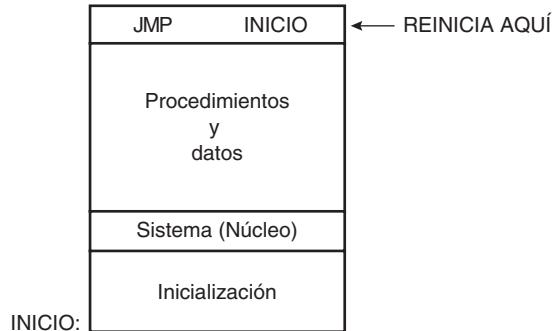
En esta sección describiremos el funcionamiento del sistema operativo en tiempo real (RTOS). Las interrupciones se utilizan para desarrollar el RTOS, debido a que se utilizan en aplicaciones integradas del microprocesador. Todos los sistemas, desde la aplicación integrada más simple hasta el sistema más sofisticado, deben tener un sistema operativo.

¿Qué es un sistema operativo en tiempo real (RTOS)?

El RTOS es un sistema operativo que se utiliza en aplicaciones integradas que realizan tareas en una cantidad predecible de tiempo. Los sistemas operativos tales como Windows aplazan muchas tareas y no garantizan su ejecución en un tiempo predecible. El RTOS es muy parecido a cualquier otro sistema operativo, en cuanto a que contiene las mismas secciones básicas. La figura 16-26 ilustra la estructura básica de la manera en que un sistema operativo podría colocarse en una EPROM o en un dispositivo de memoria Flash.

FIGURA 16-26

La estructura de un sistema operativo RTOS.



Hay tres componentes para todos los sistemas operativos: (1) inicialización, (2) el núcleo, (3) datos y procedimientos. Si el ejemplo 16-5 (sección anterior) se compara con la figura 16-26, se verán las tres secciones. La sección de inicialización se utiliza para programar todos los componentes de hardware en el sistema, para cargar controladores específicos para un sistema y programar el contenido de los registros del microprocesador. El núcleo realiza la tarea básica del sistema, proporciona las llamadas o funciones al sistema y constituye el sistema integrado. La sección de datos y procedimientos almacena todos los procedimientos y datos estáticos que emplee el sistema operativo.

La sección REINICIO. La última parte del software del ejemplo 16-5 muestra el bloque de reinicio del RTOS. La instrucción ORG coloca las instrucciones de reinicio en una posición que está a 16 bytes del final del dispositivo de memoria. En este caso, la EPROM es de 32 Kbytes, lo cual significa que empieza en 0000H y termina en 7FFFH. Recuerde que un dispositivo de 32 Kbytes tiene 15 terminales de dirección. La entrada \overline{CS} selecciona el EPROM para las posiciones F8000H a FFFFFH en el sistema. La instrucción ORG en el programa coloca el origen de la sección de reinicio en la posición 80F0H, ya que todos los programas del modelo diminuto (.COM) se ensamblan desde la dirección de desplazamiento 100H, aun y cuando el primer byte del programa sea el primer byte almacenado en el archivo. Debido a tal desviación, todas las direcciones en la EPROM deben ajustarse por 100H, como la instrucción ORG.

Sólo hay 16 bytes de memoria para la instrucción de reinicio, ya que la posición de reinicio es FFFF0H en el sistema. En este ejemplo sólo hay suficiente espacio para programar la dirección inicial de UCS como F8000H antes de un salto hacia el inicio de la EPROM. Como no se permiten saltos lejanos en el modelo diminuto, la solución fue almacenar el código de operación hexadecimal actual para un salto lejano (EAH).

Sección de inicialización. Esta sección en el ejemplo 16-5 empieza en el bloque de reinicio y continua al inicio de la EPROM. Si se ve la sección de inicialización, se programan todos los dispositivos programables en el sistema y se cargan los registros de segmento. La sección de inicialización también programa el temporizador 2, de manera que produzca una interrupción para el procedimiento TIM₂ cada milisecondo. El procedimiento de servicio de interrupciones de TIM₂ actualiza el reloj una vez cada segundo y es también la base de los retrasos de tiempo precisos en el software.

El núcleo. El núcleo en el ejemplo 16-5 es muy corto, debido a que el sistema está incompleto y sólo sirve como sistema de prueba. En este ejemplo, todo lo que el sistema hace es mostrar un mensaje de registro y la hora del día en la segunda línea de la pantalla LCD. Una vez que se logra esto, el sistema entra en un ciclo WHILE infinito. Todos los programas de sistema son ciclos infinitos, a menos que fallen.

Un sistema de ejemplo

La figura 16-27 muestra un sistema integrado simple basado en el microprocesador integrado 80188EB. Este diagrama esquemático sólo ilustra las partes que se agregaron a la figura 16-25 para leer una temperatura del LM-70. Este sistema contiene una pantalla LCD de 2 líneas × 16 caracteres por línea, la cual muestra la hora del día y la temperatura. El sistema se almacena en una EPROM pequeña de 32 K x 8. Se incluye una SRAM de 32 K x 8 para que actúe como pila y almacene la hora. Una base de datos almacena las temperaturas más recientes y las horas a las que se obtuvieron.

FIGURA 16-27 Circuitos adicionales para la figura 16-25, de manera que permita leer una temperatura.

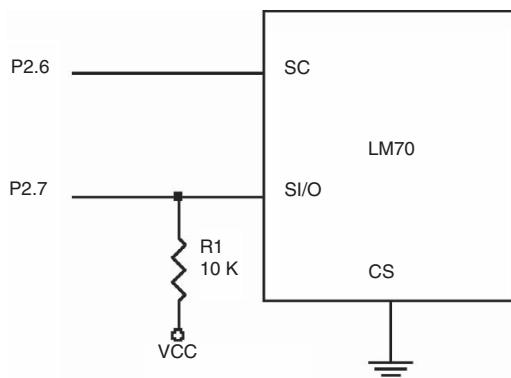
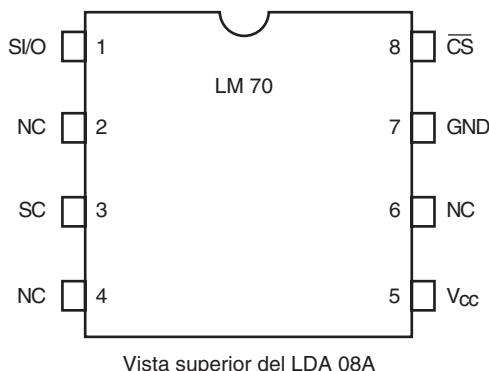


FIGURA 16-28 El sensor de temperatura LM₇₀.



El sensor de temperatura se encuentra dentro del sensor de temperatura digital LM₇₀, fabricado por National Semiconductor Corporation por menos de US\$1.00. La interfaz para el microprocesador está en formato serial y el convertidor tiene una resolución de 10 bits más un bit de signo. La figura 16-28 muestra el diagrama de terminales del sensor de temperatura LM₇₀.

El LM₇₀ transfiere datos hacia y desde el microprocesador a través de la terminal SIO, la cual es una terminal de datos en serie bidireccional. La información se controla a través de la terminal SIO mediante la terminal SC (reloj). El LM₇₀ contiene tres registros de 16 bits: el registro de configuración, el registro del sensor de temperatura y el registro de identificación. El registro de configuración selecciona el modo de apagado (XXFFH) o el modo de conversión continua (XX00). El registro de temperatura contiene la temperatura con signo en los 11 bits de más a la izquierda de la palabra de datos de 16 bits. Si la temperatura es negativa, se encuentra en sus formas de complemento respectivas. El registro de identificación presenta un 8100 cuando se lee.

Cuando se lee la temperatura del LM₇₀, se lee en grados Celsius; cada intervalo equivale a 0.25 °C. Por ejemplo, si el registro de temperatura es 0000 1100 100X XXXX, o un valor de 100 decimal, la temperatura es de 25 °C.

El ejemplo 16-6 muestra el software que se agrega al sistema operativo listado en el ejemplo 16-5. El sistema muestrea las temperaturas una vez por minuto y las almacena en una cola circular junto con el día y el tiempo en horas y minutos. El día es un número que empieza en cero cuando se inicializa el sistema. El tamaño de la cola se ha establecido en 16 Kbytes, de manera que logren almacenarse las 4,096 mediciones más recientes. En este ejemplo no se utiliza el teclado, pero algunas de las llamadas al sistema se emplean para mostrar la temperatura en la línea 1 de la pantalla. El reloj en tiempo real también se interroga para determinar el inicio de cada minuto, de manera que sea posible tomar una muestra. El software en el listado sustituye a la sección de software del ejemplo 16-5 en donde dice: “;Aquí va el software del sistema”. Este software sustituye el ciclo WHILE infinito en el ejemplo.

Para inicializar el LM₇₀ se envían 16 bits de 0s a este sensor y después se leen los 16 bits de la temperatura. La lectura del LM₇₀ se lleva a cabo en el software mediante el procedimiento TEMP, en tanto que la inicialización está a cargo del procedimiento INITT.

EJEMPLO 16-6

;Software del sistema que lee la temperatura una vez por minuto y la registra en una cola que está en 0500H - 44FFH.

```

MOV WORD PTR DS:[4FCH],500H ;apuntador cola entrada = 500H
MOV WORD PTR DS:[4FEH],500H ;apuntador cola salida = 500H
CALL INITT ;inicializa el LM70
.while 1
    .IF DS:[40CH] == 0 && DS:[40BH] == 0 && DS:[40AH] == 0
        CALL TEMP ;una vez por minuto
        CALL ENCOLA ;temperatura en la cola
        CALL MTEMP ;muestra la temperatura
    .ENDIF
.endw

INITT PROC NEAR ;envía 0000H a LM70 para reiniciarlo
    IO 0FF58H,003FH ;p2.6 y p2.7 se ajustan como salidas
    MOV CX,16 ;conteo de bits hasta 16
    MOV DX,0FF5EH ;direcciona enclavamiento del puerto 2
    .REPEAT
        MOV AL,40H
        OUT DX,AL
        MOV AL,0
        OUT DX,AL
    .UNTILCXZ
    RET

INITT ENDP

TEMP PROC NEAR ;lee la temperatura
    IO 0FF58H,00BFH ;p2.7 como entrada
    MOV CX,16
    MOV BX,0
    .REPEAT ;obtiene todos los 16 bits
        IO 0FF5EH,0C0H
        MOV DX,0FF5AH
        IN AL,DX ;lee bit
        SHR AL,1
        RCR BX,1 ;en BX
        IO 0FF5EH,40H
    .UNTILCXZ
    MOV AX,BX
    SAR AX,6 ;convierte en valor entero de temperatura
    RET

TEMP ENDP

ENCOLA PROC NEAR USES AX ;mete temperatura en cola, no revisa si está llena
    MOV BX,DS:[4FCH]
    MOV [BX],AX ;almacena temp
    ADD BX,2
    MOV AX,DS:[40DH] ;obtiene hora HH:MM
    MOV [BX],AX
    ADD BX,2
    .IF BX == 4500H
        MOV BX,500H
    .ENDIF
    RET

ENCOLA ENDP

MTEMP PROC
    MOV BX,410H ;direcciona búfer de cadena
    OR AX,AX
    .IF SIGN? ;si es negativo
        MOV BYTE PTR [BX], '-'
    .ENDIF
    RET

```

```

        NEG    AX
        INC    BX
.ENDIF
AAM           ; convierte a BCD
.IF AH != 0
        ADD    AH, 30H
        MOV    [BX], AH
        INC    BX
.ENDIF
ADD    AL, 30H
MOV    [BX], AL
INC    BX
MOV    BYTE PTR [BX], ' '
MOV    BYTE PTR [BX+1], 'C'
MOV    AX, DS
MOV    ES, AX
MOV    AL, 86H
MOV    BX, 410H
INT    42H           ; muestra temp en linea 1

MTEMP    ENDP

```

Un sistema con subprocessamiento

En ocasiones se requiere un sistema operativo que procese varios subprocessos. El núcleo maneja subprocessos múltiples mediante el uso de una interrupción en tiempo real. Un método para programar los procesos en un pequeño RTOS es el uso de una partición de tiempo para alternar entre varios procesos. La partición básica de tiempo puede ser de cualquier duración, lo cual depende en parte de la velocidad de ejecución del microprocesador. Por ejemplo, en un sistema con microprocesador moderno con un reloj de 100 MHz, se ejecutarán muchas instrucciones en uno o dos ciclos de reloj. Si suponemos que el equipo ejecuta una instrucción cada dos ciclos del reloj y elegimos una partición de tiempo de 1 ms, el equipo ejecutará una cantidad aproximada de 50,000 instrucciones por cada partición de tiempo, lo cual sería adecuado para la mayoría de los sistemas. Si se emplea una frecuencia de reloj más baja, entonces se selecciona una partición de tiempo de 10 ms o incluso de 100 ms.

Cada partición de tiempo se activa mediante la interrupción de un temporizador. El procedimiento de servicio de interrupciones debe buscar en la cola para determinar si hay una tarea disponible para ejecutarse; si la hay, debe empezar la ejecución de la nueva tarea. Si no hay una nueva tarea, tiene que continuar ejecutando las tareas anteriores o entrar en un estado de inactividad y esperar a que se ponga una nueva tarea en la cola. Ésta es circular y puede contener cualquier número de tareas para el sistema, hasta cierto límite finito. Por ejemplo, ser una cola pequeña en un sistema pequeño con 10 entradas. El tamaño se determina con base en las necesidades del sistema en general, pero puede aumentarse o disminuirse.

Cada una de las entradas de programación de la cola contiene un apuntador hacia el proceso (CS: IP) y a todo el estado de contexto del equipo. Las entradas de programación de la cola también contienen cierta forma de entrada de tiempo de vida en caso de un interbloqueo, una entrada de prioridad y una entrada para extender el tiempo de activación de la partición. En el siguiente ejemplo no se utilizará una entrada de prioridad o una entrada para extender la cantidad de particiones de tiempo consecutivas permitidas para un programa. El núcleo atenderá los procesos sólo en forma lineal o en forma de esfera circular (Round Robin), a medida que vayan llegando de la cola.

Para implementar un programador para el sistema integrado, se utilizan procedimientos o macros para iniciar una nueva aplicación, para eliminar la aplicación cuando se complete y pausar una aplicación cuando requiera tiempo para acceder a la E/S. Cada una de estas macros accede a una cola de programación ubicada en el sistema de memoria, en una dirección disponible tal como 0500H. La cola de programación utiliza la estructura de datos del ejemplo 16-7 para facilitar la creación de la cola, además de que tendrá espacio para 10 entradas. Esta cola de programación nos permite iniciar hasta 10 procesos a la vez.

EJEMPLO 16-7

PRESENTE	DB	0	; 0 = no está presente
SENUEL01	DB	?	

```

RAX      DW    ?
RBX      DW    ?
RCX      DW    ?
RDX      DW    ?
RSP      DW    ?
RBP      DW    ?
RSI      DW    ?
RDI      DW    ?
RFLAG    DW    ?
RIP      DW    ?
RCS      DW    ?
RDS      DW    ?
RES      DW    ?
RSS      DW    ?
SENUELO2 DW    ?           ;relleno para 32 bytes

```

La estructura de datos del ejemplo 16-7 se copia 10 veces en memoria para completar la estructura de la cola durante la inicialización del sistema; por ende, no contiene procesos activos al momento de inicializarse. También necesitamos un apuntador para la cola que se inicialice con 500H. En este ejemplo, el apuntador a la cola se almacena en la posición 4FEH. El ejemplo 16-8 proporciona una de las posibles inicializaciones. Aquí se almacena la estructura de datos en la RAM con 10 copias, empezando en la posición 500H. Este software hace la suposición de que un reloj de sistema de 32 MHz opera el temporizador 2 que se utiliza como preescalador para dividir la entrada del reloj (el reloj de sistema dividido entre 8) de 4 MHz entre 40,000. Esto hace que la salida del temporizador 2 sea 1 KHz (1.0 ms). El temporizador 1 se programa para dividir la señal de reloj del temporizador 2 entre 10 para generar una interrupción cada 10 ms.

EJEMPLO 16-8

;inicialización de la cola de subprocessos

```

PUSH   DS
MOV    AX, 0
MOV    DS, AX
MOV    SI, 500H
MOV    BX, OFFSET PRESENTE-100H    ;vacía todas las entradas de la cola
MOV    CX, 10
.REPEAT
    MOV   BYTE PTR DS:[SI], 0
    ADD   SI, 32
.UNTILCXZ
MOV   DS:[4FEH], 500H          ;establece el apuntador a la cola
POP   DS
MOV   DX, OFF42H              ;CMPA del temporizador 2 = 40000
MOV   AX, 40000
OUT   DX, AL
MOV   DX, OFF32H              ;CMPA del temporizador 1 = 10
MOV   AX, 10
OUT   DX, AL
MOV   AX, 0                   ;Borra los registros de conteo del
                               ;temporizador
MOV   DX, OFF30H
OUT   DX, AL
MOV   DX, OFF40H
OUT   DX, AL
MOV   DX, OFF46H
MOV   AX, 0C001H              ;Inicia el temporizador 2
OUT   DX, AL
MOV   DX, OFF36H              ;Inicia el temporizador 1
MOV   AX, 0E009H
OUT   DX, AL

```

El procedimiento NUEVO (que en el ejemplo 16-9 se instala en INT 60H) agrega un proceso a la cola. Busca a través de las 10 entradas hasta que encuentra un cero en el primer byte (PRESENTE),

el cual indica que la entrada está vacía. Si encuentra una entrada vacía, coloca la dirección inicial del proceso en RCS y RIP, así como un 0200H en la posición RFLAG. Un 200H en RFLAG asegura que la interrupción esté habilitada cuando inicie el proceso, lo cual evita que el sistema falle. El procedimiento NUEVO esperará (en caso de que ya haya 10 procesos programados) hasta que termine un proceso. A cada uno de los procesos se le asigna también espacio en la pila, en secciones de 256 bytes que comienzan en la dirección 7600H, de manera que el proceso más bajo tenga el espacio de pila 7500H-75FFH, que el siguiente proceso tenga el espacio de pila 7600H-76FFH, y así sucesivamente. Un algoritmo administrador de memoria podría encargarse de la asignación de un área de la pila.

EJEMPLO 16-9

```

INT60    PROC    FAR USES DS AX DX SI

        MOV     AX,0
        MOV     DS,AX          ;direcciona el segmento 0000
        STI               ;en interrupción

        .REPEAT           ;se queda aquí si está llena
        MOV     SI,500H
        HLT               ;se sincroniza con la interrupción RTC
        .WHILE BYTE PTR DS:[SI] != 0 && SI != 660H
        ADD     SI,32
        .ENDW
        .UNTIL BYTE PTR DS:[SI] == 0

        MOV     BYTE PTR DS:[SI],0FFH ;activa el proceso
        MOV     WORD PTR DS:[SI+18],200H ;banderas
        MOV     DS:[SI+20],BX          ;almacena IP
        MOV     DS:[SI+22],DX          ;almacena CS
        MOV     DS:[SI+28],SS          ;almacena SS
        MOV     AX,SI
        AND     AX,3FFH
        SHL     AX,3
        ADD     AX,7500H
        MOV     DS:[SI+10],AX         ;almacena SP
        IRET

INT60    ENDP

```

El procedimiento de control final (KILL), que se ubica en el vector de interrupción 61H como se muestra en el ejemplo 16-10, elimina una aplicación al colocar 00H en PRESENTE de la estructura de datos tipo cola, con lo cual se extrae de la cola de programación.

EJEMPLO 16-10

```

INT61    PROC    FAR USES DS AX DX SI

        MOV     AX,0
        MOV     DS,AX
        MOV     SI,DS:[4FEH]       ;obtiene apuntador a la cola
        MOV     BYTE PTR DS:[SI],0  ;elimina el subproceso
        JMP     INT12A

INT61    ENDP

```

El procedimiento PAUSA es sólo una llamada al procedimiento de partición de tiempo (INT 12H) que se libera del proceso y devuelve el control al procedimiento de partición de tiempo, con lo que se termina de manera prematura la partición de tiempo para el proceso. Esta *liberación temprana* permite que continúen otros procesos antes de regresar al proceso actual.

En el ejemplo 16-11 aparece el procedimiento de servicio de interrupciones de la partición de tiempo para un 80188EB que utiliza una partición de tiempo de 10 ms. Como éste es un procedimiento de servicio de interrupciones, hemos tenido cuidado de hacerlo lo más eficiente posible. El ejemplo 16-11 muestra el procedimiento de partición de tiempo que se encuentra en el vector 12H para operar con el temporizador 1 en el microprocesador 80188EB. Aunque no se muestra, este software

supone que el temporizador 2 se utiliza como preescalador, en tanto el temporizador 1 emplea la señal del temporizador 2 para generar la interrupción de 10 ms. El software también supone que no se está manejando ninguna otra interrupción en el sistema.

EJEMPLO 16-11

```

INT12    PROC    FAR USES DS AX DX SI
          MOV     AX,0
          MOV     DS,AX      ;direcciona el segmento 0000
          MOV     SI,DS:[4FEH] ;obtiene el apuntador a la cola
          CALL   GUARDAE    ;almacena el estado
INT12A:
          ADD     SI,32      ;obtiene el siguiente proceso
          .IF SI == 660H    ;hace la cola circular
          MOV     SI,500H
          .ENDIF
          .WHILE BYTE PTR DS:[SI] != 0FFH ;busca el siguiente proceso
          ADD     SI,32
          .IF SI == 660H
          MOV     SI,500H
          .ENDIF
          .ENDW
          MOV     DX,0FF30H    ;obtiene una partición de tiempo completa de 10 ms
          MOV     AX,0
          OUT    DX,AL
          MOV     DX,0FF02H    ;borra la interrupción
          MOV     AX,8000H
          OUT    DX,AX
          JMP    CARGAE      ;carga el estado del siguiente proceso
INT12    ENDP

```

El arranque del sistema (que se coloca después de la inicialización del mismo) debe ser un proceso individual dentro de un ciclo de espera infinito. El arranque se muestra en el ejemplo 16-12.

EJEMPLO 16-12

ARRANQUE_SISTEMA:

```

;bifurca el subprocesso EESPERA
          MOV     BX,OFFSET EESPERA-100H ;dirección de desplazamiento del subprocesso
          MOV     DX,0F800H      ;dirección de segmento del subprocesso
          INT    60H           ;inicia subprocesso de espera

;otros procesos del sistema se inician aquí

EESPERA:          ;ciclo inactivo del sistema
          .WHILE 1
          INT 21H        ;se libera
          .ENDW

```

Por último, los procedimientos GUARDAE y CARGAE se utilizan para cargar y almacenar el contexto del equipo, a medida que se realiza el cambio de un proceso a otro. Estos procedimientos se llaman con base en la interrupción de la partición de tiempo (INT 12H) e inician un proceso (INT 60H), como se muestra en el ejemplo 16-13.

EJEMPLO 16-13

```

GUARDAE  PROC    NEAR
          MOV     DS:[SI+4],BX      ;almacena BX
          MOV     DS:[SI+6],CX      ;almacena CX
          MOV     DS:[SI+10],SP     ;almacena SP
          MOV     DS:[SI+12],BP     ;almacena BP
          MOV     DS:[SI+16],DI     ;almacena DI

```

```

        MOV    DS:[SI+26],ES      ;almacena ES
        MOV    DS:[SI+28],SS      ;almacena SS
        MOV    BP,SP              ;obtiene SP
        MOV    AX,[SP+2]           ;obtiene SI
        MOV    DS:[SI+14],AX      ;almacena SI
        MOV    AX,[BP+4]           ;obtiene DX
        MOV    DS:[SI+8],AX      ;almacena DX
        MOV    AX,[BP+6]           ;obtiene AX
        MOV    DS:[SI+2],AX      ;almacena AX
        MOV    AX,[BP+8]           ;obtiene DS
        MOV    DS,[SI+24],AX      ;almacena DS
        MOV    AX,[BP+10]          ;obtiene las banderas
        MOV    DS:[SI+18],AX      ;almacena las banderas
        MOV    AX:[BP+12]          ;obtiene CS
        MOV    DS:[SI+22],AX      ;almacena CS
        MOV    AX,[BP+14]          ;obtiene IP
        MOV    DS:[SI+20],AX      ;almacena IP
        RET

GUARDAE ENDP

CARGAE PROC FAR

        MOV    SS,DS:[SI+28]       ;obtiene SS
        MOV    SP,DS:[SI+10]        ;obtiene SP
        PUSH   WORD PTR DS:[SI+20]  ;PUSH IP
        PUSH   WORD PTR DS:[SI+22]  ;PUSH CS
        PUSH   WORD PTR DS:[SI+18]  ;PUSH banderas
        PUSH   WORD PTR DS:[SI+24]  ;PUSH DS
        PUSH   WORD PTR DS:[SI+2]   ;PUSH AX
        PUSH   WORD PTR DS:[SI+8]   ;PUSH DX
        PUSH   WORD PTR DS:[SI+14]  ;PUSH SI
        MOV    BX,DS:[SI+4]          ;obtiene BX
        MOV    CX,DS:[SI+6]          ;obtiene CX
        MOV    BP,DS:[SI+12]         ;obtiene BP
        MOV    DI,DS:[SI+16]         ;obtiene DI
        MOV    ES,DS:[SI+26]         ;obtiene ES
        POP    SI
        POP    DX
        POP    AX
        POP    DS
        IRET

CARGAS ENDP

```

16-5**INTRODUCCIÓN AL 80286**

El microprocesador es una versión avanzada del microprocesador 8086 que se diseñó para entornos multiusuario y multitarea. El 80286 direcciona 16 Mbytes de memoria física y 1 Gbyte de memoria virtual mediante el uso de su sistema de administración de memoria. En esta sección del texto presentaremos el microprocesador 80286, que se utilizó en las primeras computadoras personales estilo AT, que una vez invadieron el mercado de las computadoras y hoy aún siguen teniendo aplicación. En esencia, el 80286 es un 8086 optimizado para ejecutar instrucciones en menos períodos de reloj. El 80286 también es una versión mejorada del 8086, ya que contiene un administrador de memoria. En este momento, el 80286 ya no ocupa un lugar en el sistema de computadora personal, pero sí tiene aplicación en sistemas de control, como controlador integrado.

Características de hardware

La figura 16-29 muestra el diagrama de bloques interno del microprocesador 80286. Al igual que el 80186/80188, el 80286 no incorpora periféricos internos; en vez de ello, contiene una unidad de administración de memoria (MMU) conocida como **unidad de dirección** en el diagrama de bloques.

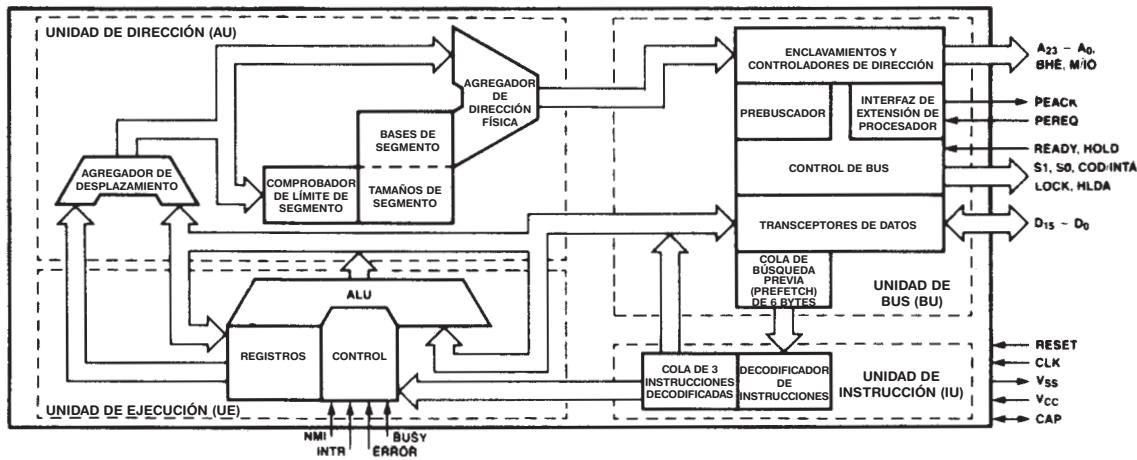


FIGURA 16-29 Diagrama de bloques del microprocesador 80286. (Cortesía de Intel Corporation.)

Como lo revela un cuidadoso análisis del diagrama de bloques, las terminales de dirección A₂₃-A₀, BUSY, CAP, ERROR, PEREQ y PEACK son terminales nuevas o adicionales que no aparecen en el microprocesador 8086. Las señales BUSY, ERROR, PEREQ y PEACK se utilizan con la extensión del microprocesador o coprocesador (por ejemplo, el 80287). Ahora la terminal TEST se conoce como terminal BUSY. El bus de direcciones es de 24 bits para manejar los 16 Mbytes de memoria física. La terminal CAP se conecta a un capacitor de 0.047 µF, ±20%, que actúa como un filtro de 12 V y se conecta a tierra. Los diagramas de terminales del 8086 y del 80286 se muestran en la figura 16-30 para fines comparativos. El 80286 no contiene un bus de direcciones/datos multiplexado.

FIGURA 16-30 Diagramas de terminales de los microprocesadores 8086 y 80286. El 80286 no tiene un bus de direcciones/datos multiplexado.

<table border="1"> <thead> <tr> <th>33</th> <th>MN</th> <th>AD0</th> <th>16</th> <th>63</th> <th>READY</th> <th>A0</th> <th>34</th> </tr> <tr> <th>22</th> <th>READY</th> <th>AD1</th> <th>15</th> <th>31</th> <th>CLK</th> <th>A1</th> <th>33</th> </tr> <tr> <th>19</th> <th>CLK</th> <th>AD2</th> <th>14</th> <th>29</th> <th>RESET</th> <th>A2</th> <th>32</th> </tr> <tr> <th>21</th> <th>RESET</th> <th>AD3</th> <th>13</th> <th>5</th> <th>S0</th> <th>A3</th> <th>28</th> </tr> <tr> <th>18</th> <th>INTR</th> <th>AD4</th> <th>12</th> <th>4</th> <th>S1</th> <th>A4</th> <th>27</th> </tr> <tr> <td></td> <td></td> <th>AD5</th> <th>11</th> <td></td> <td></td> <th>A5</th> <th>26</th> </tr> <tr> <td></td> <td></td> <th>AD6</th> <th>10</th> <td></td> <td></td> <th>A6</th> <th>25</th> </tr> <tr> <td></td> <td></td> <th>AD7</th> <th>9</th> <td></td> <td></td> <th>A7</th> <td>24 </td></tr> <tr> <td></td> <td></td> <th>AD8</th> <th>8</th> <td></td> <td></td> <th>A8</th> <td>23 </td></tr> <tr> <td></td> <td></td> <th>AD9</th> <th>7</th> <td></td> <td></td> <th>A9</th> <td>22 </td></tr> <tr> <td></td> <td></td> <th>AD10</th> <th>6</th> <td></td> <td></td> <th>A10</th> <td>21 </td></tr> <tr> <td></td> <td></td> <th>AD11</th> <th>5</th> <td></td> <td></td> <th>A11</th> <td>20 </td></tr> <tr> <td></td> <td></td> <th>AD12</th> <th>4</th> <td></td> <td></td> <th>A12</th> <td>19 </td></tr> <tr> <td></td> <td></td> <th>AD13</th> <th>3</th> <td></td> <td></td> <th>A13</th> <td>18 </td></tr> <tr> <td></td> <td></td> <th>AD14</th> <th>2</th> <td></td> <td></td> <th>A14</th> <td>17 </td></tr> <tr> <td></td> <td></td> <th>AD15</th> <th>39</th> <td></td> <td></td> <th>A15</th> <td>16 </td></tr> <tr> <td></td> <td></td> <th>AD16/S3</th> <th>38</th> <td></td> <td></td> <th>A16</th> <td>15 </td></tr> <tr> <td></td> <td></td> <th>A17/S4</th> <th>37</th> <td></td> <td></td> <th>A17</th> <td>14 </td></tr> <tr> <td></td> <td></td> <th>A18/S5</th> <th>36</th> <td></td> <td></td> <th>A18</th> <td>13 </td></tr> <tr> <td></td> <td></td> <th>A19/S6</th> <th>35</th> <td></td> <td></td> <th>A19</th> <td>12 </td></tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <th>A20</th> <td>11 </td></tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <th>A21</th> <td>10 </td></tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <th>A22</th> <td>8 </td></tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <th>A23</th> <td>7 </td></tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <td>BHE</td> <td>1</td> </tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <td>D0</td> <td>36</td> </tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <td>D1</td> <td>38</td> </tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <td>D2</td> <td>40</td> </tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <td>D3</td> <td>42</td> </tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <td>D4</td> <td>44</td> </tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <td>D5</td> <td>46</td> </tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <td>D6</td> <td>48</td> </tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <td>D7</td> <td>50</td> </tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <td>D8</td> <td>37</td> </tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <td>D9</td> <td>39</td> </tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <td>D10</td> <td>41</td> </tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <td>D11</td> <td>43</td> </tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <td>D12</td> <td>45</td> </tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <td>D13</td> <td>47</td> </tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <td>D14</td> <td>49</td> </tr> <tr> <td></td> <td></td> <td colspan="2"></td> <td></td> <td></td> <td>D15</td> <td>51</td> </tr> </thead></table>	33	MN	AD0	16	63	READY	A0	34	22	READY	AD1	15	31	CLK	A1	33	19	CLK	AD2	14	29	RESET	A2	32	21	RESET	AD3	13	5	S0	A3	28	18	INTR	AD4	12	4	S1	A4	27			AD5	11			A5	26			AD6	10			A6	25			AD7	9			A7	24			AD8	8			A8	23			AD9	7			A9	22			AD10	6			A10	21			AD11	5			A11	20			AD12	4			A12	19			AD13	3			A13	18			AD14	2			A14	17			AD15	39			A15	16			AD16/S3	38			A16	15			A17/S4	37			A17	14			A18/S5	36			A18	13			A19/S6	35			A19	12							A20	11							A21	10							A22	8							A23	7							BHE	1							D0	36							D1	38							D2	40							D3	42							D4	44							D5	46							D6	48							D7	50							D8	37							D9	39							D10	41							D11	43							D12	45							D13	47							D14	49							D15	51
33	MN	AD0	16	63	READY	A0	34																																																																																																																																																																																																																																																																																																																																	
22	READY	AD1	15	31	CLK	A1	33																																																																																																																																																																																																																																																																																																																																	
19	CLK	AD2	14	29	RESET	A2	32																																																																																																																																																																																																																																																																																																																																	
21	RESET	AD3	13	5	S0	A3	28																																																																																																																																																																																																																																																																																																																																	
18	INTR	AD4	12	4	S1	A4	27																																																																																																																																																																																																																																																																																																																																	
		AD5	11			A5	26																																																																																																																																																																																																																																																																																																																																	
		AD6	10			A6	25																																																																																																																																																																																																																																																																																																																																	
		AD7	9			A7	24																																																																																																																																																																																																																																																																																																																																	
		AD8	8			A8	23																																																																																																																																																																																																																																																																																																																																	
		AD9	7			A9	22																																																																																																																																																																																																																																																																																																																																	
		AD10	6			A10	21																																																																																																																																																																																																																																																																																																																																	
		AD11	5			A11	20																																																																																																																																																																																																																																																																																																																																	
		AD12	4			A12	19																																																																																																																																																																																																																																																																																																																																	
		AD13	3			A13	18																																																																																																																																																																																																																																																																																																																																	
		AD14	2			A14	17																																																																																																																																																																																																																																																																																																																																	
		AD15	39			A15	16																																																																																																																																																																																																																																																																																																																																	
		AD16/S3	38			A16	15																																																																																																																																																																																																																																																																																																																																	
		A17/S4	37			A17	14																																																																																																																																																																																																																																																																																																																																	
		A18/S5	36			A18	13																																																																																																																																																																																																																																																																																																																																	
		A19/S6	35			A19	12																																																																																																																																																																																																																																																																																																																																	
						A20	11																																																																																																																																																																																																																																																																																																																																	
						A21	10																																																																																																																																																																																																																																																																																																																																	
						A22	8																																																																																																																																																																																																																																																																																																																																	
						A23	7																																																																																																																																																																																																																																																																																																																																	
						BHE	1																																																																																																																																																																																																																																																																																																																																	
						D0	36																																																																																																																																																																																																																																																																																																																																	
						D1	38																																																																																																																																																																																																																																																																																																																																	
						D2	40																																																																																																																																																																																																																																																																																																																																	
						D3	42																																																																																																																																																																																																																																																																																																																																	
						D4	44																																																																																																																																																																																																																																																																																																																																	
						D5	46																																																																																																																																																																																																																																																																																																																																	
						D6	48																																																																																																																																																																																																																																																																																																																																	
						D7	50																																																																																																																																																																																																																																																																																																																																	
						D8	37																																																																																																																																																																																																																																																																																																																																	
						D9	39																																																																																																																																																																																																																																																																																																																																	
						D10	41																																																																																																																																																																																																																																																																																																																																	
						D11	43																																																																																																																																																																																																																																																																																																																																	
						D12	45																																																																																																																																																																																																																																																																																																																																	
						D13	47																																																																																																																																																																																																																																																																																																																																	
						D14	49																																																																																																																																																																																																																																																																																																																																	
						D15	51																																																																																																																																																																																																																																																																																																																																	

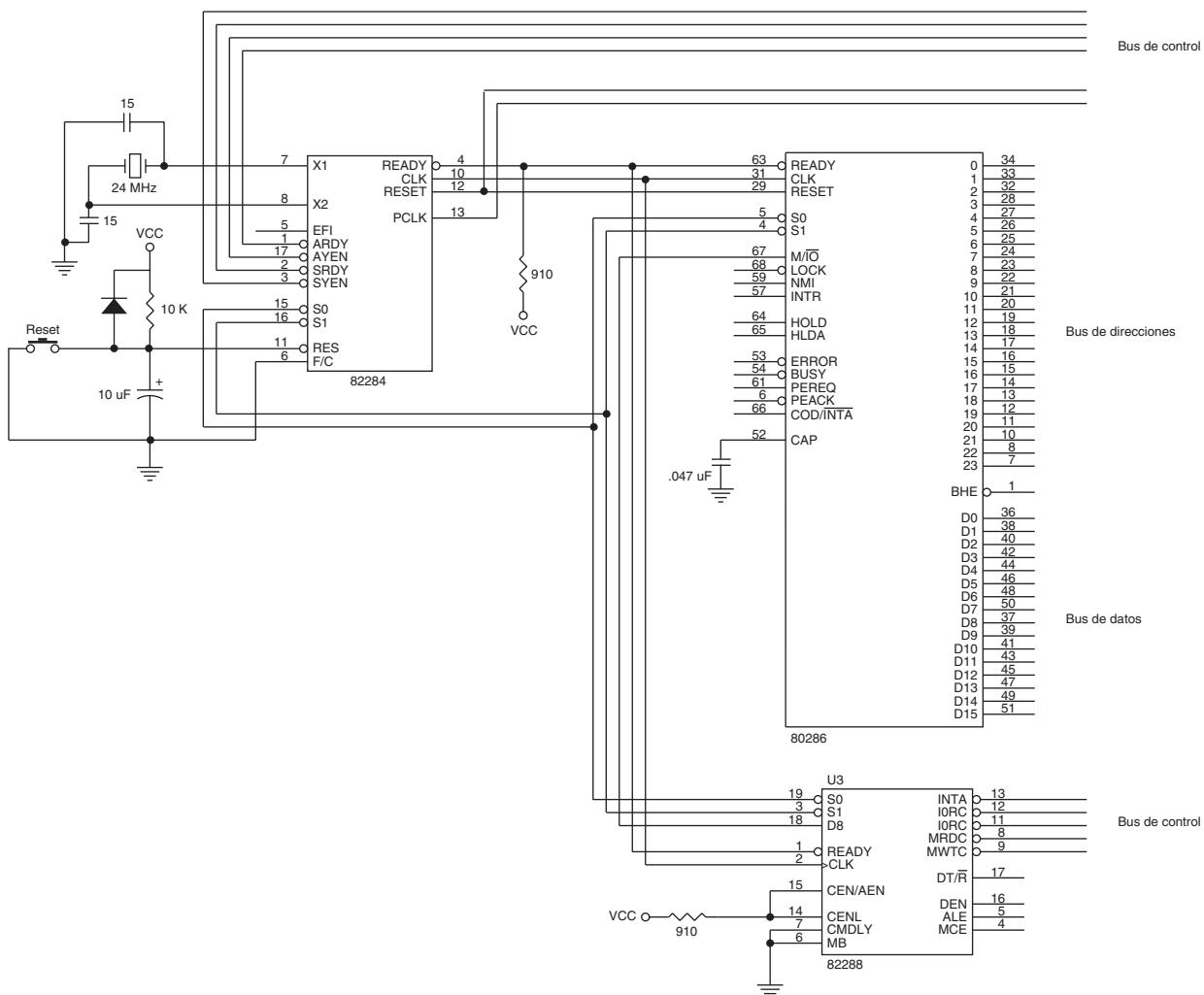


TABLA 16-9 Instrucciones adicionales del 80286.

<i>Instrucción</i>	<i>Propósito</i>
CLTS	Borra el bit de conmutación por tarea.
LDGT	Carga el registro de la tabla de descriptores globales.
SGDT	Almacena el registro de la tabla de descriptores globales.
LIDT	Carga el registro de la tabla de descriptores de interrupción.
SIDT	Almacena el registro de la tabla de descriptores de interrupción.
LLDT	Carga el registro de la tabla de descriptores locales.
SLDT	Almacena el registro de la tabla de descriptores locales.
LMSW	Carga el registro de estado del equipo.
SMSW	Almacena el registro de estado del equipo.
LAR	Carga los derechos de acceso.
LSL	Carga el límite de segmento.
SAR	Almacena los derechos de acceso.
ARPL	Ajusta el nivel de privilegio solicitado.
VERR	Verifica un acceso de lectura.
VERW	Verifica un acceso de escritura.

A continuación se muestran las descripciones de las instrucciones que no se explican en la sección de administración de memoria. Las instrucciones aquí descritas son especiales y se utilizan sólo para las condiciones indicadas.

CLTS	La instrucción borra bandera de conmutación por tarea (CLTS) borra el bit de bandera TS (tarea comutada) para ponerlo en 0 lógico. Si el bit de bandera TS es un 1 lógico y la tarea utiliza el coprocesador numérico 80287, se produce una interrupción (tipo de vector 7). Lo anterior permite emular la función del coprocesador con software. La instrucción CLTS se utiliza en un sistema y se considera una instrucción privilegiada, debido a que puede ejecutarse sólo en el modo protegido, al nivel de privilegio 0. No hay una instrucción para activar la bandera TS; esto se logra mediante la escritura de un 1 lógico en la posición de bit 3 (TS) de la palabra de estado del equipo (WS), mediante el uso de la instrucción LMSW.
LAR	La instrucción carga derechos de acceso (LAR) lee el descriptor de segmento y coloca una copia del byte de derechos de acceso en un registro de 16 bits. La instrucción LAR AX,BX es un ejemplo; dicha instrucción carga AX con el byte de derechos de acceso del descriptor seleccionado por el valor de selector que se encuentra en BX. Esta instrucción se utiliza para obtener los derechos de acceso, de manera que puedan verificarse antes de que un programa utilice el segmento de memoria definido por el descriptor.
LSL	La instrucción carga límite de segmento (LSL) carga un registro especificado por el usuario con el límite de segmento. Por ejemplo, la instrucción LSL AX,BX carga AX con el límite del segmento definido por el descriptor seleccionado por el selector en BX. Esta instrucción se utiliza para evaluar el límite de un segmento.
ARPL	La instrucción ajusta nivel de privilegio solicitado (ARPL) se utiliza para evaluar un selector, de manera que no se viole el nivel de privilegio del selector solicitado. La instrucción ARPL AX,CX es un ejemplo: AX contiene el nivel de privilegio solicitado y CX el valor del selector que va a utilizarse para acceder a un descriptor. Si el nivel de privilegio solicitado es de una menor prioridad que el descriptor que se está evaluando, se activa la bandera de cero. Esto podría requerir que un programa ajuste el nivel de privilegio solicitado o indique una violación de privilegios.
VERR	La instrucción verifica acceso de lectura (VERR) comprueba que sea posible leerse un segmento. Recuerde que en el capítulo 1 vimos que un segmento de código puede estar protegido contra lectura. Si el segmento de código se lee, se activa el bit de la

bandera de cero. La instrucción VERR AX evalúa el descriptor seleccionado por el registro AX.

VERW	La instrucción verifica acceso de lectura (VERW) se utiliza para comprobar que se pueda escribir en un segmento. Recuerde que en el capítulo 1 vimos que un segmento de datos llega a estar protegido contra escritura. Si puedes escribir en el segmento de datos, se activa el bit de la bandera de cero.
-------------	--

La máquina de memoria virtual

Una **máquina de memoria virtual** es una máquina que asigna un espacio de memoria más grande (1 Gbyte para el 80286) en un espacio de memoria física mucho más pequeño (16 Mbytes para el 80286), lo cual permite que un sistema muy grande se ejecute en sistemas con memoria física más pequeña. Esto se logra mediante el uso de colas para los datos entre el sistema de memoria en disco fijo y la memoria física. En el microprocesador 80286, el direccionamiento de un sistema de memoria de 1 Gbyte se logra mediante los descriptores. Cada descriptor del 80286 describe un segmento de memoria de 64 Kbytes y el 80286 permite 16 K descriptores. Esto ($64\text{ K} \times 16\text{ K}$) permite describir un máximo de 1 Gbyte de memoria para el sistema.

Como se mencionó en el capítulo 1, los descriptores describen el segmento de memoria en el modo protegido. El 80286 tiene descriptores que definen códigos, datos, segmentos de pila, interrupciones, procedimientos y tareas. Para acceder a los descriptores, se carga un registro de segmento con un selector en el modo protegido. El selector accede a un descriptor que describe un área de la memoria. En los capítulos 1, 17, 18 y 19 se definen detalles adicionales sobre los descriptores y sus aplicaciones. En estos capítulos podrá consultar una vista detallada sobre el sistema de administración de memoria en modo protegido.

16-6

RESUMEN

1. Los microprocesadores 80186/80188 tienen el mismo conjunto básico de instrucciones que el de los microprocesadores 8086/8088, sólo que se agregan unas cuantas instrucciones más. Por lo tanto, los microprocesadores 80186/80188 son versiones mejoradas de los microprocesadores 8086/8088. Las nuevas instrucciones son: PUSHA, POPA, INS, OUTS, BOUND, ENTER, LEAVE, junto con los conteos de multiplicación inmediata y desplazamiento/desplazamiento cíclico.
2. Las mejoras de hardware para el 80186/80188 incluyen un generador de reloj, un controlador de interrupciones programable, tres temporizadores programables, un controlador de DMA programable, una unidad lógica de selección de chip programable, un temporizador watchdog, un circuito lógico de refresco de RAM y características adicionales en diversas versiones.
3. El generador de reloj permite que el 80186/80188 opere desde una fuente de reloj de nivel TTL externa, o desde un cristal conectado a las terminales X_1 (CLK_{IN}) y X_2 (OSCOUY). La frecuencia del cristal es el doble de la frecuencia de operación del microprocesador. Los microprocesadores 80186/80188 están disponibles en velocidades de 6-20 MHz.
4. El controlador de interrupciones programable controla todas las peticiones de interrupción internas y externas. También es capaz de operar con dos controladores de interrupción 8259A internos.
5. Dentro del 80186/80188 se encuentran tres temporizadores programables. Cada temporizador es un contador de 16 bits 100% programable, el cual se utiliza para generar formas de onda o eventos de conteo. Dos de los temporizadores (0 y 1) tienen entradas y salidas externas. El tercer temporizador (2) se controla desde el reloj del sistema y se utiliza para proporcionar una señal de reloj para otro temporizador, o para solicitar una acción de DMA.
6. El controlador de DMA programable es un controlador de dos canales 100% programable. Las transferencias por DMA se realizan entre la memoria y E/S, entre E/S y E/S o entre posiciones de memoria. Las peticiones de DMA se producen desde el software, el hardware o la salida del temporizador 2.

7. La unidad de selección de chip programable es un decodificador interno que proporciona hasta 13 terminales de salida para seleccionar memoria (6 terminales) y E/S (7 terminales). También inserta de 0 a 3 estados de espera, con o sin sincronización READY externa. En las versiones EB y EC, el número de estados de espera puede programarse de 0 a 15 con 10 terminales de selección de chip.
8. La única diferencia entre la sincronización del 80186/80188 y el 8086/8088 es que ALE aparece medio pulso de reloj antes. Aparte de eso, la sincronización es idéntica.
9. La versión de 6 MHz del 80186/80188 permite 417 ns de tiempo de acceso para la memoria; la versión de 8 MHz permite 309 ns de tiempo de acceso.
10. Los periféricos internos del 80186/80188 se programan a través de un bloque de control de periféricos (PCB), el cual se inicializa en los puertos de E/S FF00H-FFFFH. El PCB puede moverse hacia cualquier área de memoria o de E/S mediante la modificación del contenido del registro de reubicación de PCB que está en las posiciones de E/S iniciales FFFEY y FFFFH.
11. El 80286 es un 8086 mejorado para incluir una unidad de administración de memoria (MMU). El 80286 es capaz de direccionar un espacio de memoria física de 16 Mbytes gracias a la unidad de administración.
12. El 80286 contiene las mismas instrucciones que el 80186/80188, excepto por varias instrucciones adicionales que controlan la unidad de administración de memoria.
13. Mediante la unidad de administración de memoria, el microprocesador 80286 dirige un espacio de memoria virtual de 1 Gbyte, según lo especificado por los descriptores de 16 K almacenados en dos tablas de descriptores.

16-7

PREGUNTAS Y PROBLEMAS

1. Liste las diferencias entre los microprocesadores 8086/8088 y 80186/80188.
2. ¿Qué mejoras se agregaron al 80186/80188 que no están presentes en el 8086/8088?
3. ¿En qué tipos de circuitos integrados viene empaquetado el 80186/80188?
4. Si el cristal de 20 MHz se conecta a X_1 y X_2 , ¿qué señal de frecuencia se encuentra en CLKOUT?
5. Describa las diferencias entre las versiones 80C188XL y 80C188EB del controlador integrado 80188.
6. El factor de salida de cualquier terminal del 80186/80188 es _____ para un 0 lógico.
7. ¿Cuántos periodos de reloj se encuentran en un ciclo de bus del 80186/80188?
8. ¿Cuál es la principal diferencia entre la sincronización del 8086/8088 y del 80186/80188?
9. ¿Cuál es la importancia del tiempo de acceso a memoria?
10. ¿Cuánto tiempo de acceso a memoria permite el 80186/80188 si se opera con un reloj de 10 MHz?
11. ¿En dónde se encuentra el bloque de control de periféricos después de que se reinicia el 80186/80188?
12. Escriba el software requerido para mover el bloque de control de periféricos hacia las posiciones de memoria 10000H-100FFH.
13. ¿Qué vector de interrupción utiliza la terminal INT₀ en los microprocesadores 80186/80188?
14. ¿Cuántos vectores de interrupción están disponibles para el controlador de interrupciones que se encuentra dentro de los microprocesadores 80186/80188?
15. ¿Cuáles son los dos modos de operación disponibles para el controlador de interrupciones?
16. ¿Cuál es el propósito del registro de control de interrupciones?
17. Cada vez que se enmascara una fuente de interrupción, el bit de máscara en el registro de máscara de interrupción es un _____ lógico.
18. ¿Cuál es la diferencia entre los registros de sondeo de interrupciones y de estado de sondeo de interrupciones?
19. ¿Cuál es el propósito del registro de fin de interrupción (EOI)?
20. ¿Cuántos temporizadores de 16 bits se encuentran dentro del 80186/80188?

21. ¿Cuáles temporizadores tienen conexiones de terminales de entrada y de salida?
22. ¿Cuál temporizador se conecta al reloj del sistema?
23. Si se utilizan dos registros de comparación de conteo máximo con un temporizador, explique la operación del temporizador.
24. ¿Cuál es el propósito del bit INH del registro de control de temporizador?
25. ¿Cuál es el propósito del bit P de registro de control de temporizador?
26. ¿Qué tipo de operación selecciona el bit ALT del registro de control de temporizador para los temporizadores 0 y 1?
27. Explique cómo se utilizan las terminales de salida del temporizador.
28. Desarrolle un programa que haga que el temporizador 1 genere una señal continua que sea un 1 lógico para 123 conteos y un 0 lógico para 23 conteos.
29. Desarrolle un programa que haga que el temporizador 0 genere un pulso individual después de 105 pulsos de reloj en su terminal de entrada.
30. ¿Cuántos canales de DMA se controlan mediante el controlador de DMA en el 80C186XL?
31. Los registros de origen y de destino del controlador de DMA son de _____ bits cada uno.
32. ¿Cómo se inicia el canal de DMA mediante software?
33. La unidad de selección de chip (XL y EA) tiene _____ terminales para seleccionar dispositivos de memoria.
34. La unidad de selección de chip (XL y EA) tiene _____ terminales para seleccionar dispositivos periféricos.
35. La última posición del bloque de memoria superior que selecciona la terminal UCS es la posición _____.
36. Las terminales de selección de chip de memoria media (XL y EA) se programan para un tamaño de _____ y de bloque.
37. El área de memoria inferior seleccionada por LCS empieza en la dirección _____.
38. El generador de estados de espera interno (versiones EB y EC) es capaz de insertar entre cero y _____ estados de espera.
39. Programe el registro A₈H (XL y EA) de manera que el tamaño del bloque de memoria de rango medio sea de 128 Kbytes y que el tamaño del chip sea de 32 K.
40. ¿Cuál es el propósito del bit EX en el registro A₈H?
41. Desarrolle el software requerido para programar la terminal GCS3, de manera que seleccione la memoria de las posiciones 20000H-2FFFFH y que inserte dos estados de espera.
42. Desarrolle el software requerido para programar la terminal GCS4, de manera que seleccione un dispositivo de E/S para los puertos 1000H-103FH y que inserte un estado de espera.
43. El microprocesador 80286 direcciona _____ bytes de memoria física.
44. Cuando se está utilizando el administrador de memoria, el 80286 direcciona _____ bytes de memoria virtual.
45. El conjunto de instrucciones del 80286 es idéntico al del _____, exceptuando por las instrucciones de administración de memoria.
46. ¿Cuál es el propósito de la instrucción VERR?
47. ¿Cuál es el propósito de la instrucción LSL?
48. ¿Qué es un RTOS?
49. ¿Cómo se manejan los subprocesos múltiples con el RTOS?
50. Busque en Internet por lo menos dos RTOS distintos y escriba un informe breve en el que los compare.

CAPÍTULO 17

Los microprocesadores 80386 y 80486

INTRODUCCIÓN

El microprocesador 80386 es una versión completa de 32 bits de los microprocesadores anteriores 8086/80286 de 16 bits y representa un importante avance en la arquitectura: un cambio de la arquitectura de 16 bits a la de 32 bits. Aparte del aumento en el tamaño de las palabras, hay muchas mejoras y características adicionales. El microprocesador 80386 cuenta con multitarea, administración de memoria, memoria virtual (con o sin paginación), protección de software y un sistema de memoria extenso. Todo el software escrito para los microprocesadores 8086/8088 y 80286 es compatible en forma ascendente con el microprocesador 80386. La cantidad de memoria que este microprocesador puede direccionar es de 4 Gbytes, un aumento considerable en comparación con la cantidad de 1 Mbyte que dirige el 8086/8088 y los 16 Mbytes en el 80286. El 80386 llega a cambiar de modo protegido a modo real (o viceversa) sin necesidad de reiniciarse. En el microprocesador 80286 el cambio de modo protegido a modo real era un problema, pues requería un reinicio de hardware.

El microprocesador 80486 es una versión mejorada del 80386 que ejecuta muchas de sus instrucciones en un periodo de reloj. El microprocesador 80486 también contiene una memoria caché de 8 Kbytes y un coprocesador numérico 80387 mejorado (el 80486DX4 contiene una caché de 16 Kbytes). Cuando el 80486 se opera a la misma frecuencia de reloj que un 80386, presenta un incremento aproximado del 50% en su velocidad de ejecución. En el capítulo 18 hablaremos con detalle sobre el Pentium y el Pentium Pro. Estos microprocesadores contienen una memoria caché de 16 K y operan a más del doble de la velocidad del microprocesador 80486. El Pentium y el Pentium Pro también contienen coprocesadores numéricos mejorados que operan cinco veces más rápido que el coprocesador numérico 80486. En el capítulo 19 hablaremos sobre las mejoras adicionales en los microprocesadores del Pentium II al Pentium 4.

OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

1. Comparar los microprocesadores 80386 y 80486 con los primeros microprocesadores Intel.
2. Describir la operación de las unidades de administración de memoria y de paginación del 80386 y del 80486.
3. Alternar entre el modo protegido y el modo real.
4. Definir la operación de las instrucciones adicionales del 80386/80486 y sus modos de direccionamiento.
5. Explicar la operación de un sistema de memoria caché.
6. Detallar la estructura de interrupciones y la estructura del acceso directo a memoria del 80386/80486.
7. Comparar el microprocesador 80486 con el 80386.
8. Explicar la operación de la memoria caché del 80486.

17-1

INTRODUCCIÓN AL MICROPROCESADOR 80386

Para que el 80386 o cualquier otro microprocesador pueda utilizarse en un sistema, hay que comprender la función de cada una de sus terminales. En esta sección explicaremos con detalle la operación de cada terminal, junto con las estructuras del sistema de memoria externa y de E/S del 80386.

La figura 17-1 muestra el diagrama de terminales del microprocesador 80386DX. Este microprocesador viene empaquetado en una PGA (matriz de rejillas de terminales) de 132 terminales. Por lo general, hay dos versiones disponibles del 80386: el 80386DX, que veremos en este capítulo; y el 80386SX, que es una versión del 80386 con un bus reducido. Una nueva versión del 80386 (la 80386EX) incorpora el sistema de bus AT, un controlador de RAM dinámico, lógica de selección de chips programable, 26 terminales de dirección, 16 terminales de datos y 24 terminales de E/S. La figura 17-2 muestra una ilustración de la PC integrada 80386EX.

El 80386DX direcciona 4 Gbytes de memoria a través de su bus de datos de 32 bits y su bus de direcciones de 32 bits. El 80386SX, que es más parecido al 80286, direcciona 16 Mbytes de memoria con su bus de direcciones de 24 bits, a través de su bus de datos de 16 bits. El 80386SX se desarrolló después del 80386DX para aplicaciones que no requerían la versión completa de 32 bits. El 80386SX se utilizó en muchas de las primeras computadoras personales con el mismo diseño básico en su tarjeta principal (o tarjeta madre) que el 80286. Durante el tiempo cuando el 80386SX se hizo popular, muchas aplicaciones (incluyendo Windows 3.11) requerían menos de 16 Mbytes de memoria, por lo que el 80386SX fue una versión popular y menos costosa del microprocesador 80386. Aún y cuando el 80486 se ha convertido en una ruta de actualización menos costosa para los sistemas más recientes, el 80386 es posible utilizarlo todavía para muchas aplicaciones. Por ejemplo, el 80386EX no aparece en sistemas computacionales, pero se está volviendo muy popular en aplicaciones integradas.

Al igual que con las primeras versiones de la familia de microprocesadores Intel, el 80386 requiere una fuente de energía de +5.0 V para operar. La corriente promedio de la fuente de energía es de 550 mA para la versión de 25 MHz del 80386, de 500 mA para la versión de 20 MHz y de 450 mA para la versión de 16 MHz. También hay una versión de 33 MHz disponible, la cual requiere 600 mA de corriente de la fuente de energía. La corriente de la fuente de energía para el 80386EX es de 320 mA cuando se opera a 33 MHz. Hay que tener en cuenta que durante ciertos modos de operación normal,

FIGURA 17-1 Diagramas de terminales de los microprocesadores 80386DX y 80386SX.

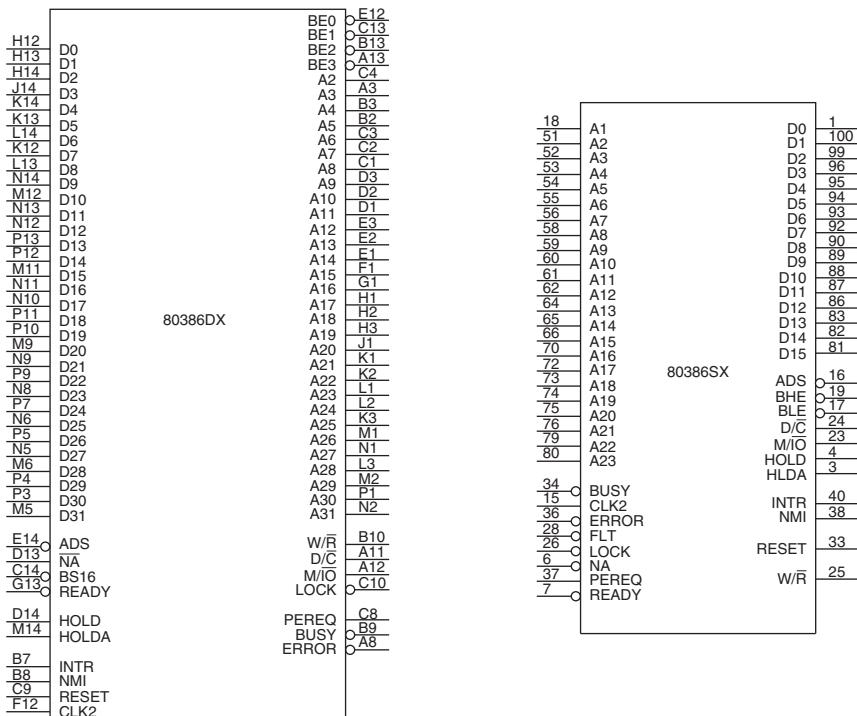
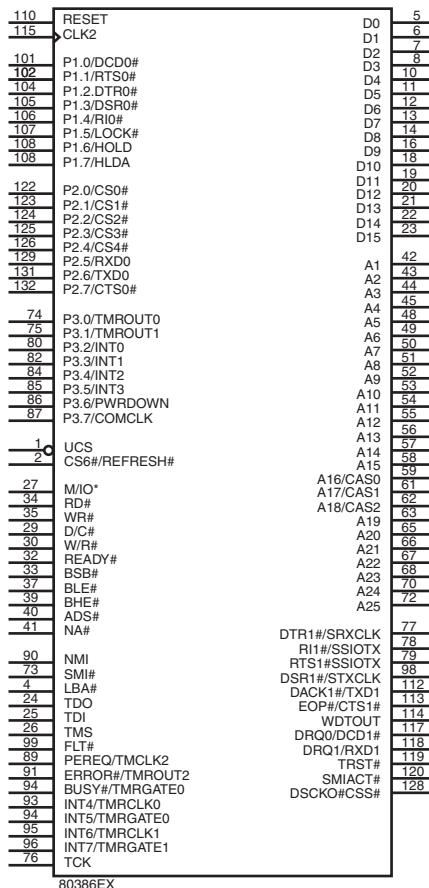


FIGURA 17-2 La PC integrada 80386EX.



la corriente de la fuente de energía llega a oscilar de manera repentina hasta 1.0A. Esto significa que la fuente de energía y la red de distribución de energía deben ser capaces de soportar tales oscilaciones transitorias de corriente. Dicho dispositivo contiene varias conexiones V_{CC} y V_{SS}, que deben conectarse a +5.0 V y aterrizar para una operación adecuada. Algunas de las terminales se identifican como N/C (sin conexión) y no deben conectarse. Hay versiones adicionales del 80386SX y del 80386EX disponibles para usarse con una fuente de energía de +3.3 V. Por lo general, se utilizan en computadoras portátiles y se empaquetan en un dispositivo de montaje superficial.

Cada una de las terminales de salida del 80386 son capaces de suministrar 4.0 mA (conexiones de dirección y de datos) o 5.0 mA (otras conexiones). Esto representa un aumento en la corriente de control, en comparación con los 2.0 mA disponibles en las terminales de salida de los microprocesadores 8086, 8088 y 80286. La corriente de salida disponible en la mayoría de las terminales de salida del 80386EX es de 8.0 mA. Cada terminal de entrada representa una pequeña carga, la cual requiere sólo $\pm 10 \mu A$ de corriente. En algunos sistemas (excepto los más pequeños) estos niveles de corriente requieren búferes de bus.

A continuación se explica la función de cada grupo de terminales del 80386DX:

A₃₁-A₂

Las conexiones del **bus de direcciones** dirige cualquiera de las 1 G × 32 (4 Gbytes) posiciones de memoria del sistema de memoria del 80386. Observe que A₀ y A₁ se codifican en la señal de habilitación de bus ($\overline{BE3}$ - $\overline{BE0}$) para seleccionar cualquiera o todos los cuatro bytes en una posición de memoria de 32 bits. Observe además que como el 80386SX contiene un bus de datos de 16 bits en vez del bus de datos de 32 bits del 80386DX, A₁ está presente en el 80386SX, en tanto que el banco de señales de selección se sustituyen por \overline{BHE} y \overline{BLE} . La señal \overline{BHE} habilita la mitad superior del bus de datos, mientras la señal \overline{BLE} habilita la mitad inferior.

D_{31-D₀}	Las conexiones del bus de datos transfieren datos entre el microprocesador y sus sistemas de memoria y de E/S. El 80386SX contiene las terminales D ₁₅ -D ₀ .
BE3-BE0	Las señales de habilitación de banco seleccionan el acceso de un byte, una palabra o una doble palabra de datos. El microprocesador genera tales señales, de manera interna, a partir de los bits de dirección A ₁ y A ₀ . En el 80386SX, estas terminales se sustituyen por BHE, BLE y A ₁ .
M/IO	La señal de memoria/IO selecciona un dispositivo de memoria cuando es un 1 lógico o un dispositivo de E/S, cuando es un 0 lógico. Durante la operación de E/S, el bus de direcciones contiene una dirección de E/S de 16 bits en las conexiones de direcciones A ₁₅ -A ₂ .
W/R	La señal de escritura/lectura indica que el ciclo de bus actual es una escritura cuando es 1 lógico o una lectura, cuando es un 0 lógico.
ADS	El estrobo de datos de dirección se activa cuando el 80386 envía una dirección válida de memoria o de E/S. Esta señal se combina con la señal W/R para generar las señales separadas de lectura y escritura presentes en los sistemas anteriores basados en los microprocesadores 8086-80286.
RESET	La señal reinicio inicializa el 80386, con lo cual empieza a ejecutar software en la posición de memoria FFFFFFFF0H. El 80386 se reinicia en el modo real, en tanto que las 12 conexiones de dirección de más a la izquierda se mantienen en el nivel de 1 lógico (FFFH) hasta que se ejecutan un salto lejano o una llamada lejana. Esto facilita la compatibilidad con los microprocesadores anteriores.
CLK₂	La señal reloj por 2 se controla mediante una señal de reloj que equivale al doble de la frecuencia de operación del 80386. Por ejemplo, para operar el 80386 a 16 MHz, se aplica una señal de reloj de 32 MHz a esta terminal.
READY	La señal listo controla el número de estados de espera que se insertan en la sincronización para extender los accesos a memoria.
LOCK	La señal bloqueo se vuelve un 0 lógico cada vez que se coloca el prefijo LOCK en una instrucción. Este prefijo se utiliza con frecuencia durante los accesos de DMA.
D/C	La señal datos/control indica que el bus de datos contiene datos para enviar a, o recibir de, la memoria o E/S cuando es un 1 lógico. Si D/C es un 0 lógico, el microprocesador se detiene o ejecuta un reconocimiento de interrupción.
BS16	La señal tamaño de bus 16 selecciona un bus de datos de 32 bits (BS16 = 1) o un bus de datos de 16 bits (BS16 = 0). En la mayoría de los casos, si se va a operar un 80386DX en un bus de datos de 16 bits, utilizamos el 80386SX, que tiene un bus de datos de 16 bits. En el 80386EX, la terminal BS8 selecciona un bus de datos de 8 bits.
NA	La señal siguiente dirección hace que el 80386 envíe la dirección de la siguiente instrucción o de los datos en el ciclo de bus actual. Esta terminal se utiliza por lo general para canalizar la dirección.
HOLD	La señal retención solicita una acción de DMA.
HLDA	La señal reconocimiento de retención indica que el 80386 se encuentra en una condición de retención.
PEREQ	La señal peticIÓN de coprocesador pide al 80386 que renuncie al control y es una conexión directa al coprocesador aritmético 80387.
BUSY	La señal ocupado es una entrada que se utiliza con las instrucciones WAIT o FWAIT para esperar a que el coprocesador esté disponible. También es una conexión directa al 80387 desde el 80386.
ERROR	La señal error indica al microprocesador que el coprocesador detectó un error.
INTR	Los circuitos externos utilizan una peticIÓN de interrupción para solicitar una interrupción.
NMI	Una interrupción no enmascarable solicita una interrupción no enmascarable, de igual forma que en las versiones anteriores del microprocesador.

El sistema de memoria

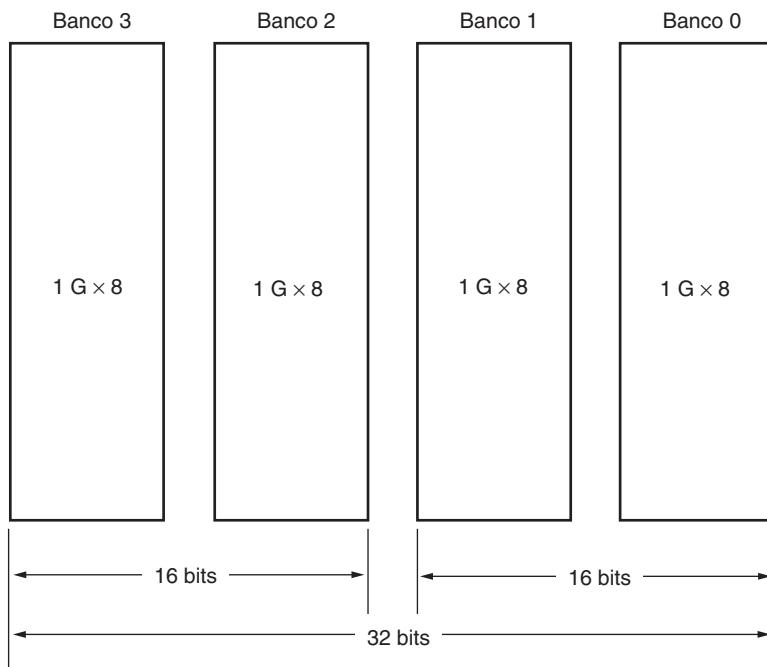
El sistema de memoria física del 80386DX tiene 4 Gbytes de tamaño y se direcciona como tal. Si se utiliza el direccionamiento virtual, se asignan 64 Tbytes en los 4 Gbytes de espacio físico mediante la unidad de administración de memoria y los descriptores. (El direccionamiento virtual permite que un programa sea mayor de 4 Gbytes si hay un método para intercambiar con un disco duro de gran capacidad.) La figura 17-3 muestra la organización del sistema de memoria física del 80386DX.

La memoria se divide en cuatro bancos de memoria de 8 bits, cada uno de los cuales contiene hasta 1 Gbyte de memoria. Esta organización de la memoria en 32 bits permite acceder directamente a los datos de la memoria en forma de bytes, palabras o dobles palabras. El 80386DX transfiere un número de hasta 32 bits en un solo ciclo de memoria, mientras que el 8088 requiere cuatro ciclos para realizar la misma transferencia; los microprocesadores 80286 y 80386SX requieren dos ciclos. Hoy es importante tener en cuenta la anchura de los datos, en especial con los números de punto flotante con precisión simple de 32 bits. Por lo general, el software de alto nivel utiliza números de punto flotante para el almacenamiento de datos, así que las posiciones de memoria de 32 bits agilizan la ejecución del software de alto nivel cuando éste se escribe de una forma en la que logre aprovechar esta memoria más amplia.

Cada byte de memoria está numerado en hexadecimal, como en las versiones anteriores de la familia. La diferencia es que el 80386DX utiliza una dirección de memoria de 32 bits, en donde los bytes de memoria se numeran desde la posición 00000000H hasta FFFFFFFFH.

Los dos bancos de memoria en los sistemas 8086, 80286 y 80386SX se utilizan a través de BLE (A_0 en el 8086 y el 80286) y BHE. En el 80386DX, los bancos de memoria se utilizan a través de cuatro señales de habilitación de banco, BE3-BE0. Este arreglo permite acceder a un solo byte cuando el microprocesador activa una señal de habilitación de banco. También permite acceder a una palabra cuando se activan dos señales de habilitación de banco. En la mayoría de los casos, una palabra se direcciona en los bancos 0 y 1, o en los bancos 2 y 3. La posición de memoria 00000000H está en el banco 0, la posición 00000001H está en el banco 1, la posición 00000002H está en el banco 2 y la posición 00000003H en el banco 3. El 80386DX no contiene las conexiones de dirección A_0 y A_1 porque están codificadas como las señales de habilitación de banco. De igual forma, el 80386SX no contiene la terminal de dirección A_0 debido a que está codificada en las señales BLE y BHE. El 80386EX direcciona datos, ya sea en dos bancos para un sistema de memoria de 16 bits si BS8 = 1 o como un sistema de 8 bits si BS8 = 0.

FIGURA 17-3 El sistema de memoria para el microprocesador 80386. Observe que la memoria está organizada en forma de cuatro bancos, donde cada uno de los cuales contiene 1 Gbyte. El acceso a la memoria es en forma de datos de 8, 16 o 32 bits.



Sistema con búferes. La figura 17-4 muestra el 80386DX conectado a búferes que aumentan el factor de salida de sus conexiones de dirección, datos y control. Este microprocesador opera a 25 MHz mediante el uso de una señal de entrada de reloj de 50 MHz, que se genera a consecuencia de un módulo oscilador integrado. Los módulos osciladores se utilizan con frecuencia para suministrar una señal de reloj en el equipo moderno basado en microprocesador. La señal HLDA se utiliza para habilitar todos los búferes en un sistema que utiliza el acceso directo a memoria. Para cualquier otro caso, las terminales de habilitación de búfer se conectan a tierra en un sistema que no utilice DMA.

Canalizaciones y cachés. La memoria caché es un búfer que permite que el 80386 funcione de una manera más eficiente con menores velocidades de DRAM. Una **canalización** es una forma especial de manejar los accesos a memoria, de manera que la memoria tenga tiempo adicional para acceder a los datos. Un 80386 de 16 MHz permite que los dispositivos de memoria con tiempos de acceso de 50 ns o menos operen al máximo de velocidad. Es obvio que hay pocas DRAMs disponibles hoy con estos tiempos de acceso. De hecho, las DRAMs más veloces que se utilizan en la actualidad tienen un tiempo de acceso de 40 ns o mayor. Esto significa que debe encontrarse cierta técnica para conectar tales dispositivos de memoria, que son más lentos de lo que requiere el microprocesador. Hay tres técnicas disponibles: la memoria intercalada, el uso de caché y la canalización.

La canalización es el medio preferido para conectar la memoria, ya que el microprocesador 80386 soporta los accesos canalizados a memoria. La canalización permite a la memoria un periodo de reloj adicional para acceder a los datos. Ese periodo de reloj adicional extiende el tiempo de acceso de 50 ns hasta 81 ns en un 80386 que opera con un reloj de 16 MHz. El microprocesador establece el **canal**, como se le llama comúnmente. Cuando se obtiene una instrucción de memoria, por lo general el microprocesador cuenta con un tiempo extra antes de obtener la siguiente instrucción. Durante este tiempo extra, se envía la dirección de la siguiente instrucción del bus de direcciones antes de tiempo. Este tiempo extra (un periodo de reloj) se utiliza para permitir un tiempo de acceso adicional para los componentes de memoria más lentos.

No todas las referencias a memoria logran sacar ventaja del canal, lo cual significa que algunos ciclos de memoria no se canalizan. Estos ciclos de memoria no canalizados solicitan un estado de espera, si el ciclo de canalización normal no requiere estados de espera. En general, un canal es una característica para ahorrar en costos, la cual reduce el tiempo de acceso requerido por el sistema de memoria en sistemas de baja velocidad.

No todos los sistemas llegan a sacar ventaja del canal. Esos sistemas operan por lo general a 20, 25 o 33 MHz. En los de mayor velocidad, debe utilizarse otra técnica para incrementar la velocidad del sistema de memoria. El sistema de memoria **caché** mejora el rendimiento general de los sistemas de memoria para los datos que se utilizan más de una vez. El 80486 contiene una caché interna, a la cual se le conoce como **caché de nivel 1**, en tanto que el 80386 sólo puede contener una caché externa, a la cual se le conoce como **caché de nivel 2**.

Una **caché** es un sistema de memoria (SRAM) de alta velocidad que se coloca entre el microprocesador y el sistema de memoria DRAM. Por lo general, los dispositivos de memoria caché son componentes de memoria RAM estática con tiempos de acceso menores a 10 ns. En muchos casos, vemos sistemas de memoria caché de nivel 2 con tamaños entre 32 K y 1 Mbyte. El tamaño de la memoria caché se determina más con base en la aplicación que en el microprocesador. Si un programa es pequeño y hace referencia a pocos datos en la memoria, una caché pequeña es beneficiosa. Si un programa es grande y hace referencia a bloques extensos de memoria, se recomienda el mayor tamaño de caché posible. En muchos casos, una caché de 64 Kbytes produce un aumento suficiente en la velocidad, pero el beneficio máximo se obtiene a menudo de una caché de 256 Kbytes. Las investigaciones han demostrado que si se aumenta el tamaño de la caché mucho más allá de 256 K, se produce un beneficio muy pequeño en cuanto a la velocidad de operación del sistema que contiene un microprocesador 80386.

Sistemas de memoria intercalada. Un **sistema de memoria intercalada** es otro método para mejorar la velocidad de un sistema. Su única desventaja es que se requiere una cantidad mucho mayor de memoria debido a su estructura. Los sistemas de memoria intercalada se utilizan en algunos sistemas, por lo que los tiempos de acceso a la memoria pueden extenderse sin la necesidad de estados de espera. En algunos sistemas, una memoria intercalada llega a requerir todavía de estados de espera, pero en una cantidad menor. Un sistema de memoria intercalada necesita dos o más conjuntos completos de buses de direcciones y un controlador que proporcione las direcciones para cada bus. Los sistemas que emplean

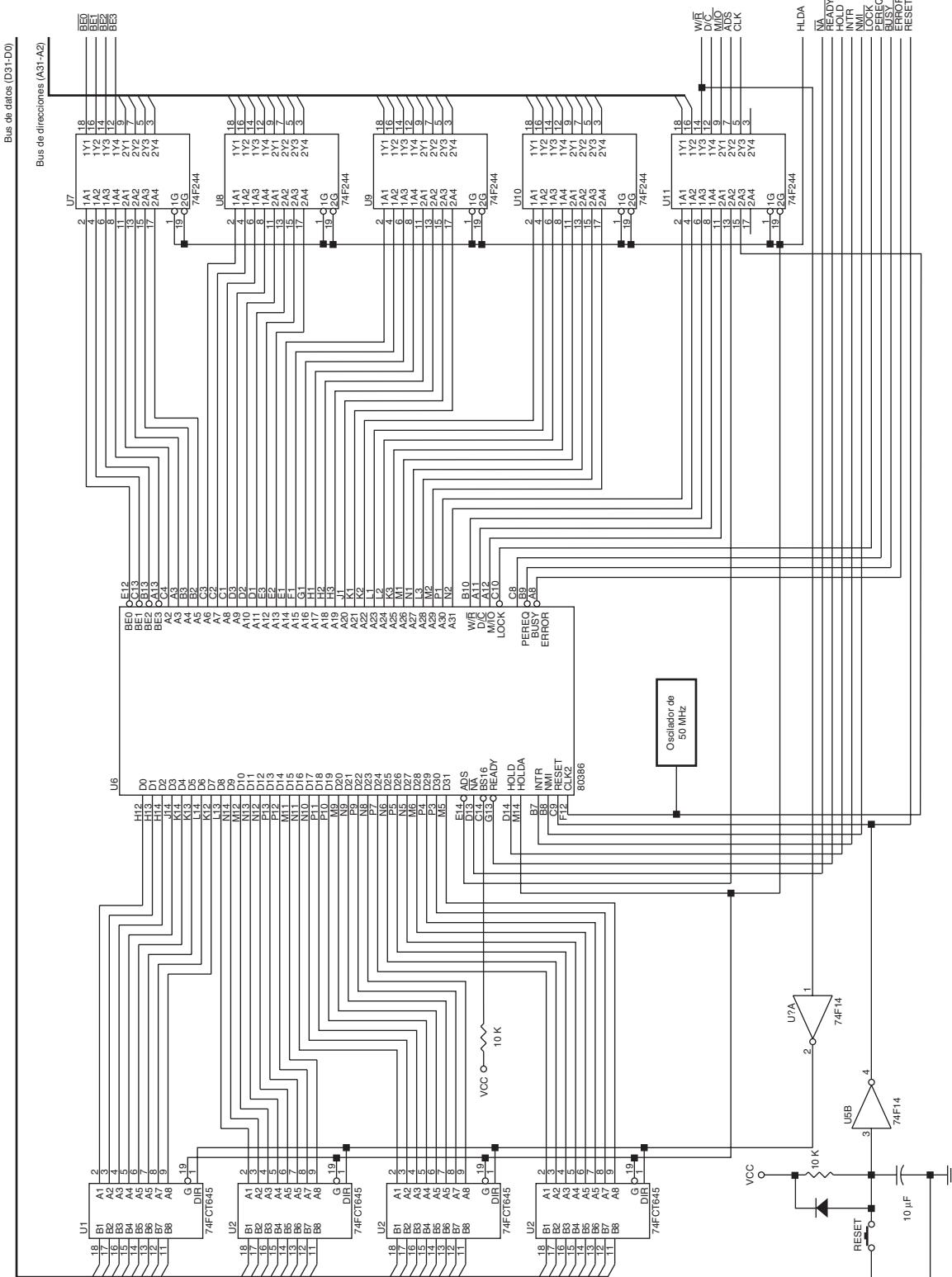


FIGURA 17-4 Un 80386DX de 25 MHz que utiliza búferes en todos sus buses.

dos buses completos se llaman sistemas **intercalados de dos vías**; los sistemas que utilizan cuatro buses completos se llaman sistemas **intercalados de cuatro vías**.

Una memoria intercalada se divide en dos o cuatro partes. Por ejemplo, si se desarrolla un sistema de memoria intercalada para el microprocesador 80386SX, una sección contiene las direcciones de 16 bits 000000H-000001H, 000004H-000005H, y así sucesivamente; la otra sección contiene la direcciones 000002H-000003H, 000006H-000007H, etcétera. Mientras el microprocesador accede a las posiciones 000000H-000001H, la lógica de control de intercalación genera la señal de estrobo de dirección para las posiciones 000002H-000003H. Esta operación selecciona y accede a la palabra que se encuentra en la posición 000002H-000003H, en tanto que el microprocesador procesa la palabra que se encuentra en la posición 000000H-000001H. Tal proceso alterna las secciones de memoria, con lo cual se incrementa el rendimiento del sistema de memoria.

La intercalación incrementa la cantidad de tiempo de acceso que se otorga a la memoria, porque la dirección se genera para seleccionar la memoria antes de que el microprocesador la utilice. Esto se debe a que el microprocesador canaliza las direcciones de memoria y envía la siguiente dirección antes de leer los datos de la dirección anterior.

El problema con la intercalación (aunque no es grave) es que las direcciones de memoria deben utilizarse de manera que se dirijan cada sección en forma alterna. Esto no siempre ocurre a medida que se ejecuta un programa. Durante la ejecución común de un programa, el microprocesador dirige la memoria, de manera alterna, durante un 93% del tiempo. Para el 7% restante, el microprocesador dirige los datos en la misma sección de memoria, lo cual significa que en este 7% de los accesos a memoria el sistema de memoria tiene que esperar a que se transfieran los datos anteriores para obtener su dirección. Esto deja a la memoria con menos tiempo de acceso; por lo tanto, se requiere un estado de espera para los accesos en el mismo banco de memoria.

En la figura 17-5 consultará el diagrama de sincronización de la dirección, según la forma en que aparece en las terminales de dirección del microprocesador. Este diagrama de sincronización muestra cómo se envía la siguiente dirección antes de acceder a los datos actuales. También, cómo se incrementa el tiempo de acceso cuando se utilizan direcciones de memoria intercalada para cada sección de memoria, en comparación con un acceso no intercalado, el cual requiere un estado de espera.

La figura 17-6 contiene el controlador de intercalación. Hay que admitir que es un circuito lógico complejo, que necesita explicación. En primer lugar, si la entrada SEL (que se utiliza para seleccionar esta sección de la memoria) está inactiva (0 lógico) entonces la señal WAIT es un 1 lógico. Además, ALE0 y ALE1 (que se utilizan para enviar mediante estrobo la dirección hacia las secciones de memoria) tienen el valor de 1 lógico, lo cual hace que los enclavamientos conectados se vuelvan transparentes.

Tan pronto como la entrada SEL se convierte en un 1 lógico, este circuito empieza a funcionar. La entrada A1 se utiliza para determinar qué enclavamiento (U2B o U5A) se va a convertir en 0 lógico, para seleccionar una sección de la memoria. Además, la terminal ALE que se vuelve 0 lógico se compara con el estado anterior de las terminales ALE. Si se accede a la misma sección de memoria por segunda ocasión, la señal WAIT se convierte en un 0 lógico para solicitar un estado de espera.

La figura 17-7 muestra un sistema de memoria intercalada que utiliza el circuito de la figura 17-6. Observe cómo se manejan las señales ALE₀ y ALE₁ para capturar la dirección para cualquier sección de memoria. La memoria en cada banco es de 16 bits. Si los accesos a memoria requieren datos de 8 bits, el sistema produce estados de espera en la mayoría de los casos. A medida que se ejecuta un programa, el 80386SX obtiene las instrucciones 16 bits desde posiciones de memoria secuenciales comunes. Para la ejecución de programas se utiliza el intercalado en la mayoría de los casos. Si un sistema va a acceder datos de 8 bits la mayor parte del tiempo, es poco probable que la intercalación de memoria reduzca el número de estados de espera.

El tiempo de acceso permitido por un sistema intercalado, tal como el que se muestra en la figura 17-7, se incrementa de 69 ns a 112 ns mediante el uso de un reloj de sistema de 16 MHz. (Si se inserta un estado de espera, el tiempo de acceso con un reloj de 16 MHz es de 136 ns, lo cual significa que un sistema intercalado funciona casi a la misma velocidad que un sistema con un estado de espera.) Si el reloj se aumenta a 20 MHz la memoria intercalada requerirá 89.6 ns, mientras que las interfaces de memoria no intercalada estándar emplean 48 ns para el acceso a memoria. A esta velocidad de reloj más alta, las DRAMs de 80 ns funcionan a la perfección sin estados de espera cuando las direcciones de memoria son intercaladas. Si ocurre un acceso a la misma sección, se inserta un estado de espera.

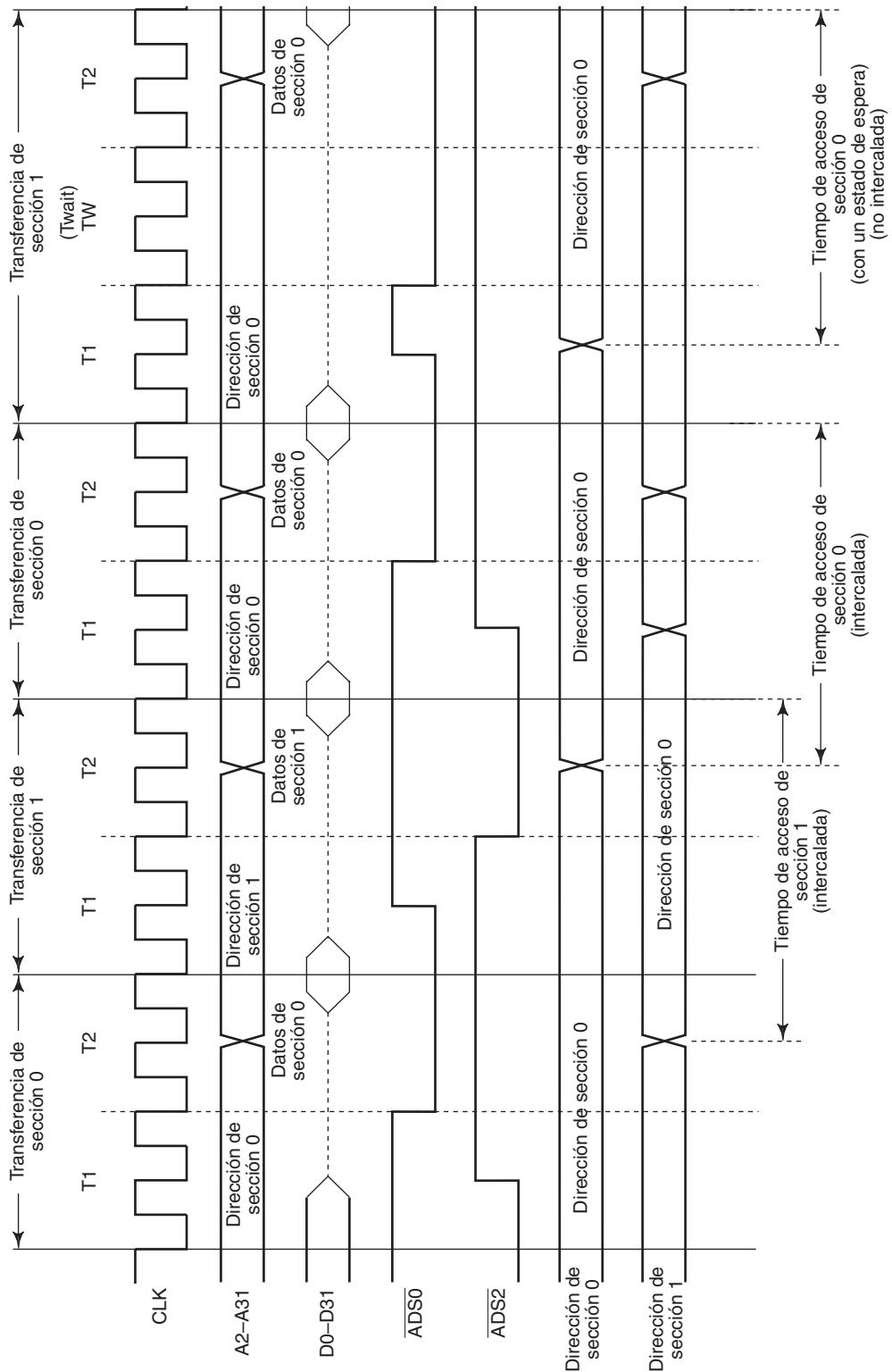


FIGURA 17-5 Diagrama de sincronización de un sistema de memoria intercalada, en el que se muestran los tiempos de acceso y las señales de dirección para ambas secciones de memoria.

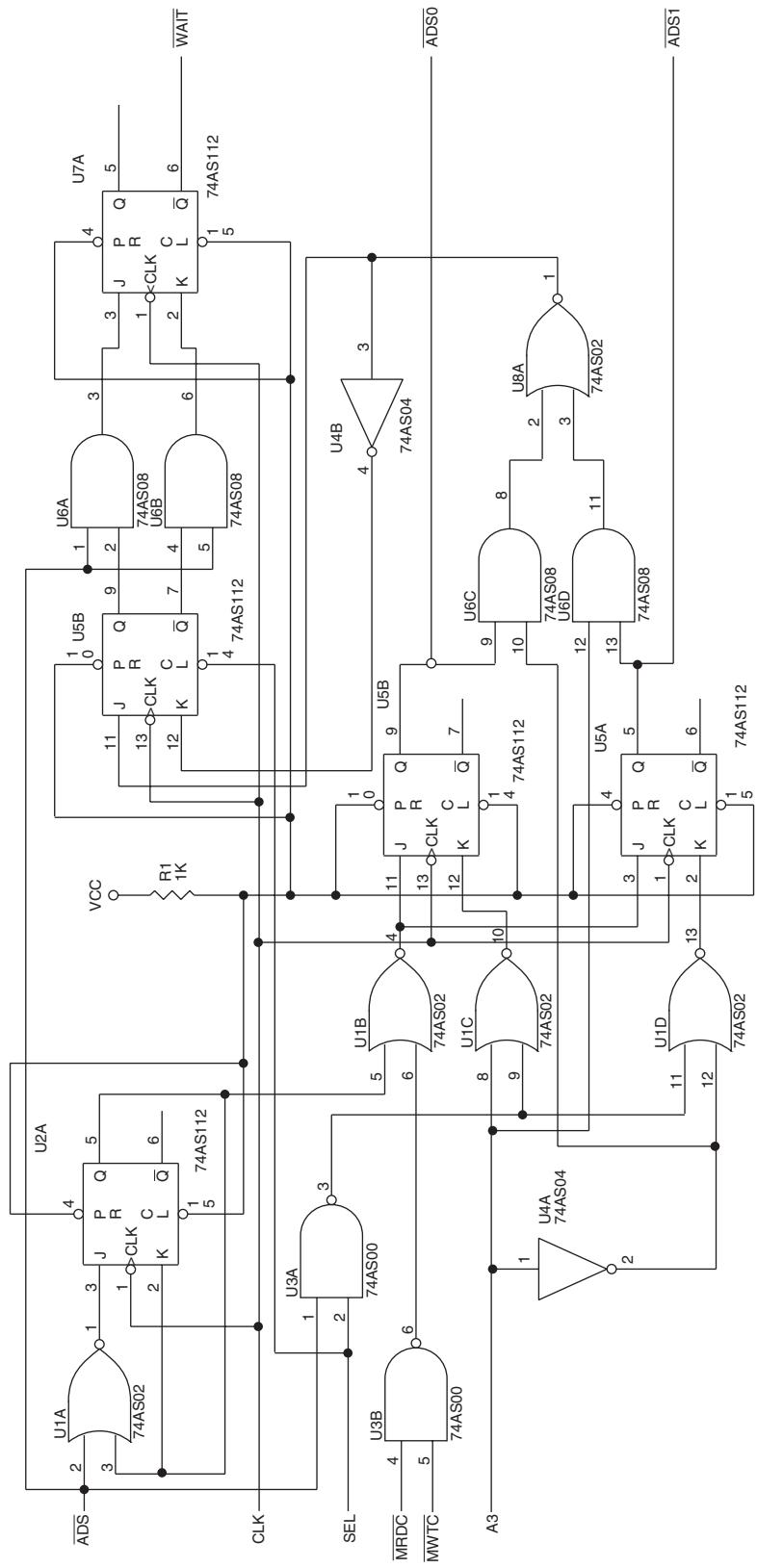


FIGURA 17-6 La lógica de control de intercalación, que genera señales ADS separadas y una señal WAIT que se utiliza para controlar la memoria intercalada.

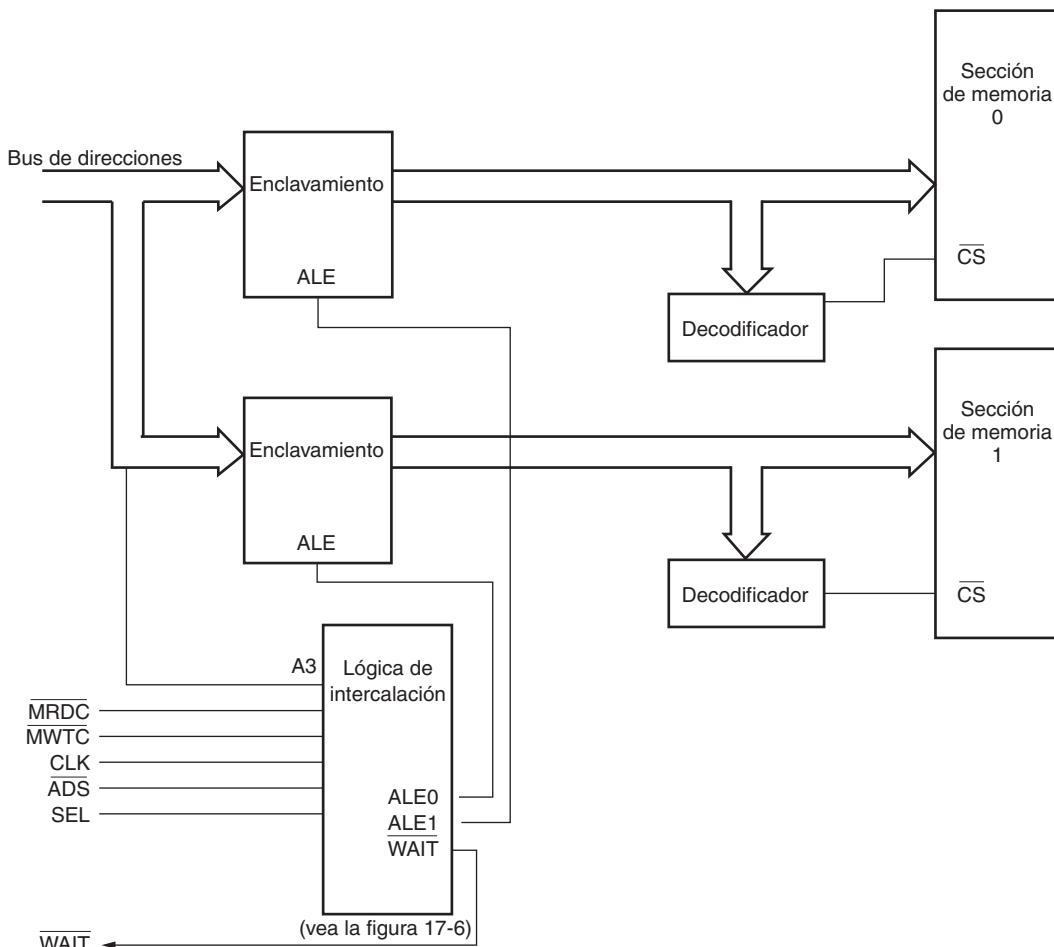


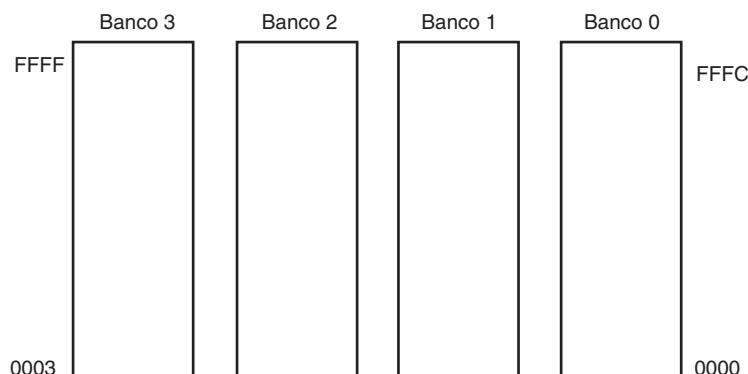
FIGURA 17-7 Un sistema de memoria intercalada que muestra los enclavamientos de dirección y el circuito de lógica de intercalación.

El sistema de entrada/salida

El sistema de entrada/salida del 80386 es el mismo que se utiliza en cualquier sistema basado en microprocesador de la familia Intel 8086. Existen 64 Kbytes distintos de espacio de E/S disponibles, si se pone en marcha la E/S aislada. Con este tipo de E/S, las instrucciones IN y OUT se utilizan para transferir datos de E/S entre el microprocesador y los dispositivos de E/S. La dirección de puerto de E/S aparece en las conexiones del bus de direcciones A₁₅-A₂, en tanto que las terminales BE₃-BE₀ se utilizan para seleccionar un byte, una palabra o una doble palabra de datos de E/S. Si se instrumenta la E/S por asignación de memoria, entonces el número de posiciones de E/S llegará a ser cualquier cantidad hasta 4 Gbytes. Con la E/S por asignación de memoria puede usarse cualquier instrucción que transfiera datos entre el microprocesador y el sistema de memoria para las transferencias de E/S, ya que el dispositivo de E/S se trata como un dispositivo de memoria. Casi todos los sistemas 80386 utilizan E/S aislada debido al esquema de protección de E/S, que proporciona el 80386 en la operación en modo protegido.

La figura 17-8 muestra el mapa de E/S para el microprocesador 80386. A diferencia del mapa de E/S de los primeros microprocesadores Intel, que eran de 16 bits, el 80386 utiliza un sistema de E/S de 32 bits que se divide en cuatro bancos. Esto es idéntico al sistema de memoria, que también se divide en cuatro bancos. La mayoría de las transferencias de E/S son de 8 bits, ya que con frecuencia manejamos el código ASCII (un código de siete bits) para transferir datos alfanuméricos entre el microprocesador y las impresoras y teclados. Esto podría cambiar si el uso de Unicode (un código alfanumérico de 16 bits) se hace común y sustituye al código ASCII. Hace poco aparecieron dispositivos de E/S de 16, e incluso de 32 bits, para sistemas tales como la memoria en disco y las interfaces de pantalla de vídeo.

FIGURA 17-8 El mapa de E/S aislada para el microprocesador 80386. Aquí se utilizan cuatro bancos de 8 bits cada uno para direccionar 64 K posiciones de E/S distintas. La E/S se enumera desde la posición 0000H hasta FFFFH.



Estas rutas de E/S más amplias incrementan la velocidad de transferencia de datos entre el microprocesador y el dispositivo de E/S, en comparación con las transferencias de 8 bits.

Las posiciones de E/S se enumeran desde 0000H hasta FFFFH. Una porción del mapa de E/S fue diseñada para el coprocesador aritmético 80387. Aunque los números de puerto para el coprocesador están muy por encima del mapa normal de E/S, es importante tomarlos en cuenta cuando se decodifica el espacio de E/S (por los traslapamientos). El coprocesador utiliza las posiciones de E/S 800000F8H-800000FFH para las comunicaciones entre el 80387 y el 80386. El coprocesador numérico 80287, diseñado para integrarse al 80286, utiliza las direcciones de E/S 00F8H-00FFH para las comunicaciones. Como por lo general sólo decodificamos las conexiones de dirección A₁₅-A₂ para seleccionar un dispositivo de E/S, hay que estar conscientes de que el coprocesador activará los dispositivos 00F8H-00FFH a menos que también se decodifique la línea de dirección A₃₁. Esto no tendría por qué ser un problema, ya que en realidad no deberíamos de utilizar los puertos de E/S 00F8H-00FFH para ningún propósito.

La única nueva característica que se agregó al 80386 con respecto a la E/S es la información de privilegio de E/S que se agrega al extremo final del TSS cuando el 80386 se opera en modo protegido. Como se describió en la sección sobre administración de memoria, una posición de E/S puede bloquearse o inhibirse en el modo protegido. Si se direcciona la posición de E/S bloqueada se genera una interrupción (tipo 13, fallo general). Este esquema se agregó de manera que prohibiese el acceso a la E/S en un entorno multiusuario. El bloqueo es una extensión de la operación en modo protegido, al igual que los niveles de privilegios.

Señales de control de memoria y de E/S

La memoria y la E/S se controlan mediante señales separadas. La señal M/\overline{IO} indica si la transferencia de datos se va a realizar entre el microprocesador y la memoria ($M/\overline{IO} = 1$) o entre el microprocesador y la E/S ($M/\overline{IO} = 0$). Además de M/\overline{IO} , los sistemas de memoria y de E/S deben leer o escribir datos. La señal W/\overline{R} es un 0 lógico para una operación de lectura y un 1 lógico para una operación de escritura. La señal \overline{ADS} se utiliza para calificar las señales de control M/\overline{IO} y W/\overline{R} . Ésta es una ligera desviación de los primeros microprocesadores Intel, que no utilizaban \overline{ADS} para la calificación.

En la figura 17-9 verá un circuito simple que genera cuatro señales de control para la memoria y los dispositivos de E/S en el sistema. Observe que se desarrollan dos señales de control para la memoria ($MRDC$ y $MWTC$) y dos para el control de la E/S ($IORC$ e $IOWC$). Estas señales son consistentes con las señales de control de memoria y E/S generadas para usarse en las versiones anteriores del microprocesador Intel.

Sincronización

La sincronización es importante para saber cómo conectar la memoria y la E/S al microprocesador 80386. La figura 17-10 muestra el diagrama de sincronización de un ciclo de lectura de memoria sin canalización. Observe que la sincronización hace referencia a la señal de entrada CLK_2 y que un ciclo de bus consiste en cuatro períodos de reloj.

FIGURA 17-9 Generación de señales de control de memoria y E/S para el 80386, el 80486 y el Pentium.

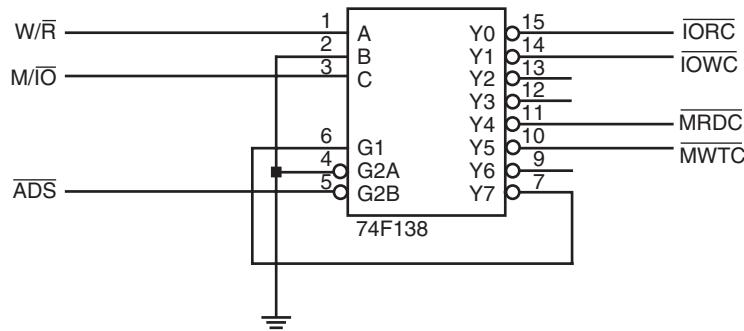
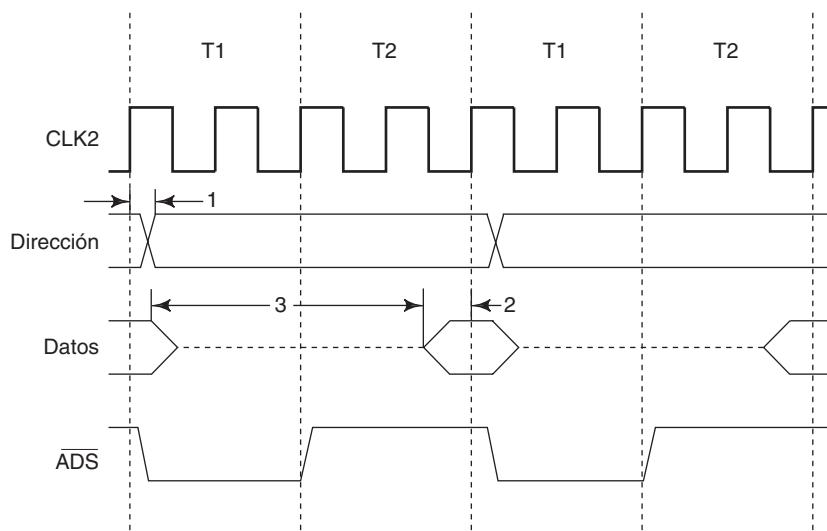


FIGURA 17-10

La sincronización de lectura sin canalización para el microprocesador 80386.



	33 MHz	25 MHz	20 MHz	16 MHz
Tiempo 1:	4–15 ns	4–21 ns	4–30 ns	4–36 ns
Tiempo 2:	5 ns	7 ns	11 ns	11 ns
Tiempo 3:	46 ns	52 ns	59 ns	78 ns

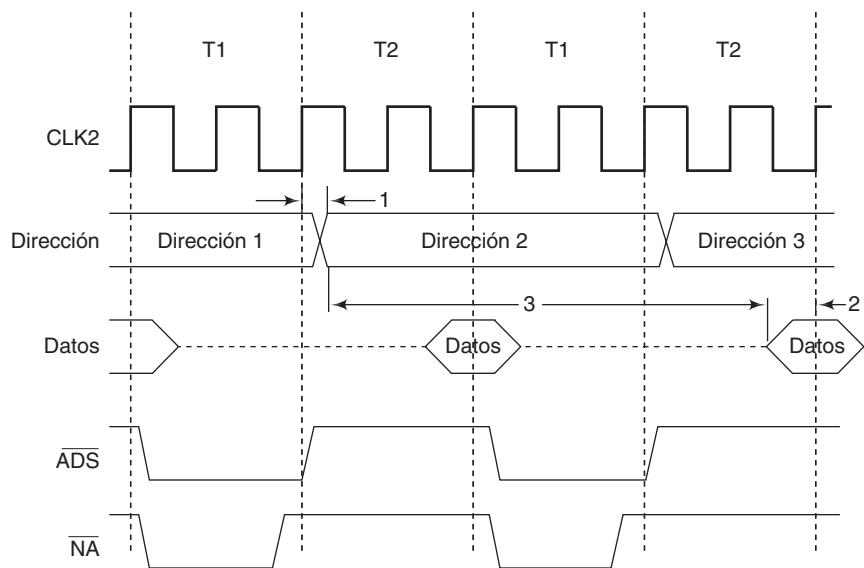
Cada ciclo de bus contiene dos estados de reloj y cada estado (T_1 y T_2), dos períodos de reloj. Observe en la figura 17-10 que el tiempo de acceso se muestra como el tiempo número 3. La versión de 16 MHz permite a la memoria un tiempo de acceso de 78 ns antes de insertar estados de espera en este modo de operación sin canalización. Para seleccionar el modo sin canalización, colocamos un 1 lógico en la terminal NA.

La figura 17-11 muestra la sincronización de lectura cuando el 80386 opera en modo de canalización. Observe que un tiempo adicional está permitido a la memoria para accesar datos debido a que las direcciones se envían primero. El modo de canalización se selecciona colocando un 0 lógico en la terminal NA y utilizando enclavamientos de dirección para capturar la dirección canalizada. El pulso de reloj que se aplica a los enclavamientos de dirección viene de la señal ADS. Los enclavamientos de dirección deben manejarse con un sistema canalizado, al igual que con bancos de memoria intercalados. En algunas aplicaciones se ha utilizado con éxito una cantidad mínima de dos y de cuatro bancos intercalados.

Observe que en la dirección canalizada aparece un estado de reloj completo antes del direccionamiento sin canalización. En la versión de 16 MHz del 80386 esto permite un tiempo adicional de 62.5 ns para el acceso a memoria. En un sistema no canalizado se permite un tiempo de acceso a memoria de 78 ns para el sistema de memoria; en un sistema canalizado se permiten 140.5 ns. Las ventajas del sistema canalizado son que no se requieren estados de espera (en muchos, pero no en todos los ciclos) y

FIGURA 17-11

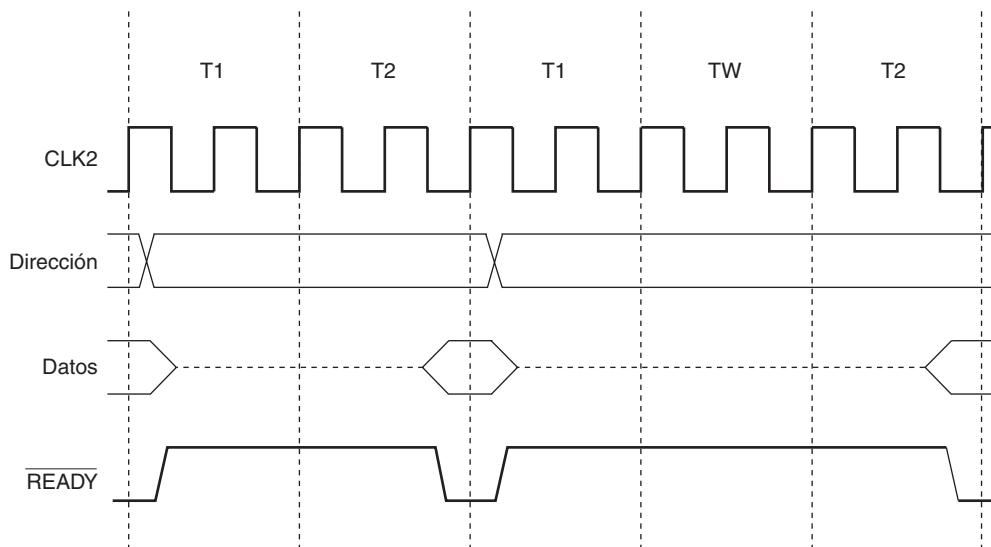
La sincronización de lectura canalizada para el microprocesador 80386.



pueden conectarse dispositivos de memoria de mucho menor velocidad al microprocesador. La desventaja es que necesitamos memoria intercalada para utilizar un canal, lo cual requiere circuitos adicionales y estados de espera ocasionales.

Estados de espera

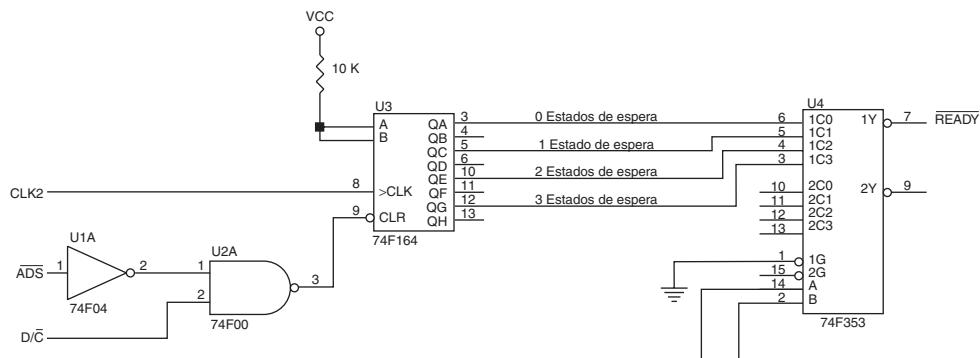
Los estados de espera son necesarios si los tiempos de acceso a la memoria son largos, en comparación con el tiempo permitido por el 80386 para el acceso a memoria. En un sistema de 33 MHz no canalizado, el tiempo de acceso a memoria es de sólo 46 ns. Hasta el momento sólo hay unas cuantas memorias DRAM, que tienen un tiempo de acceso de 46 ns. Esto significa que deben introducirse estados de espera con frecuencia para acceder a la DRAM (un estado de espera para la DRAM de 60 ns) o a una EPROM que tenga un tiempo de acceso de 100 ns (dos estados de espera). El estado de espera está integrado en una tarjeta madre y no puede eliminarse.

**FIGURA 17-12** Un 80386 no canalizado con 0 y 1 estados de espera.

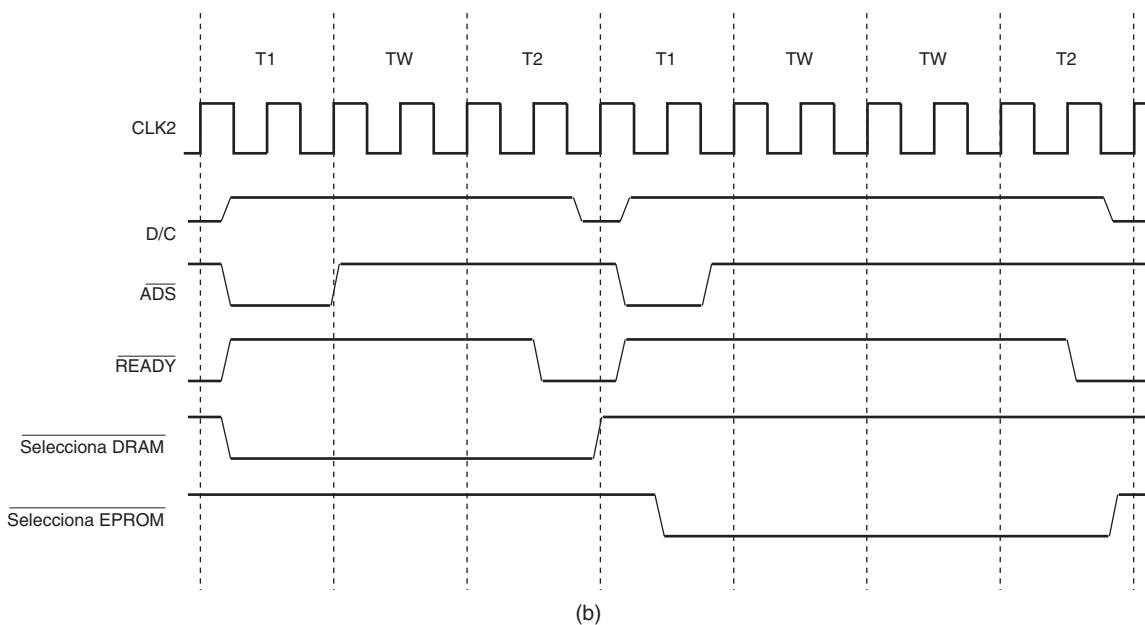
La entrada READY controla si se inserten o no estados de espera en la sincronización. La entrada READY en el 80386 es una entrada dinámica que debe activarse durante cada ciclo de bus. La figura 17-12 muestra unos cuantos ciclos de bus con un ciclo normal (sin estado de espera) y uno que contiene un solo estado de espera. Observe cómo se controla READY para producir 0 o 1 estados de espera.

La señal READY se muestrea al final de un ciclo de bus para determinar si el ciclo de reloj es T_2 o T_W . Si $\overline{READY} = 0$ en este momento, es el final del ciclo de bus o de T_2 . Si $\overline{READY} = 1$ al final de un ciclo de reloj, el ciclo es un T_W y el microprocesador continúa evaluando READY, buscando un 0 lógico y el final del ciclo de bus.

En el sistema sin canalización, cada vez que ADS se convierte en un 0 lógico, se inserta un estado de espera si READY = 1. Una vez que ADS regresa a un 1 lógico, se cuentan los bordes positivos del reloj para generar la señal READY. Esta señal se convierte en un 0 lógico después del primer ciclo de reloj para insertar 0 estados de espera. Si se inserta un estado de espera, la línea READY debe permanecer en nivel de 1 lógico hasta que hayan transcurrido por lo menos dos ciclos de reloj. Si se desean estados de espera adicionales, entonces tiene que transcurrir más tiempo antes de borrar READY. En esencia, esto permite insertar cualquier número de estados de espera en la sincronización.



(a)



(b)

FIGURA 17-13 (a) Circuito y (b) sincronización que selecciona 1 estado de espera para DRAM y 2 estados de espera para EPROM.

La figura 17-13 muestra un circuito que inserta de 0 a 3 estados de espera para varias direcciones de memoria. En el ejemplo se produce un estado de espera para el acceso a una DRAM y dos estados de espera para el acceso a una EPROM. El 74F164 se borra cada vez que ADS está en nivel bajo y que D/C está en nivel alto. Comienza el desplazamiento después de que ADS regresa al nivel de 1 lógico. A medida que se mueve, el 00000000 en el registro de desplazamiento, comienza a llenarse con 1 lógicos desde la conexión QA hasta la conexión QH. Las cuatro salidas se conectan a un multiplexor inversor que genera la señal READY activa en nivel bajo.

17-2

REGISTROS ESPECIALES DEL 80386

En el 80386 aparece una nueva serie de registros que no se encuentran en los microprocesadores Intel anteriores; éstos son los registros de control, de depuración y de prueba. Los registros de control CR₀-CR₃ controlan varias características, los registros DR₀-DR₇ facilitan la depuración y los registros TR₆-TR₇ se utilizan para probar la paginación y el uso de la caché.

Registros de control

Además de los registros EFLAGS y EIP que describimos antes, hay otros registros de control en el 80386. El registro de control 0 (CR₀) es idéntico al registro MSW (palabra de estado del equipo) del microprocesador 80286, sólo que es de 32 bits en lugar de 16. Los registros de control adicionales son CR₁, CR₂ y CR₃.

La figura 17-14 muestra la estructura de los registros de control del 80386. El registro de control CR₁ no se utiliza en el 80386, pero se reserva para productos futuros. El registro de control CR₂ almacena la dirección de página lineal de la última página utilizada antes de una interrupción por fallo de página. Por último, el registro de control CR₃ almacena la dirección base del directorio de páginas. Los 12 bits de más a la derecha de la dirección de la tabla de páginas de 32 bits contienen ceros y se combinan con el resto del registro para localizar el inicio de la tabla de páginas de 4 Kbytes.

El registro CR₀ contiene un número de bits de control especial que se definen de la siguiente manera en el 80386:

- | | |
|-----------|---|
| PG | Selecciona la traducción de tabla de páginas de direcciones lineales en direcciones físicas cuando PG = 1. La traducción de la tabla de páginas permite asignar cualquier dirección de memoria física a cualquier dirección lineal. |
| ET | Selecciona el coprocesador 80287 cuando ET = 0, o el coprocesador 80387 cuando ET = 1. Este bit se instaló debido a que el 80387 no estaba disponible cuando apareció el 80386 por primera vez. En la mayoría de los sistemas, ET se activa para indicar que hay un 80387 presente en el sistema. |
| TS | Indica que el 80386 ha conmutado tareas (en modo protegido, si se modifica el contenido de TR se coloca un 1 en TS). Si TS = 1, una instrucción del coprocesador numérico produce una interrupción tipo 7 (coprocesador no disponible). |

FIGURA 17-14 La estructura de los registros de control del microprocesador 80386.

MSW										
P G	0000000000000000	000000000000	E	T	E	M	P	E	CR0	
No se utiliza									CR1	
Dirección lineal de fallo de página									CR2	
Base del directorio de páginas				000000000000						CR3

- EM** El bit **emulación** se activa para producir una interrupción tipo 7 para cada instrucción ESC. (Las instrucciones ESCape se utilizan para codificar instrucciones para el coprocesador 80387.) Alguna vez tal característica se manejó para emular interrupciones mediante software, la función del coprocesador. La emulación reduce el costo del sistema, pero a menudo se tarda por lo menos 100 veces más en ejecutar las instrucciones del coprocesador emuladas.
- MP** Se activa para indicar que el **coprocesador está presente** en el sistema.
- PE** Se activa para seleccionar el **modo protegido** de operación para el 80386. También puede borrarse para volver a entrar al modo real. Este bit sólo puede activarse en el 80286, el cual no podría regresar al modo real sin un reinicio de hardware, que excluye su uso en la mayoría de los sistemas con el modo protegido.

Registros de depuración y prueba

La figura 17-15 muestra los conjuntos de registros de depuración y prueba. Los primeros cuatro registros de depuración contienen direcciones de punto de interrupción lineales de 32 bits. (Una dirección lineal es una dirección de 32 bits que se genera mediante una instrucción del microprocesador, que puede ser o no igual a la dirección física.) Las direcciones de punto de interrupción, que hacen posible localizar una instrucción o un dato, se comparan en forma constante con las direcciones que genera el programa. Si ocurre una concordancia, el 80386 producirá una interrupción tipo 1 (interrupción TRAP o de depuración), si lo indican los registros de depuración DR₆ y DR₇. Esta característica es una versión mucho muy expandida del atrapamiento o rastreo básico que se permitía con los primeros microprocesadores Intel a través de la interrupción tipo 1. Las direcciones de punto de interrupción son muy útiles para depurar el software con fallas. Los bits de control en DR₆ y DR₇ se definen de la siguiente manera:

- BT** Si está activo (1), la interrupción de depuración se produjo debido a una conmutación de tarea.
- BS** Si está activo, la interrupción de depuración se ocasionó por el bit TF en el registro de banderas.
- BD** Si está activo, la interrupción de depuración ocurrió por un intento de leer el registro de depuración con el bit GD activado. El bit GD protege el acceso a los registros de depuración.
- B₃-B₀** Indican cuál de las cuatro direcciones de punto de interrupción de depuración produjo la interrupción de depuración.

DR0	DR1	DR2	DR3	DR4	DR5	DR6	DR7
31	16 15	0					
DIRECCIÓN LINEAL DE PUNTO DE INTERRUPCIÓN 0							
DIRECCIÓN LINEAL DE PUNTO DE INTERRUPCIÓN 1							
DIRECCIÓN LINEAL DE PUNTO DE INTERRUPCIÓN 2							
DIRECCIÓN LINEAL DE PUNTO DE INTERRUPCIÓN 3							
Reservado por Intel. No se debe definir.							
Reservado por Intel. No se debe definir.							
0							
LEN 3 R 3 W 3 LEN 2 R 2 W 2 LEN 1 R 1 W 1 LEN 0 R 0 W 0							
0 0 G D 0 0 0 0 G E L 3 3 G L 2 2 G L 1 0 G L 0 0							
31	16 15	0					
TR6	TR1	TR2	TR3	TR4	TR5	TR6	TR7
12 11							
V D D # U U # W W # 0 0 0 0 C							
DIRECCIÓN LINEAL							
DIRECCIÓN FÍSICA							
0 0 0 0 0 0 0 P L REP 0 0							
31	12	11	0				

FIGURA 17-15 Los registros de depuración y prueba del 80386. (Cortesía de Intel Corporation.)

LEN	Cada uno de los cuatro campos de longitud pertenece a cada una de las cuatro direcciones de punto de interrupción almacenadas en DR ₀ -DR ₃ . Estos bits definen el tamaño del acceso en la dirección del punto de interrupción como 00 (byte), 01 (palabra) o 11 (doble palabra).
RW	Cada uno de los cuatro campos de lectura/escritura pertenece a cada una de las cuatro direcciones de punto de interrupción almacenadas en DR ₀ -DR ₃ . El campo RW selecciona la causa de la acción que habilitó una dirección de punto de interrupción como 00 (acceso de instrucción), 01 (escritura de datos) y 11 (lectura y escritura de datos).
GD	Si está activo, GD evita la lectura o escritura de un registro de depuración mediante la generación de la interrupción de depuración. Este bit se borra de manera automática durante la interrupción de depuración, de forma que los registros de depuración puedan leerse o modificarse, en caso de ser necesario.
GE	Si está activo, selecciona una dirección de punto de interrupción global para cualquiera de los cuatro registros de dirección de punto de interrupción.
LE	Si está activo, selecciona una dirección de punto de interrupción local para cualquiera de los cuatro registros de dirección de punto de interrupción.

Los registros de prueba TR₆ y TR₇ se utilizan para probar el **búfer de traducción adelantada** (TLB). El TLB se maneja con la unidad de paginación dentro del 80386. El TLB, que almacena las traducciones de las direcciones de la tabla de páginas que se utilizan con más frecuencia, reduce el número de lecturas de memoria requeridas para buscar direcciones de traducción de página en las tablas de traducción de páginas. El TLB almacena las 32 entradas más comunes de la tabla de páginas, lo que se prueba con los registros de prueba TR₆ y TR₇.

El registro de prueba TR₆ almacena el campo de identificación (dirección lineal) del TLB y el TR₇ almacena la dirección física del TLB. Para escribir una entrada en el TLB, realice los siguientes pasos:

1. Escriba TR₇ para los valores deseados de dirección física, PL y REP.
2. Escriba TR₆ con la dirección lineal, asegurándose de que C = 0.

Para leer una entrada en el TLB:

1. Escriba TR₆ con la dirección lineal, asegurándose de que C = 1.
2. Lea TR₆ y TR₇. Si el bit PL indica una coincidencia, entonces los valores deseados de TR₆ y TR₇ indican el contenido del TLB.

Los bits que se encuentran en TR₆ y en TR₇ indican las siguientes condiciones:

V	Indica que la entrada en el TLB es válida.
D	Indica que la entrada en el TLB es inválida o mala.
U	Un bit para el TLB.
W	Indica que se puede escribir en el área direccionada por la entrada TLB.
C	Selecciona la operación de escritura (0) o de búsqueda inmediata (1) para el TLB.
PL	Indica una coincidencia si es un 1 lógico.
REP	Selecciona el bloque del TLB en el que se va a escribir.

En la sección sobre administración de memoria y la unidad de paginación encontrará más detalles acerca el funcionamiento del TLB.

realiza la tarea de convertir las direcciones lineales, a medida que aparecen como salidas de un programa, en direcciones físicas que acceden a una posición en memoria física que puede estar en cualquier parte dentro del sistema de memoria. El 80386 maneja el mecanismo de paginación para asignar cualquier dirección física a cualquier dirección lógica. Por lo tanto, aun cuando el programa accede a la posición de memoria A0000H con una instrucción, la dirección física actual sería la posición de memoria 100000H o cualquier otra si está habilitada la paginación. Esta característica permite que casi cualquier software escrito para operar en cualquier posición de memoria funcione en un 80386, ya que cualquier posición lineal se convertiría en cualquier posición física. Los primeros microprocesadores Intel no tenían tal flexibilidad. La paginación se utiliza con DOS para reubicar la memoria del 80386 y del 80486 en las direcciones superiores a FFFFFH y en espacios entre las ROMs, en las posiciones D0000-DFFFFH y en otras áreas, a medida que sean disponibles. Por lo general, el área entre las ROMs se conoce como *memoria superior*; el área superior a FFFFFH se conoce como *memoria extendida*.

Descriptoros y selectores

Antes de hablar sobre la unidad de paginación de memoria, examinaremos el descriptor y el selector para el microprocesador 80386. Este microprocesador utiliza descriptoros en forma muy parecida al 80286. En ambos microprocesadores, un **descriptor** es una serie de ocho bytes que describen y ubican un segmento de memoria. Un **selector** (registro de segmento) se maneja para indizar un descriptor de una tabla de descriptoros. La principal diferencia entre el 80286 y el 80386 es que el segundo tiene dos selectores adicionales (FS y GS); además, los dos bytes más significativos del descriptor están definidos para el 80386. Otra diferencia es que los descriptoros del 80386 emplean una dirección base de 32 bits y un límite de 20 bits, en lugar de la dirección base de 24 bits y el límite de 16 bits del 80286.

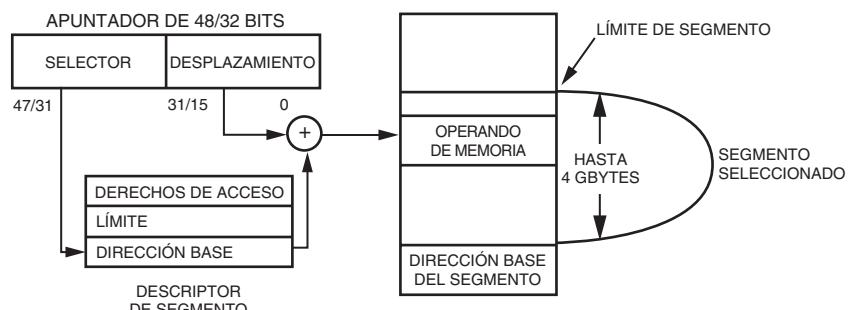
El 80286 direcciona un espacio de memoria de 16 Mbytes con su dirección base de 24 bits y tiene un límite de 64 Kbytes en la longitud de segmento, debido al límite de 16 bits. El 80386 direcciona un espacio de memoria de 4 Gbytes con su dirección base de 32 bits y tiene un límite de 1 Mbyte o de 4 Gbytes en la longitud de segmento, debido al límite de 20 bits que se maneja de dos maneras. Con el límite de 20 bits puedes posible acceder a un segmento con una longitud de 1 Mbyte, si el bit de granularidad (G) = 0. Si G = 1, el límite de 20 bits permite una longitud de segmento de 4 Gbytes.

El bit de granularidad se encuentra en el descriptor del 80386. Si G = 0, el número que se almacena en el límite se interpreta directamente como un límite, lo cual le permite soportar cualquier límite entre 00000H y FFFFFH para un tamaño de segmento de hasta 1 Mbyte. Si G = 1, el número que se almacena en el límite se interpreta como 00000XXXH-FFFFFXXXH, en donde XXX es cualquier valor entre 000H y FFFH. Esto sirve para que el límite del segmento varíe entre 0 bytes y 4 Gbytes, en intervalos de 4 Kbytes. Un límite de 00001H indica que el límite es de 4 Kbytes cuando G = 1 y de 1 byte cuando G = 0. Un ejemplo es un segmento que comienza en la dirección física 10000000H. Si el límite es 00001H y G = 0, este segmento comienza en 10000000H y termina en 10000001H. Si G = 1 con el mismo límite (00001H), el segmento comienza en la posición 10000000H y termina en la posición 10001FFFH.

La figura 17-6 muestra cómo el 80386 direcciona un segmento de memoria en el modo protegido, mediante el uso de un selector y un descriptor. Esto es idéntico a la manera en que el 80286 direcciona un segmento. La diferencia es el tamaño del segmento del 80386. El selector maneja sus 13 bits de más a la izquierda para seleccionar un descriptor de una tabla de descriptoros. El bit TI indica la tabla de

FIGURA 17-16

Direccionamiento en modo protegido, mediante el uso de un registro de segmento como selector.



descriptores locales ($TI = 1$) o globales ($TI = 0$). Los dos bits de más a la derecha del selector definen el nivel de privilegio solicitado del acceso.

Como el selector utiliza un código de 13 bits para acceder a un descriptor, hay como máximo 8192 descriptores en cada tabla, locales o globales. Como cada segmento (en un 80386) puede ser de 4 Gbytes de longitud, es posible acceder a 16,384 segmentos a la vez con las dos tablas de descriptores. Esto permite al 80386 acceder a un tamaño de memoria virtual de 64 Tbytes. Claro que en realidad sólo existen 4 Gbytes de memoria en el sistema de memoria (1 Tbyte = 1024 Gbytes). Si un programa requiere más de 4 Gbytes de memoria a la vez, puede intercambiarse entre el sistema de memoria y una unidad de disco o cualquier otra forma de almacenamiento de gran volumen.

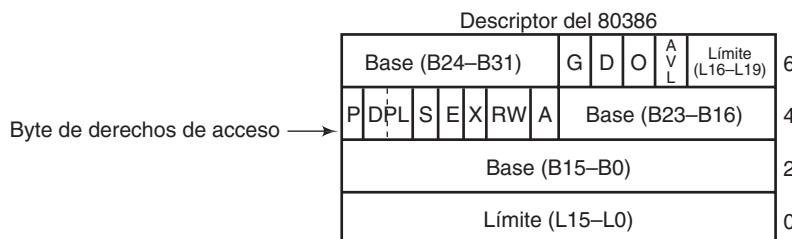
El 80386 utiliza tablas de descriptores para los descriptores globales (GDT) y los locales (LDT). Hay una tercera tabla de descriptores para los descriptores de interrupción (IDT) o compuertas. Los primeros seis bytes del descriptor son iguales que en el 80286, lo cual permite que el software del 80286 sea compatible en forma ascendente con el 80386. (Un descriptor del 80286 utilizaba 00H para sus dos bytes más significativos.) En la figura 17-17 verá el descriptor del 80286 y el del 80386. La dirección base es de 32 bits en el 80386, el límite es de 20 bits y un bit G selecciona el multiplicador de límite (1 o 4 K veces). Los campos en el descriptor para el 80386 se definen de la siguiente manera:

Límite ($L_{19}-L_0$)	Define la dirección inicial de 32 bits del segmento dentro del espacio de direcciones físicas de 4 Gbytes del microprocesador 80386. También, el límite del segmento en unidades de bytes, si el bit G = 0, o en unidades de 4 Kbytes, si G = 1. Esto permite que un segmento sea de cualquier longitud, desde 1 byte hasta 1 Mbyte, si G = 0, y de 4 Kbytes a 4 Gbytes, si G = 1. Recuerde que el límite indica el último byte en un segmento.
Derechos de acceso	Determina el nivel de privilegio y demás información acerca del segmento. Este byte varía con los distintos tipos de descriptores y se elabora con el tipo de cada descriptor.
G	El bit de granularidad selecciona un multiplicador de 1 o 4 K veces para el campo del límite. Si G = 0, el multiplicador es 1; si G = 1, el multiplicador es 4 K.
D	Selecciona el modo de instrucción predeterminado. Si D = 0, los registros y apuntadores de memoria son de 16 bits, como en el 80286; si D = 1, son de 32 bits, como en el 80386. Este bit determina si se requieren prefijos para datos de 32 bits y registros índice. Si D = 0, entonces se necesita un prefijo para acceder a los registros de 32 bits y para utilizar apuntadores de 32 bits. Si D = 1 entonces se requiere un prefijo para acceder a los registros de 16 bits y a los apuntadores de 16 bits. Las directivas USE ₁₆ y USE ₃₂ , que se anexan a la instrucción SEGMENT en lenguaje ensamblador, controlan el ajuste del bit D. En el modo real siempre se asume que los registros son de 16 bits, por lo que cualquier instrucción que haga referencia a un registro o apuntador de 32 bits debe llevar un prefijo. La versión actual de DOS asume que D = 0, en tanto que la mayoría de los programas Windows asumen que D = 1.
AVL	El bit está disponible para que el sistema operativo lo utilice de la forma que crea conveniente. A menudo indica que el segmento descrito por el descriptor está disponible.

Descriptor del 80286		Descriptor del 80386				
Reservado		Base (B24–B31)				6
Derechos de acceso	Base (B23–B16)	G D O A V L				6
Base (B15–B0)		Derechos de acceso				4
Límite ($L_{15}-L_0$)		Base (B23–B16)				4
		Base (B15–B0)				2
		Límite ($L_{15}-L_0$)				0

FIGURA 17-17 Los descriptores para los microprocesadores 80286 y 80386.

FIGURA 17-18 El formato del descriptor de segmento del 80386.



En el microprocesador 80386, los descriptores aparecen en dos formas: el descriptor de segmento y el descriptor de sistema. El descriptor de segmento define los segmentos de datos, de pila y de código; el descriptor de sistema define información acerca de las tablas, tareas y compuertas del sistema.

Descriptores de segmento. La figura 17-18 muestra el descriptor de segmento. Este descriptor se ajusta al formato general, como lo indica la figura 17-17, pero los bits de derechos de acceso se definen de manera que se indique el funcionamiento del segmento de datos, de pila o de código descrito por el descriptor. La posición de bit 4 del byte de derechos de acceso determina si el descriptor es un descriptor del segmento de datos o de código ($S = 1$), o un descriptor de segmento del sistema ($S = 0$). Tenga en cuenta que las etiquetas que se utilizan para estos bits pueden variar en distintas versiones de la literatura de Intel, aunque realizan las mismas tareas.

A continuación se muestra una descripción de los bits de derechos de acceso y su función en el descriptor de segmento:

- P** El bit **presente** es un 1 lógico que indica que el segmento está presente. Si $P = 0$ y se accede al segmento mediante el descriptor, se produce una interrupción tipo 11. Tal interrupción indica que se accedió a un segmento que no está en el sistema.
- DPL** El bit **nivel de privilegio del descriptor** establece el nivel de privilegio del descriptor; 00 tiene el privilegio más alto y 11 el más bajo. Esto se utiliza para proteger el acceso a los segmentos. Si se accede a un segmento con un nivel de privilegio que sea menor (mayor en número) que el DPL, se produce una interrupción por violación de privilegio. Los niveles de privilegio se manejan en sistemas multiusuario para evitar el acceso a un área de la memoria del sistema.
- S** **Segmento** indica un descriptor de segmento de datos o de código ($S = 1$) o un descriptor de segmento del sistema ($S = 0$).
- E** **Ejecutable** selecciona un segmento de datos (pila) ($E = 0$) o un segmento de código ($E = 1$). E también define la función de los dos bits siguientes (X y RW).
- X** Si $E = 0$, entonces X indica la dirección de **expansión** del segmento de datos. Si $X = 0$, el segmento se expande hacia arriba, como en un segmento de datos; si $X = 1$ el segmento se expande hacia abajo, como en un segmento de pila. Si $E = 1$, entonces X indica si se debe ignorar el nivel de privilegio del segmento de código ($X = 0$) o si se debe observar ($X = 1$).
- RW** Si $E = 0$, entonces el bit de **lectura/escritura** (RW) indica si se puede escribir ($RW = 1$) o no ($RW = 0$) en el segmento de datos. Si $E = 1$, entonces RW indica si es posible leer ($RW = 1$) o no ($RW = 0$) el segmento de código.
- A** El bit **acceso** se activa cada vez que el microprocesador accede al segmento. Algunas veces lo utiliza el sistema operativo para llevar la cuenta del acceso a los segmentos.

Descriptor del sistema. En la figura 17-19 se muestra el descriptor del sistema. Hay 16 tipos de descriptores de sistema posibles (en la tabla 17-1 se verá los distintos tipos de descriptores), aunque no todos se utilizan en el microprocesador 80386. Algunos de estos tipos se definen para el 80286, con el objetivo de que el software de este microprocesador sea compatible con el 80386. Algunos de los tipos son nuevos y únicos para el 80386; otros no se han definido todavía y se reservan para futuros productos Intel.

FIGURA 17-19 El formato general de un descriptor de sistema del 80386.

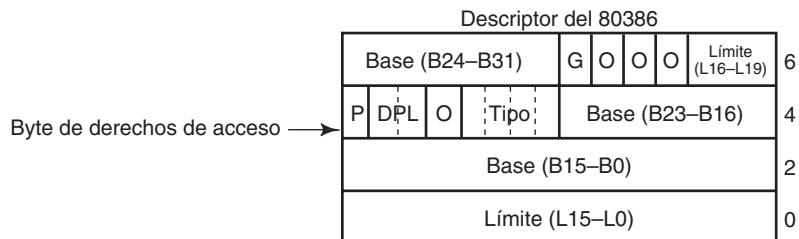


TABLA 17-1 Tipos de descriptores de sistema.

	<i>Tipo</i>	<i>Propósito</i>
0000		Inválido
0001		TSS del 80286 disponible
0010		LDT
0011		TSS del 80286 ocupado
0100		Compuerta de tareas del 80286
0101		Compuerta de tareas (80386 y superiores)
0110		Compuerta de interrupciones del 80286
0111		Compuerta de trampas del 80286
1000		Inválido
1001		TSS del 80386 y superiores disponible
1010		Reservado
1011		TSS del 80386 y superiores ocupado
1100		Compuerta CALL del 80386 y superiores
1101		Reservado
1110		Compuerta de interrupciones del 80386 y superiores
1111		Compuerta de trampas del 80386 y superiores

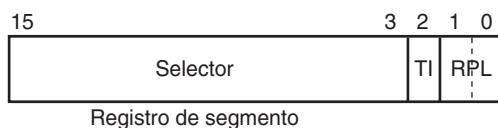
Tablas de descriptores

Las tablas de descriptores definen todos los segmentos que se utilizan en el 80386, cuando éste opera en el modo protegido. Hay tres tipos de tablas de descriptores: la tabla de descriptores globales (GDT), la tabla de descriptores locales (LDT) y la tabla de descriptores de interrupción (IDT). Los registros que utiliza el 80386 para direccionar estas tres tablas se llaman: registro de tabla de descriptores globales (GDTR), registro de tabla de descriptores locales (LDTR) y registro de tabla de descriptores de interrupción (IDTR). Tales registros se cargan con sus respectivas instrucciones LGDT, LLDT y LIDT.

La **tabla de descriptores** es un arreglo de datos de longitud variable, en donde cada entrada almacena un descriptor de 8 bytes. Las tablas de descriptores locales y globales almacenan hasta 8192 entradas cada una, en tanto que la tabla de descriptores de interrupción almacena hasta 256 entradas. Un descriptor se indiza desde la tabla de descriptores locales o globales mediante el selector que aparece en un registro de segmento. La figura 17-20 muestra un registro de segmento y el selector que almacena en el modo protegido. Los 13 bits de más a la izquierda indizan a un descriptor; 1 bit TI selecciona la tabla de descriptores locales (TI = 1) o globales (TI = 0); los bits RPL indican el nivel de privilegio solicitado.

Cada vez que se coloca un nuevo selector en uno de los registros de segmento, el 80386 accede a una de las tablas de descriptores y carga el descriptor de manera automática en una porción de caché del registro de segmento, la cual es invisible para el programa. Mientras el selector siga siendo el mismo en el registro de segmento, no se requieren accesos adicionales a la tabla de descriptores. La operación

FIGURA 17-20 Un registro de segmento que muestra el selector, el bit T₁ y los bits de nivel de privilegio (RPL).



de obtener un nuevo descriptor de la tabla de descriptores es invisible para el programa, ya que el microprocesador realiza esto de manera automática cada vez que se modifica el contenido del registro de segmento en el modo protegido.

La figura 17-21 muestra la forma de acceder a una tabla de descriptores globales (GDT) de ejemplo (la cual se almacena en la dirección de memoria 00010000H) a través del registro de segmento y su selector. Esta tabla contiene cuatro entradas. La primera es un descriptor nulo (0). El descriptor 0 siempre debe ser nulo. Las demás entradas direccionan varios segmentos en el sistema de memoria del 80386 en modo protegido. En la ilustración, el registro del segmento de datos contiene 0008H. Esto significa que el selector está indizando la posición del descriptor 1 en la tabla de descriptores globales (TI = 0), con un nivel de privilegio solicitado de 00. El descriptor 1 se ubica ocho bytes más arriba de la dirección base de la tabla de descriptores, comenzando en la posición 00010008H. El descriptor ubicado en esta posición de memoria accede a una dirección base de 00200000H y un límite de 100H. Lo anterior significa que este descriptor direcciona las posiciones de memoria 00200000H-00200100H. Como éste es el registro DS (segmento de datos), el segmento de datos se ubica en dichas posiciones del sistema de memoria. Si se accede a datos que estén fuera de estos límites, se produce una interrupción.

El acceso a la tabla de descriptores locales (LDT) es de la misma forma que para la tabla de descriptores globales (GDT). La única diferencia es que el bit TI es 0 para un acceso global y es 1 para un acceso local. Si examinamos los registros de las tablas de descriptores locales y globales, encontraremos otra diferencia. El registro de la tabla de descriptores globales (GDTR) contiene la dirección base de la tabla de descriptores globales y el límite. El registro de la tabla de descriptores locales (LDTR) contiene sólo un selector de 16 bits. El contenido del LDTR dirige un descriptor de sistema tipo 0010, que contiene la dirección base y el límite de la LDT. Este esquema permite una tabla global para todas las tareas, pero también muchas tablas locales (una o más para cada tarea) si es necesario. Los descriptores

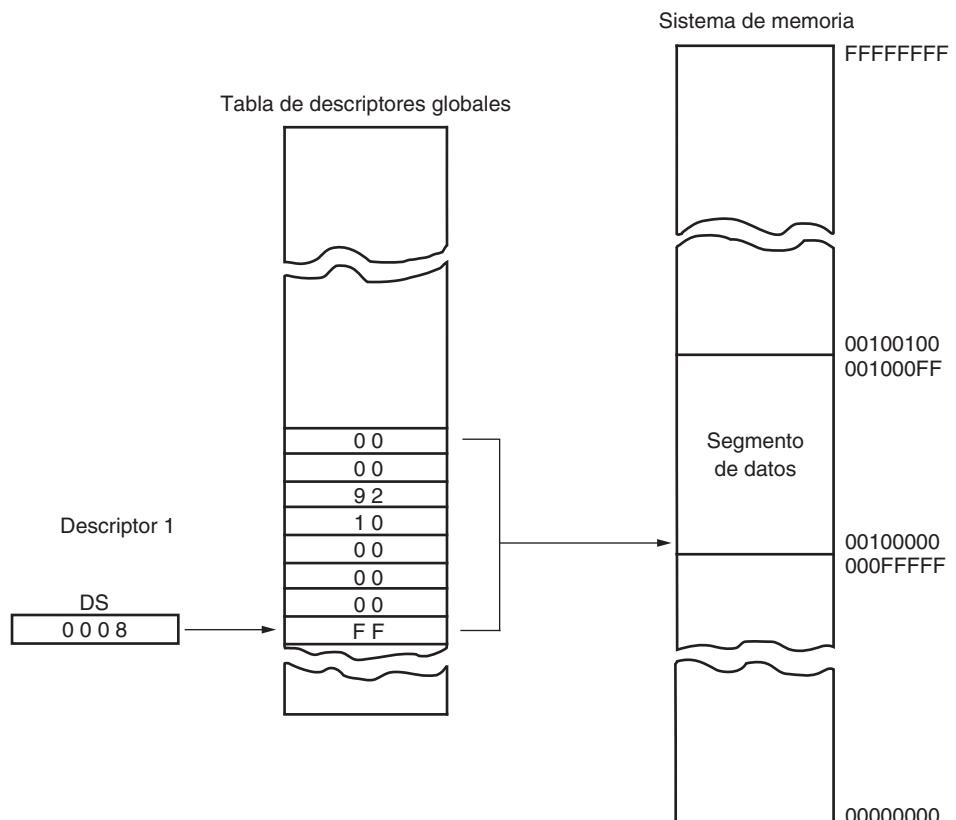
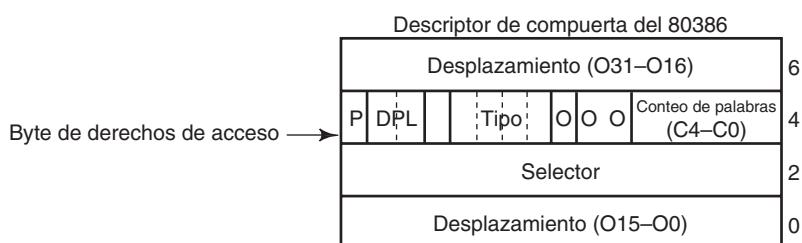


FIGURA 17-21 Uso del registro DS para seleccionar un descriptor a partir de la tabla de descriptores globales. En este ejemplo, el registro DS accede a las posiciones de memoria 00100000H-001000FFH como un segmento de datos.

FIGURA 17-22 El descriptor de compuerta para el microprocesador 80386.



globales describen la memoria para el sistema, mientras que los descriptores locales describen la memoria para las aplicaciones o tareas.

Al igual que la GDT, la tabla de descriptores de interrupción (IDT) se direcciona mediante el almacenamiento de la dirección base y el límite en el registro de la tabla de descriptores de interrupción (IDTR). La principal diferencia entre la GDT y la IDT es que la IDT sólo contiene compuertas de interrupciones. La GDT y la LDT contienen descriptores de segmento y de sistema, pero nunca compuertas de interrupciones.

La figura 17-22 muestra el descriptor de compuerta, una forma especial del descriptor de sistema que ya describimos. (En la tabla 17-1 consultará los distintos tipos de descriptores de compuerta.) Observe que el descriptor de compuerta contiene una dirección de desplazamiento de 32 bits, un conteo de palabras y un selector. La dirección de desplazamiento de 32 bits apunta hacia la ubicación del procedimiento de servicio de interrupciones o de otro procedimiento. El conteo de palabras indica cuántas palabras se transfieren de la pila del que hizo la llamada a la pila del procedimiento al que se accede mediante una compuerta de llamada. Esta característica de transferir datos desde la pila del que hace la llamada es útil para instrumentar lenguajes de alto nivel tales como C/C++. El campo conteo de palabras no se maneja con una compuerta de interrupción. El selector se utiliza para indicar la ubicación del segmento de estado de tarea (TSS) en la GDT o en la LDT si es un procedimiento local.

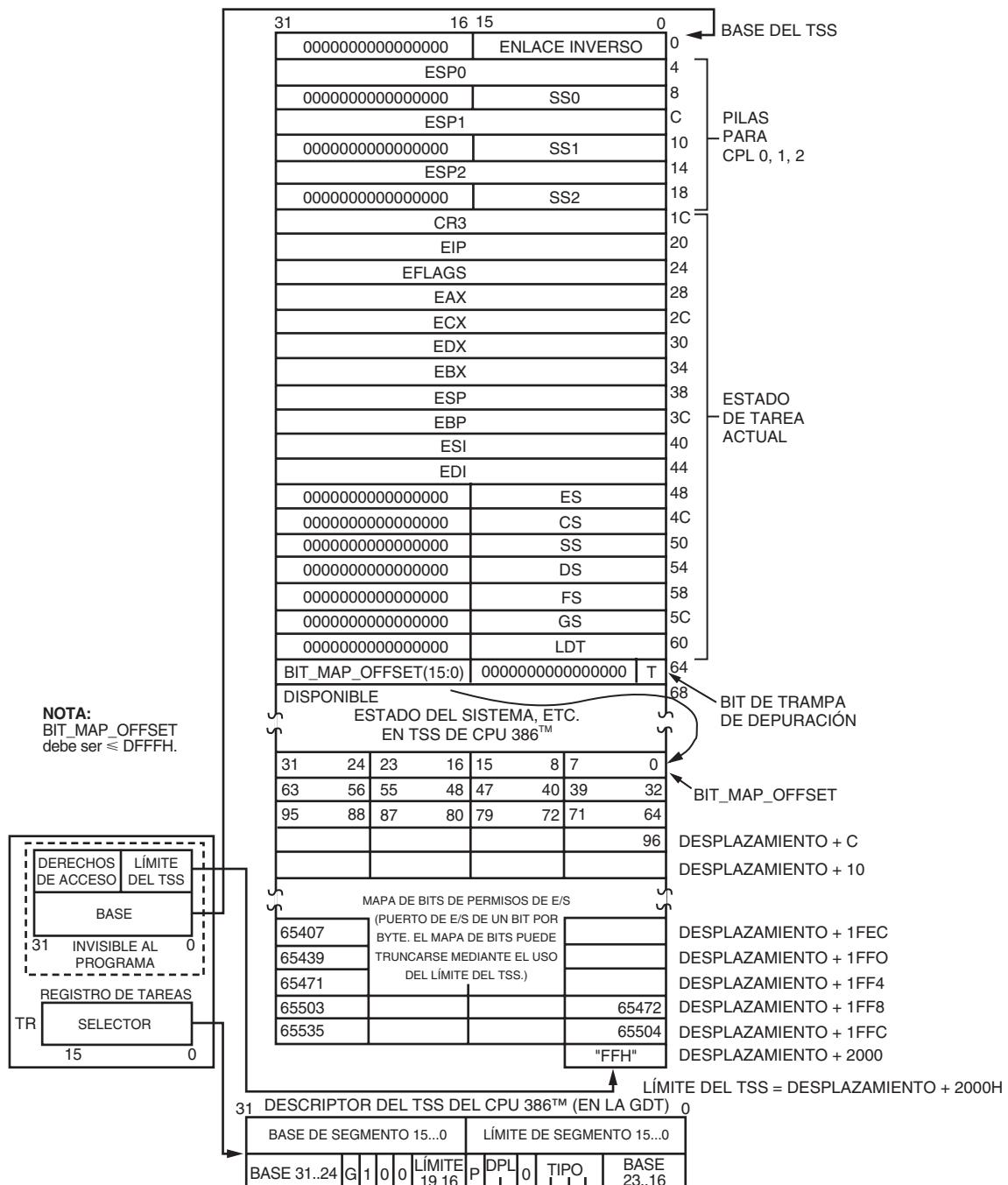
Cuando se accede a una compuerta, el contenido del selector se carga en el registro de tareas (TR), con lo que se produce una conmutación de tareas. La aceptación de la compuerta depende de los niveles de privilegio y prioridad. Una instrucción de retorno (RET) termina un procedimiento de compuerta de llamada y un retorno de la instrucción de interrupción (IRET) termina un procedimiento de compuerta de interrupción. Por lo general, se accede a la tarea mediante una instrucción CALL o una INT, en donde la instrucción de llamada direcciona a una compuerta de llamada en la tabla de descriptores y la interrupción direcciona un descriptor de interrupciones.

La diferencia entre las interrupciones en modo real y las interrupciones en modo protegido es que la tabla de vectores de interrupción es una IDT en el modo protegido. La IDT contiene hasta 256 niveles de interrupción, pero se accede a cada nivel a través de una compuerta de interrupción, en vez de usar un vector de interrupción. Por lo tanto, el número de tipo de vector 2 (INT 2) se ubica en el número de descriptor 2 de la IDT, a 16 posiciones por encima de la dirección base de la IDT. Esto también significa que el primer 1 Kbyte de memoria ya no contiene vectores de interrupción como en el modo real. La IDT puede colocarse en cualquier ubicación del sistema de memoria.

El segmento de estado de tarea (TSS)

El descriptor del **segmento de estado de tarea** (TSS) contiene información acerca de la ubicación, el tamaño y el nivel de privilegios del segmento de estado de tarea, de igual forma que cualquiera otro descriptor. La diferencia es que la TSB que describe el descriptor de TSS no contiene datos ni código. Contiene el estado de la tarea y los enlaces para que las tareas se aniden (una tarea puede llamar a una segunda tarea, la cual a su vez llamaría a una tercera, y así sucesivamente). El descriptor del TSS se dirige mediante el **registro de tareas** (TR). El contenido del TR se modifica mediante la instrucción LTR. Cada vez que el programa en modo protegido ejecuta una instrucción JMP o CALL, también se modifica el contenido del TR. La instrucción LTR se utiliza para acceder por primera vez a una tarea durante la inicialización del sistema. Después de la inicialización, por lo general las instrucciones CALL o JUMP comutan tareas. En la mayoría de los casos utilizamos la instrucción CALL para iniciar una nueva tarea.

En la figura 17-23 se muestra el TSS. Como se ve, el TSS es una formidable estructura de datos, la cual contiene muchos tipos distintos de información. La primera palabra del TSS se identifica como **enlace inverso**. Éste es el selector que se utiliza en un retorno (RET o IRET) para realizar un enlace de regreso con la TSS anterior; para ello se carga el selector de enlace inverso en el TR. La siguiente palabra debe contener un 0. Las dobles palabras de la segunda a la séptima contienen los valores de ESP



Tipo = 9: TSS del CPU 386™ disponible.

Tipo = B: TSS del CPU 386™ ocupado.

FIGURA 17-23 El descriptor del segmento de estado de tarea (TSS). (Cortesía de Intel Corporation.)

y ESS para los niveles de privilegios 0-2. Éstos se requieren en caso de que se interrumpa la tarea actual, para que acceder a estas pilas de nivel de privilegio (PL). La octava palabra (desplazamiento 1CH) almacena el contenido de CR₃, el cual guarda la dirección base del registro del directorio de páginas del estado anterior. Éste debe restaurarse si la paginación está en efecto. El contenido de las 17 dobles palabras siguientes se carga en los registros indicados. Cada vez que se accede a una tarea, se guarda todo el estado del equipo (todos los registros) en estas posiciones de memoria y después se vuelven a cargar desde las mismas posiciones en el TSS nuevo. La última palabra (desplazamiento 66H) contiene la dirección base del mapa de bits de permisos de E/S.

Este mapa de bits permite que el TSS bloquee las operaciones de E/S para las direcciones de los puertos de E/S inhibidos a través de una interrupción de negación de permisos de E/S. Dicha interrupción de negación de permisos es el número de tipo de interrupción 13, la interrupción por fallo de protección general. La dirección base del mapa de bits de permisos de E/S es la dirección de desplazamiento a partir del inicio del TSS. Lo anterior permite que muchos TSSs utilicen el mismo mapa de permisos.

Cada mapa de bits de permisos de E/S es de 64 K bits (8 Kbytes), comenzando en la dirección de desplazamiento que indica la dirección base del mapa de bits de permisos de E/S. El primer byte del mapa de bits de permisos de E/S contiene los permisos de E/S para los puertos de E/S 0000H-0007H. El bit de más a la derecha contiene el permiso para el puerto número 0000H. El bit de más a la izquierda, el permiso para el puerto número 0007H. Esta secuencia continúa para las últimas direcciones de puerto (FFFFH) que se almacenan en el bit de más a la izquierda del último byte del mapa de bits de permisos de E/S. Si se coloca un 0 lógico en un bit del mapa de bits de permisos de E/S se habilita la dirección de puerto de E/S, mientras que un 1 lógico inhibe o bloquea la dirección de puerto de E/S. Hasta hoy, sólo Windows NT, Windows 2000 y Windows XP utilizan el esquema de permisos de E/S para deshabilitar los puertos de E/S que dependen de la aplicación o del usuario.

Para revisar la operación de un conmutador de tareas, que requiere sólo 17 µs para ejecutarse en un microprocesador 80386, listaremos los siguientes pasos:

1. La compuerta contiene la dirección del procedimiento o de la ubicación hacia la que se salta mediante el conmutador de tareas. También, el número de selector del descriptor del TSS y el número de palabras que se transfieren del proceso que hace la llamada hacia el área de la pila del usuario, para el paso de parámetros.
2. El selector se carga en TR desde la compuerta. (Este paso se logra mediante una instrucción CALL o una JMP, que hace referencia a un descriptor TSS válido.)
3. El TR selecciona el TSS.
4. El estado actual se almacena en el TSS actual y se accede al nuevo TSS para cargar el estado de la nueva tarea (todos los registros) en el microprocesador. El estado actual se almacena en el selector del TSS que se encuentra en el TR en ese momento. Una vez que se almacena el estado actual, se carga un nuevo valor (mediante JMP o CALL) para el selector del TSS en TR y el nuevo estado se carga desde el nuevo TSS.

El retorno de una tarea se lleva a cabo mediante los siguientes pasos:

1. Se almacena el estado del microprocesador en el TSS actual.
2. Se carga el selector de enlace inverso en el TR para acceder al TSS anterior, de manera que regrese al estado anterior del equipo y que se restaure en el microprocesador. La instrucción IRET se encarga del retorno de la llamada de un TSS.

17-4

CAMBIO AL MODO PROTEGIDO

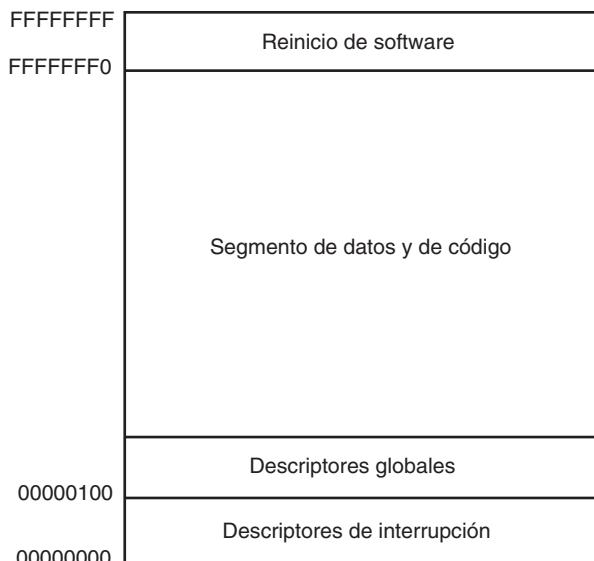
Para cambiar la operación del 80386 del modo real al modo protegido, hay que realizar varios pasos. Es posible acceder a la operación en modo real, después de un reinicio de hardware o después de cambiar el bit PE a un 0 lógico en CR₀. Para acceder al modo protegido, se coloca un 1 lógico en el bit PE de CR₀; sin embargo, antes de hacer esto deben inicializarse otros procesos. Los siguientes pasos realizan el cambio del modo real al modo protegido:

1. Inicialice la tabla de descriptores de interrupción, de manera que contenga compuertas de interrupción válidas por lo menos para los primeros 32 números de tipo de interrupción. El IDT llega a (y a menudo así es) contener hasta 256 compuertas de interrupción de ocho bytes, las cuales definen los 256 tipos de interrupción.
2. Inicialice la tabla de descriptores globales (GDT), de manera que contenga un descriptor nulo en el descriptor 0 y descriptores válidos para cuando menos un segmento de código, uno de pila y otro de datos.
3. Cambie al modo protegido mediante la activación del bit PE en CR₀.
4. Ejecute una instrucción JMP entre segmentos (lejana) para vaciar la cola de instrucciones interna.
5. Cargue todos los selectores de datos (registros de segmento) con sus valores de selector inicial.
6. Ahora el 80386 estará operando en modo protegido y utilizará los descriptores de segmento definidos en la GDT y la IDT.

La figura 17-24 muestra la configuración del mapa de memoria del sistema en modo protegido que se obtiene al seguir los pasos del 1 al 6. En el ejemplo 17-1 se muestra el software para dicha tarea. Este sistema contiene un descriptor de segmento de datos y un descriptor de segmento de código; cada segmento es de 4 Gbytes. Éste es el sistema en modo protegido más sencillo que puede obtenerse (que se conoce como **modelo plano**): se cargan todos los registros de segmento, excepto el de código, con el mismo descriptor del segmento de datos de la GDT. El nivel de privilegios se inicializa con 00, el nivel más alto. Este sistema se utiliza con más frecuencia cuando un usuario tiene acceso al microprocesador y requiere el espacio de memoria completo. El programa se diseñó para usarse en un sistema sin DOS o sin la interfaz de DOS desde Windows. Más adelante, en esta sección, veremos cómo entrar al modo protegido en un entorno de DOS. (Tome en cuenta que el software del ejemplo 17-1 se diseñó para un sistema independiente tal como el microprocesador integrado 80386EX, y no para usarse en la PC.)

En el ejemplo 17-1 no se almacenan vectores de interrupción en la tabla de descriptores de interrupción, porque no se utilizan. Si se manejan vectores de interrupción, entonces tiene que incluirse software para cargar las direcciones de los procedimientos de servicio de interrupciones en la IDT. El software debe generarse como dos piezas separadas que luego se conecten entre sí y se quemen en una ROM. La primera parte habrá que escribirla como se muestra en el modo real y la segunda (vea el comentario en el listado), con el conjunto de ensamblador para generar código en modo protegido mediante el uso del modelo plano de 32 bits. Este software no funcionará en una computadora personal, ya que está escrito para funcionar en un sistema integrado. El código debe convertirse en un archivo binario con el uso de EXE2BIN después de ensamblarlo y antes de quemarlo en una ROM.

FIGURA 17-24 El mapa de memoria para el ejemplo 17-1.



EJEMPLO 17-1

```

.MODEL SMALL
.386P
DIREC    STRUC
        DW      ?          ;estructura de dirección
        DD      ?
DIREC    ENDS

.DATA
IDT      DQ      32 DUP(?)           ;tabla de descriptores de interrupción
GDT      DQ      8                  ;tabla de descriptores globales
DESC1    DW      0FFFFH             ;descriptor del segmento de código
        DW      0
        DB      0
        DB      9EH
        DB      8FH
        DB      0
DESC2    DW      0FFFFH             ;descriptor del segmento de datos
        DW      0
        DB      0
        DB      92H
        DB      8FH
        DB      0
IDTR    DIREC    <0FFH, IDT>       ;datos del IDTR
GDTR    DIREC    <17H, GDT>       ;datos del GDTR
JDIR    DIREC    <8, PM>          ;datos de salto (JMP) lejano
.CODE
.STARTUP
        MOV     AX, 0            ;inicializa DS
        MOV     DS, AX

        LIDT   IDTR             ;inicializa IDTR
        LGDT   GDTR             ;inicializa GDTR

        MOV     EAX, CRO          ;activa PE
        OR     EAX, 1
        MOV     CRO, EAX

        JMP     JDIR             ;salto lejano a PM
PM:::                           ;fuerza una etiqueta PM

;el software que sigue debe desarrollarse por separado
;con el conjunto de ensamblador para generar código en modo protegido de 32 bits
;
; por ejemplo: .MODEL FLAT

        MOV     AX, 10H            ;carga registros de segmento
        MOV     DS, AX             ;ahora en modo protegido
        MOV     ES, AX
        MOV     SS, AX
        MOV     FS, AX
        MOV     GS, AX
        MOV     SP, 0FFFFF000H

;aquí aparece la inicialización adicional

end

```

En sistemas más complejos (que aparecen muy poco en sistemas integrados), los pasos requeridos para inicializar el sistema en modo protegido son más elaborados. Para los sistemas complejos, que por lo general son sistemas multiusuario, los registros se cargan mediante el uso del segmento de estado de tareas (TSS). Los pasos necesarios para colocar el 80386 en la operación en modo protegido, para un sistema complejo que utilice un conmutador de tareas, son:

1. Inicialice la tabla de descriptores de interrupción, de manera que haga referencia a descriptores de interrupción válidos con por lo menos 32 descriptores en la IDT.

2. Inicialice la tabla de descriptores globales, de manera que contenga un descriptor de segmento de estado de tarea (TSS), junto con los segmentos iniciales de código y de datos requeridos para la tarea inicial.
3. Inicialice el registro de tareas (TR) de tal forma que apunte a un TSS.
4. Cambie al modo protegido mediante el uso de un salto entre segmentos (lejano) para vaciar la cola de instrucciones interna. Este proceso carga el TR con el selector del TSS y la tarea inicial actuales.
5. Ahora el 80386 opera en el modo protegido, bajo el control de la primera tarea.

El ejemplo 17-2 muestra el software requerido para inicializar el sistema y cambiar al modo protegido mediante el uso de un conmutador de tareas. La tarea inicial del sistema opera al nivel más alto de protección (00) y controla todo el entorno de operación completo para el 80386. En muchos casos se utiliza para arrancar (cargar) software que permite a muchos usuarios el acceso al sistema en un entorno multiusuario. Al igual que el ejemplo 17-1, este software no funcionará en una computadora personal, ya que se diseñó para funcionar sólo en un sistema integrado.

EJEMPLO 17-2

```
.MODEL SMALL
.386P
.DATA
DIREC    STRUC          ;estructura para la dirección de 48 bits
    DW   ?
    DD   ?
    ENDS
DESC     STRUC          ;estructura de un descriptor
    DW   ?
    DW   ?
    DB   ?
    DB   ?
    DB   ?
    DB   ?
    ENDS
TSS      STRUC          ;estructura del TSS
    DD   18 DUP(?)
    DD   18H           ;ES
    DD   10H           ;CS
    DD   4 DUP(18H)
    DD   28H           ;LDT
    DD   IOBP          ;mapa de privilegios de E/S
    ENDS
GDT      DESC  <>          ;nulo
        DESC <2067H, TS1, 0, 89H, 90H, 0> ;descriptor del TSS
        DESC <-1, 0, 0, 9AH, 0CFH, 0>  ;segmento de código
        DESC <-1, 0, 0, 9AH, 0CFH, 0>  ;segmento de datos
        DESC <0, 0, 0, 0, 0, 0>       ;LDT para el TSS
LDT      DESC  <>          ;nulo
IOBP    DB    2000H DUP(0)    ;toda la E/S activada
IDT     DQ    32 DUP(?)     ;IDT
TS1     TSS   <>          ;crea el TSS
IDTA    DIREC <0FFH, IDT>   ;IDTR
GDTA    DIREC <27H, GDT>    ;GDTR
JDIREC  DIREC <10H, PM>    ;dirección de salto
.CODE
.STARTUP
MOV     AX, 0
```

```

        MOV    DS, AX
        LGDT   GDTA
        LIDT   IDTA

        MOV    EAX, CR0
        OR     EAX, 1
        MOV    CR0, EAX

        MOV    AX, 8
        LTR    AX
        JMP    JDIREC

PM:
        ;modo protegido

END

```

Ninguno de los ejemplos 17-1 o 17-2 fueron escritos para funcionar en el entorno de la computadora personal. Este entorno requiere el uso del controlador VCPI (interfaz de programa de control virtual) que proporciona el controlador HIMEM.SYS, en el DOS, o del controlador DPMI (**interfaz DOS en modo protegido**) que proporciona Windows cuando se utiliza la interfaz del DOS. El ejemplo 17-3 indica cómo cambiar al modo protegido con el DPMI y después mostrar el contenido de cualquier área de memoria. Lo anterior incluye la memoria en el área de memoria extendida o en cualquier otro lado. Esta aplicación DOS deja ver en el monitor el contenido de cualquier posición de memoria en formato hexadecimal, incluyendo las posiciones que están por encima del primer 1 Mbyte del sistema de memoria.

EJEMPLO 17-3

```

;Un programa que muestra el contenido de cualquier área de memoria,
;incluyendo la memoria extendida.
;***sintaxis de la línea de comandos***
;EDUMP XXXX, YYYY en donde XXXX es la dirección inicial y YYYY es
;la dirección final.
;Nota: este programa debe ejecutarse desde Windows.
;

        .MODEL SMALL
        .386
        .STACK 1024           ;área de pila de 1,024 bytes
        .DATA

0000  00000000  ENTRADA DD    ?          ;punto de entrada DPMI
0004  00000000  SALIDA  DD    ?          ;punto de salida DPMI
0008  00000000  PRIMERA DD   ?          ;primera dirección
000C  00000000  ULTIMA1 DD   ?          ;última dirección
0010  0000      MTAM   DW    ?          ;tamaño de memoria para DPMI
0012  0D 0A 0A 50 61  ERR1   DB    13,10,10,'Parameter Error.$'
    72 61 6D 65 74
    65 72 20 65 72
    72 6F 72 2E 24
0026  0D 0A 0A 44 50  ERR2   DB    13,10,10,'DPMI not present.$'
    4D 49 20 6E 6F
    74 20 70 72 65
    73 65 6E 74 2E
    24
003B  0D 0A 0A 4E 6F  ERR3   DB    13,10,10,'Not enough real memory.$'
    74 20 65 6E 6F
    75 67 68 20 72
    65 61 6C 20 6D
    65 6D 6F 72 79
    2E 24
0056  0D 0A 0A 43 6F  ERR4   DB    13,10,10,'Could not move to protected mode.$'
    75 6C 64 20 6E
    6F 74 20 6D 6F
    76 65 20 74 6F
    20 70 72 6F 74
    65 63 74 65 64
    20 6D 6F 64 65
    2E 24

```

```

007B 0D 0A 0A 43 61 ERR5    DB      13,10,10,'Cannot allocate selector.$'
 6E 6E 6F 74 20
 61 6C 6C 6F 63
 61 74 65 20 73
 65 6C 65 63 74
 6F 72 2E 24
0098 0D 0A 0A 43 61 ERR6    DB      13,10,10,'Cannot use base address.$'
 6E 6E 6F 74 20
 75 73 65 20 62
 61 73 65 20 61
 64 64 72 65 73
 73 2E 24
00B4 0D 0A 0A 43 61 ERR7    DB      13,10,10,'Cannot allocate 64K to limit.$'
 6E 6E 6F 74 20
 61 6C 6C 6F 63
 61 74 65 20 36
 34 4B 20 74 6F
 20 6C 69 6D 69
 74 2E 24
00D5 0D 0A 24      CRLF   DB      13,10,'$'
00D8 50 72 65 73 73 MENS1   DB      'Press any key...$'
 20 61 6E 79 20
 6B 65 79 2E 2E
 2E 24
;
; registra almacenamiento de arreglo para función 0300H de DPMI
;
00E9 = 00E9      ARRAY   EQU     THIS BYTE
00E9 00000000  REDI    DD      0          ;EDI
00ED 00000000  RESI    DD      0          ;ESI
00F1 00000000  REBP    DD      0          ;EBP
00F5 00000000            DD      0          ;reservado
00F9 00000000  REBX    DD      0          ;EBX
00FD 00000000  REDX    DD      0          ;EDX
0101 00000000  RECX    DD      0          ;ECX
0105 00000000  REAX    DD      0          ;EAX
0109 0000  RFLAG   DW      0          ;banderas
010B 0000  RES     DW      0          ;ES
010D 0000  RDS     DW      0          ;DS
010F 0000  RFS     DW      0          ;FS
0111 0000  RGS     DW      0          ;GS
0113 0000  RIP     DW      0          ;IP
0115 0000  RCS     DW      0          ;CS
0117 0000  RSP     DW      0          ;SP
0119 0000  RSS     DW      0          ;SS
0000 .CODE
0000 .STARTUP
0010 8C C0      MOV     AX,ES
0012 8C DB      MOV     BX,DS      ;busca tamaño de programa y datos
0014 2B D8      SUB     BX,AX
0016 8B C4      MOV     AX,SP      ;busca tamaño de pila
0018 C1 E8 04    SHR     AX,4
001B 40          INC     AX
001C 03 D8      ADD     BX,AX      ;BX = longitud en párrafos
001E B4 4A      MOV     AH,4AH
0020 CD 21      INT     21H      ;modifica asignación de memoria
0022 E8 00D1    CALL    GETDA   ;obtiene información de línea de comandos
0025 73 0A      JNC     MAIN1   ;si los parámetros son buenos
0027 B4 09      MOV     AH,9       ;error en parámetros
0029 BA 0012 R   MOV     DX,OFFSET ERR1
002C CD 21      INT     21H
002E E9 00AA    JMP     MAINE   ;sale al DOS
0031           MAIN1:
0031 E8 00AB    CALL    ISDPMI  ;se cargó DPMI?
0034 72 0A      JC     MAIN2   ;si DPMI está presente
0036 B4 09      MOV     AH,9
0038 BA 0026 R   MOV     DX,OFFSET ERR2

```

```

003B CD 21           INT  21H      ;muestra que DPMI no está presente
003D E9 009B         JMP  MAINE   ;sale al DOS
0040
    MAIN2:
0040 B8 0000         MOV  AX,0     ;indica que se necesita 0 memoria
0043 83 3E 0010 R 00 CMP  MTAM,0  ;si DPMI no necesita memoria
0048 74 F6           JE   MAIN2   ;obtiene la cantidad
004A 8B 1E 0010 R    MOV  BX,MTAM
004E B4 48           MOV  AH,48H
0050 CD 21           INT  21H      ;asigna memoria para DPMI
0052 73 09           JNC  MAIN3   ;si no hay suficiente memoria real
0054 B4 09           MOV  AH,9     ;aplicación de 16 bits
0056 BA 003B R       MOV  DX,OFFSET ERR3
0059 CD 21           INT  21H
005B EB 7E           JMP  MAINE   ;sale al DOS
005D
    MAIN3:
005D 8E C0           MOV  ES,AX
005F B8 0000         MOV  AX,0     ;cambia al modo protegido
0062 FF 1E 0000 R    CALL DS:ENTRADA
0066 73 09           JNC  MAIN4
0068 B4 09           MOV  AH,9     ;si falló el cambio
006A BA 0056 R       MOV  DX,OFFSET ERR4
006D CD 21           INT  21H
006F EB 6A           JMP  MAINE   ;sale al DOS
;
;MODO PROTEGIDO
;
0071
    MAIN4:
0071 B8 0000         MOV  AX,0000H ;obtiene el selector local
0074 B9 001           MOV  CX,1     ;sólo se necesita uno
0077 CD 31           INT  31H
0079 72 48           JC   MAIN7   ;si hay error
007B 8B D8           MOV  BX,AX
007D 8E C0           MOV  ES,AX
007F B8 0007          MOV  AX,0007H ;establece dirección base
0082 8B 0E 000A R    MOV  CX,WORD PTR PRIMERA+2
0086 8B 16 0008 R    MOV  DX,WORD PTR PRIMERA
008A CD 31           INT  31H
008C 72 3D           JC   MAIN8   ;si hay error
008E B8 0008          MOV  AX,0008H
0091 B9 0000          MOV  CX,0
0094 BA FFFF          MOV  DX,0FFFFH ;establece límite en 64 K
0097 CD 31           INT  31H
0099 72 38           JC   MAIN9   ;si hay error
009B B9 0018          MOV  CX,24
009E BE 0000          MOV  SI,0   ;carga conteo de línea
00A1
    MAIN5:
00A1 E8 00F4           CALL MDIREC ;muestra dirección, si se necesita
00A4 E8 00CE           CALL MDATOS ;muestra datos
00A7 46               INC  SI      ;apunta a los siguientes datos
00A8 66| A1 0008 R    MOV  EAX,PRIMERA ;comprueba el final
00AC 66| 3B 06 000C R  CMP  EAX,ULTIMO1
00B1 74 07             JE   MAIN6   ;si terminó
00B3 66| FF 06 0008 R  INC  PRIMERA
00B8 EB E7             JMP  MAIN5
00BA
    MAIN6:
00BA B8 0001           MOV  AX,0001H ;libera el descriptor
00BD 8C C3             MOV  BX,ES
00BF CD 31             INT  31H
00C1 EB 18             JMP  MAINE   ;sale al DOS
00C3
    MAIN7:
00C3 BA 007B R         MOV  DX,OFFSET ERR5
00C6 E8 0096           CALL MUESTS ;muestra: no se puede asignar selector
00C9 EB 10             JMP  MAINE   ;sale al DOS
00CB
    MAIN8:
00CB BA 0098 R         MOV  DX,OFFSET ERR6
00CE E8 008E           CALL MUESTS ;muestra: no se puede usar direc base
00D1 EB E7             JMP  MAIN6   ;libera el descriptor

```

```

00D3          MAIN9:
00D3    BA 00B4 R      MOV     DX,OFFSET ERR7
00D6    E8 0086        CALL    MUESTS           ;muestra: no se puede asignar límite 64 K
00D9    EB DF          JMP    MAIN6           ;libera el descriptor
00DB          MAINE:
00DB          .EXIT
;
;El procedimiento ISDPMI comprueba la presencia de DPMI.
;***parámetros de salida***
;acarreo = 1; si DPMI está presente
;acarreo = 0; si DPMI no está presente
;
00DF          ISDPMI  PROC   NEAR
00DF    B8 1687        MOV     AX,1687H       ;obtiene estado de DPMI
00E2    CD 2F          INT     2FH            ;multiplexión de DOS
00E4    0B C0          OR      AX,AX
00E6    75 0D          JNZ    ISDPMI1        ;si no hay DPMI
00E8    89 36 0010 R    MOV     MSIZE,SI      ;almacena cantidad de memoria necesaria
00EC    89 3E 0000 R    MOV     WORD PTR ENTRADA,DI
00F0    8C 06 0002 R    MOV     WORD PTR ENTRADA+2,ES
00F4    F9              STC
00F5          ISDPMI1:
00F5          RET
00F6          ISDPMI  ENDP
;
;El procedimiento GETDA recupera los parámetros de la línea de
;comandos para mostrar memoria en hexadecimal.
;PRIMERA = la primera dirección de la línea de comandos
;ULTIMA = la última dirección de la línea de comandos
;***parámetros de retorno***
;acarreo = 1; si hay error
;acarreo = 0; si no hay error
;
00F6          GETDA   PROC   NEAR
00F6    1E              PUSH   DS
00F7    06              PUSH   ES
00F8    1F              POP    DS
00F9    07              POP    ES           ;intercambia ES con DS
00FA    BE 0081        MOV    SI,81H        ;direcciona línea de comandos
00FD          GETDA1:
00FD    AC              LODSB
00FE    3C 20          CMP    AL,' '
0100    74 FB          JE     GETDA1        ;si es espacio
0102    3C 0D          CMP    AL,13
0104    74 1E          JE     GETDA3        ;si introducción = error
0106    4E              DEC    SI           ;ajusta SI
0107          GETDA2:
0107    E8 0020        CALL   GETNU         ;obtiene primer número
010A    3C 2C          CMP    AL,','
010C    75 16          JNE    GETDA3        ;si no es coma = error
010E    66| 26: 89 16 0008 R  MOV    ES:PRIMERA,EDX
0114    E8 0013        CALL   GETNU         ;obtiene segundo número
0117    3C 0D          CMP    AL,13
0119    75 09          JNE    GETDA3        ;si es error
011B    66| 26: 89 16 000C R  MOV    ES:ULTIMA1,EDX
0121    F8              CLC
0122    EB 01          JMP    GETDA4        ;indica que no hay error
0124          GETDA3:
0124    F9              STC
0125          GETDA4:
0125    1E              PUSH   DS           ;intercambia ES con DS
0126    06              PUSH   ES
0127    1F              POP    DS
0128    07              POP    ES
0129    C3              RET

```

```

012A          GETDA    ENDP
;
; El procedimiento GETNU extrae un número de la línea de
; comandos y regresa con él en EDX y el último carácter de
; la línea de comandos en AL como un delimitador.
;

012A          GETNU    PROC    NEAR

012A  66| BA 00000000      MOV     EDX,0           ;borra el resultado
0130          GETNU1:
130          AC          LODSB   ;obtiene dígito de línea de comandos
; .IF AL >= 'a' && AL <= 'z'
0139  2C 20          SUB     AL,20H  ;convierte en mayúsculas
; .ENDIF
013B  2C 30          SUB     AL,'0'  ;convierte desde ASCII
013D  72 12          JB      GETNU2  ;si no es un número
; .IF     AL>9       ;convierte A-F desde ASCII
0143  2C 07          SUB     AL,7   ;.ENDIF
0145  3C 0F          CMP     AL,0FH
0147  77 08          JA      GETNU2  ;si no es 0-F
0149  66| C1 E2 04      SHL     EDX,4
014D  02 D0          ADD     DL,AL  ;agrega dígito a EDX
014E  EB DF          JMP     GETNU1  ;obtiene el siguiente dígito
0151          GETNU2:
0151  8A 44 FF          MOV     AL,[SI-1]  ;obtiene el delimitador
0154          C3          RET

0155          GETNU    ENDP
;
; El procedimiento MUESTC muestra el carácter ASCII encontrado
; en el registro AL.
; ***usa***
; INT21H
;
0155          MUESTC   PROC NEAR

0155  52          PUSH    DX
0156  8A D0          MOV     DL,AL
0158  B4 06          MOV     AH,6
015A  E8 0084          CALL    INT21H  ;hace INT 21H real
015D  5A          POP     DX
015E  C3          RET

015F          MUESTC   ENDP
;
; El procedimiento MUESTS muestra una cadena de caracteres
; desde el modo protegido direccionado por DS:EDX.
; ***usa***
; MUESTC
;
015F          MUESTS   PROC NEAR

015F  66| 81 E2 0000FFFF  AND     EDX,0FFFFH
0166  67& 8A 02          MOV     AL,[EDX]  ;obtiene carácter
0169  3C 24          CMP     AL,'$'  ;prueba si es el fin
016B  74 07          JE      DISP1  ;si es el fin
016D  66| 42          INC     EDX  ;direcciona siguiente carácter
016F  E8 FFE3          CALL    MUESTC  ;muestra el carácter
0172  EB EB          JMP     MUESTS  ;repite hasta $
0174          DISP1:
0174          C3          RET

0175          MUESTS   ENDP
;
; El procedimiento MDATOS muestra un byte de datos en la posición
; direccionada por ES:SI. El byte va seguido de un espacio.
; ***usa***
; DIP y MUESTC

```

```

;          MDATOS  PROC   NEAR
0175  26: 8A 04      MOV     AL,ES:[SI]      ;obtiene byte
0178  C0 E8 04      SHR     AL,4
017B  E8 000C      CALL    DIP
017E  26: 8A 04      MOV     AL,ES:[SI]      ;muestra el primer dígito
0181  E8 0006      CALL    DIP
0184  B0 20          MOV     AL,' '
0186  E8 FFCC      CALL    MUESTC
0189  C3            RET

018A           MDATOS  ENDP
;
;El procedimiento DIP muestra el cuarteto derecho en AL como
;un dígito hexadecimal.
;***usa***
;MUESTC
;
018A           DIP     PROC NEAR
;
018A  24 0F          AND     AL,0FH      ;obtiene el cuarteto derecho
018C  04 30          ADD     AL,30H      ;convierte a ASCII
;
0192  04 07          .IF AL > 39H    ;si A-F
;
0194  28 FFBF      CALL    MUESTC      ;muestra el dígito
0197  C3            RET

0198           DIP     ENDP
;
;El procedimiento MDIREC muestra la dirección hexadecimal que
;se encuentra en DS:PRIMERA si es un límite de párrafo.
;***usa***
;DIP, MUESTS, MUESTC e INT21H
;
0198           MDIREC  PROC NEAR
;
0198  66| A1 0008 R  MOV     EAX,PRIMERA  ;obtiene dirección
019C  A8 0F          TEST    AL,0FH      ;prueba para XXXXXXX0
019E  75 40          JNZ     MDIREC4    ;si no, no muestra dirección
01A0  BA 00D5 R      MOV     DX,OFFSET CRLF
01A3  E8 FFB9      CALL    MUESTS      ;muestra CR y LF
01A6  49             DEC     CX
01A7  75 18          JNZ     MDIREC2    ;si no es fin de página
01A9  BA 00D8 R      MOV     DX,OFFSET MENS1  ;si es fin de página
01AC  E8 FFB0      CALL    MUESTS      ;muestra: oprima cualquier tecla
01AF  MDIREC1:        MOV     AH,6
01AF  B4 06          MOV     DL,0FFH
01B1  B2 FF          CALL    INT21H    ;hace INT 21H real
01B3  E8 002B      CALL    MDIREC1    ;si no se escribió nada
01B6  74 F7          JZ      MDIREC1
01B8  BA 00D5 R      MOV     DX,OFFSET CRLF
01BB  E8 FFA1      CALL    MUESTS      ;muestra CRLF
01BE  B9 0018      MOV     CX,24
01C1  MDIREC2:        PUSH    CX
01C1  51             PUSH    CX
01C2  B9 0008      MOV     CX,8
01C5  66| 8B 16 0008 R  MOV     EDX,PRIMERA  ;almacena conteo de línea
01CA  MDIREC3:        MOV     EDX,CX
01CA  66| C1 C2 04  ROL     EDX,4
01CE  8A C2          MOV     AL,DL
01D0  E8 FFB7      CALL    DIP
01D3  E2 F5          LOOP   MDIREC3    ;muestra dígito
01D5  59             POP     CX
01D6  B0 3A          MOV     AL,':'
01D8  E8 FF7A      CALL    MUESTC      ;se repite 8 veces
01DB  B0 20          MOV     AL,' '
;
```

```

01DD E8 FF75          CALL    MUESTC      ;muestra un espacio
01E0                           MDIREC4:
01E0           C3          RET

01E1           MDIREC ENDP
;
;El procedimiento INT21H obtiene acceso a la instrucción
;INT 21H de DOS en modo real, con los parámetros intactos.
;
01E1           INT21H PROC NEAR

01E1   66| A3 0105 R     MOV     REAX,EAX      ;almacena los registros
01E5   66| 89 1E 00F9 R   MOV     REBX,EBX
01EA   66| 89 0E 0101 R   MOV     RECX,ECX
01EF   66| 89 16 00FD R   MOV     REDX,EDX
01F4   66| 89 36 00ED R   MOV     RESI,ESI
01F9   66| 89 3E 00E9 R   MOV     REDI,EDI
01FE   66| 89 2E 00F1 R   MOV     REBP,EBP
0203   9C              PUSHF
0204   58              POP    AX
0205   A3 109 R        MOV    RFLAG,AX      ;realiza interrupción de DOS
0208   06              PUSHH ES
0209   B8 0300          MOV    AX,0300H
020C   BB 0021          MOV    BX,21H
020F   B9 0000          MOV    CX,0
0212   1E              PUSHH DS
0213   07              POP    ES
0214   BF 00E9 R        MOV    DI,OFFSET ARRAY
0217   CD 31            INT    31H
0219   07              POP    ES
021A   A1 0109 R        MOV    AX,RFLAG      ;restaura los registros
021D   50              PUSHH AX
021E   9D              POPF
021F   66| 8B 3E 00E9 R   MOV    EDI,REDI
0224   66| 8B 36 00ED R   MOV    ESI,RESI
0229   66| 8B 2E 00F1 R   MOV    EBP,REBP
022E   66| A1 0105 R     MOV    EAX,REAX
0232   66| 8B 1E 00F9 R   MOV    EBX,REBX
0237   66| 8B 0E 0101 R   MOV    ECX,RECX
023C   66| 8B 16 00FD R   MOV    EDX,REDX
0241   C3              RET

0242           INT21H ENDP
END

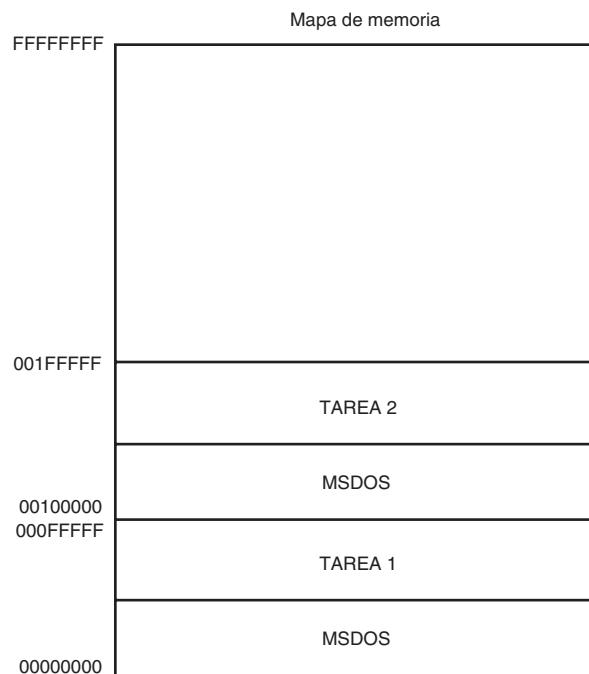
```

Tal vez haya observado que la llamada a la función INT 21H del DOS debe tratarse en forma distinta cuando opera en el modo protegido. El procedimiento que llama a INT 21H del DOS está al final del ejemplo 17-3. Como es en extremo extenso y consume mucho tiempo, optamos por tratar de no usar las interrupciones de DOS desde una aplicación Windows. La mejor manera de desarrollar software para Windows es con C/C++, con la inclusión de procedimientos en lenguaje ensamblador para tareas arduas.

17-5**MODO 8086 VIRTUAL**

Un modo especial de operación que no hemos visto hasta ahora es el 8086 virtual. Es especial porque se diseñó de manera que sea posible ejecutar varias aplicaciones de software en modo real a la vez. La PC opera así para las aplicaciones de DOS que utilizan el emulador de DOS cmd.exe (el símbolo del sistema). La figura 17-25 muestra dos aplicaciones para el 8086 asignadas en el 80386 mediante el uso del modo virtual. El sistema operativo permite la ejecución de varias aplicaciones, lo cual se realiza por lo general a través de una técnica llamada **partición de tiempo**. El sistema operativo asigna una cantidad fija de tiempo a cada tarea.

FIGURA 17-25 Dos tareas residentes en un 80386 que operan en el modo 8086 virtual.



Por ejemplo, si se están ejecutando tres tareas, el sistema operativo puede asignar 1 ms a cada una. Lo anterior significa que después de cada milisegundo se produce una conmutación de tareas para atender la siguiente tarea. De esta forma, todas las tareas reciben una porción del tiempo de ejecución del microprocesador, lo cual produce un sistema que parece ejecutar más de una tarea a la vez. Los tiempos llegan a ajustarse para otorgar a cualquier tarea un porcentaje del tiempo de ejecución del microprocesador.

Un sistema que logra utilizar dicha técnica es la cola de impresión. Ésta puede funcionar en una partición del DOS y utilizar el 10% del tiempo. Así, el sistema imprime mediante el uso de la cola de impresión, sin perder efectividad, ya que sólo utiliza el 10% de su tiempo.

La principal diferencia entre la operación del 80386 en modo protegido y el modo 8086 virtual es la manera en que el microprocesador interpreta los registros de segmento. En el modo 8086 virtual, los registros de segmento se utilizan de igual forma que en el modo real: como una dirección de segmento y una dirección de desplazamiento capaces de acceder a un espacio de memoria de 1 Mbyte de las posiciones 00000H-FFFFFH. El acceso a muchos sistemas en modo 8086 virtual es posible gracias a la unidad de paginación, cuya función explicaremos en la siguiente sección. Por medio de la paginación, el programa aún tiene acceso a la memoria que está por debajo del límite de 1 Mbyte, pero el microprocesador puede acceder a un espacio de memoria física en cualquier posición del rango de 4 Gbytes del sistema de memoria.

Para entrar al modo 8086 virtual se asigna un 1 lógico al bit VM en el registro EFLAG. Si el nivel de privilegios es 00, entraremos a dicho modo a través de una instrucción IRET. Este bit no puede activarse de ninguna otra manera. Un intento de acceder a una dirección de memoria por encima del límite de 1 Mbyte producirá una interrupción tipo 13.

El modo 8086 virtual puede usarse para compartir un microprocesador con muchos usuarios mediante el particionamiento de la memoria, de forma que cada usuario tenga su propia partición de DOS. Al usuario 1 es posible asignarle las posiciones de memoria 00100000H-01FFFFFH, al usuario 2, las posiciones 0020000H-01FFFFFFH y así sucesivamente. El software del sistema que se encuentra en las posiciones de memoria 00000000H-000FFFFH puede, entonces, compartir el microprocesador entre los usuarios, alternando de uno a otro para ejecutar software. De esta forma, varios usuarios comparten un solo microprocesador.

17-6**EL MECANISMO DE PAGINACIÓN DE MEMORIA**

Este mecanismo permite que cualquier dirección lineal (lógica), a medida que sea generada por un programa, se coloque en una página de memoria física, a medida que sea generada por el mecanismo de paginación. Una **página de memoria lineal** es una página que se direcciona con un selector y un desplazamiento, ya sea en el modo real o en el modo protegido. Una **página de memoria física** es una página que existe en cierta posición de memoria física real. Por ejemplo, la posición de memoria lineal 20000H se asignaría a la posición de memoria física 30000H o a cualquier otra posición, mediante la unidad de paginación. Esto significa que una instrucción que accede a la posición 20000H, en realidad está accediendo a la posición 30000H.

Cada página de memoria del 80386 es de 4 Kbytes. La paginación permite colocar el software del sistema en cualquier dirección física mediante el mecanismo de paginación. En la traducción de direcciones de página se utilizan tres componentes: el directorio de páginas, la tabla de páginas y la página de memoria física real. El administrador de memoria extendida EEM386.EXE utiliza el mecanismo de paginación para simular la memoria expandida en su memoria extendida y para generar bloques de memoria superior entre las ROMs del sistema.

El directorio de páginas

Este directorio contiene la posición de hasta 1024 tablas de traducción de páginas. Cada una de esas tablas traduce una dirección lógica en una dirección física. El directorio de páginas se almacena en la memoria y el acceso a este directorio se realiza mediante el registro de dirección de descriptor de página (CR₃) (vea la figura 17-14). El registro de control CR₃ almacena la dirección base del directorio de páginas, el cual comienza en cualquier límite de 4 Kbytes en el sistema de memoria. La instrucción MOV CR_{3,reg} se utiliza para inicializar CR₃ para la paginación. En un sistema en modo 8086 virtual, cada partición de DOS del 8086 tendría su propio directorio de páginas.

El directorio de páginas contiene hasta 1024 entradas, cada una de cuatro bytes. El directorio de página por sí solo ocupa una página de memoria de 4 Kbytes. Cada entrada en el directorio de páginas (vea la figura 17-26) traduce los 10 bits de más a la izquierda de la dirección de memoria. Esta porción de 10 bits de la dirección lineal se utiliza para localizar distintas tablas de páginas para distintas entradas en la tabla. La dirección de la tabla de páginas (A₃₂-A₁₂), que se almacena en una entrada del directorio de páginas, accede a una tabla de traducción de páginas de 4 Kbytes. Para traducir por completo cualquier dirección lineal en una dirección física se requieren 1024 tablas de páginas, cada una de 4 Kbytes, más el directorio de tablas de páginas, que también es de 4 Kbytes. Este esquema de traducción requiere hasta 4 Mbytes más 4 Kbytes de memoria para una traducción de dirección completa. Sólo los sistemas operativos más extensos soportan una traducción de direcciones de este tamaño. Muchos sistemas operativos comunes sólo traducen los primeros 16 Mbytes del sistema de memoria si la paginación está habilitada. Esto incluye programas tales como Windows. Esta traducción requiere cuatro entradas en el directorio de páginas (16 bytes) y cuatro tablas de páginas completas (16 Kbytes).

Cada uno de los bits de control de entradas en el directorio de tablas de páginas, como se muestra en la figura 17-26, realizan las siguientes funciones:

- D** El microprocesador 80386 no define el bit **malo** para las entradas en el directorio de tablas de páginas, está destinado para que lo use el sistema operativo.
- A** El bit **acceso** se establece en 1 lógico cada vez que el microprocesador accede a la entrada en el directorio de páginas.
- R/W y U/S** Los bits **lectura/escritura** y **usuario/supervisor** se utilizan en el esquema de protección, como se muestra en la tabla 17-2. Ambos bits se combinan para desarrollar la protección para el nivel 3 de prioridad de paginación, el nivel de usuario más bajo.

FIGURA 17-26 La entrada en el directorio de tablas de paginación.

31	12	11	10	9	8	7	6	5	4	3	2	1	0
Dirección de tabla de páginas (A ₃₁ -A ₁₂)	Reservados	0	0	D	A	0	0	U/S	R/W	P			

TABLA 17-2 Protección para el nivel 3 mediante el uso de U/S y R/W.

<i>U/S</i>	<i>R/W</i>	<i>Nivel de acceso 3</i>
0	0	Ninguna
0	1	Ninguna
1	0	Sólo lectura
1	1	Sólo escritura

- P** Si el bit **presente** es un 1 lógico, indica que la entrada es posible utilizarla en la traducción de direcciones. Si P = 0, la entrada no puede usarse para la traducción. Una entrada no presente llega a usarse para otros fines, por ejemplo para indicar que la página está almacenada en el disco. Si P = 0, el resto de los bits de la entrada se usarían para indicar la ubicación de la página en el sistema de memoria en disco.

La tabla de páginas

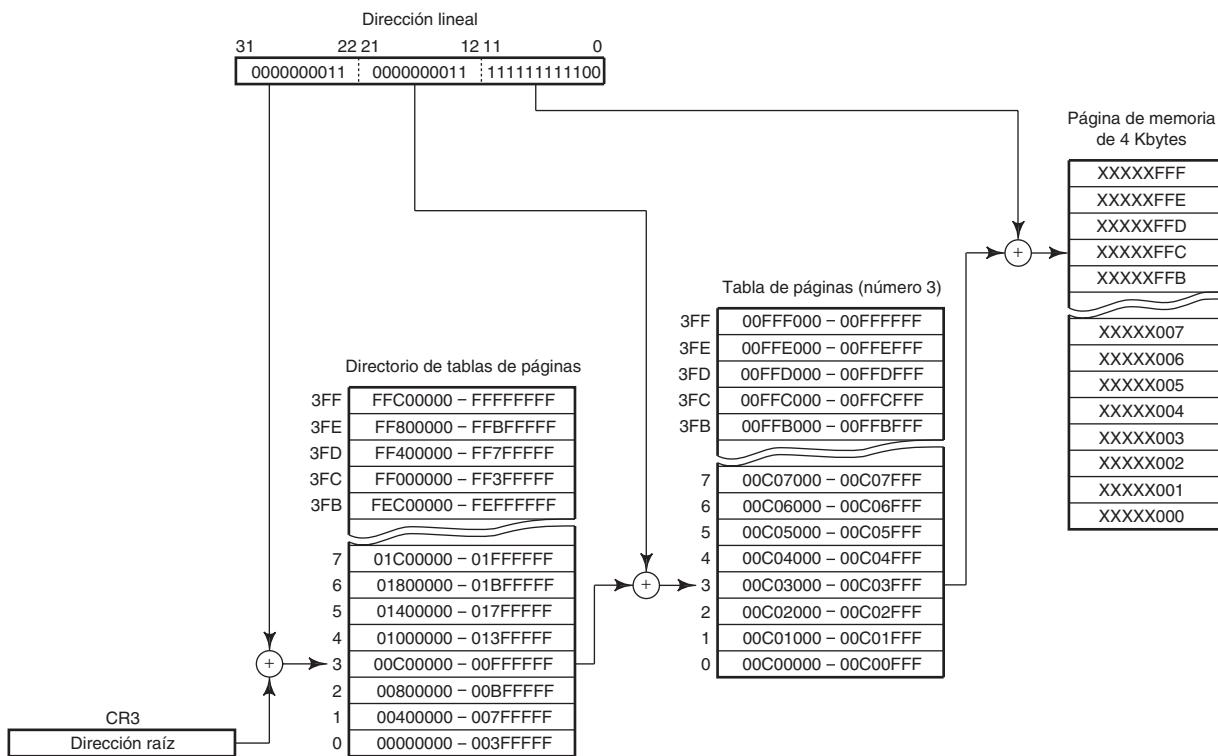
Esta tabla contiene 1024 direcciones de páginas físicas, a las cuales se accede para traducir una dirección lineal en una dirección física. Cada tabla de páginas traduce una sección de 4 Mbytes de la memoria lineal en 4 Mbytes de memoria física. El formato para la entrada de la tabla de páginas es el mismo que para la entrada del directorio de páginas (vea la figura 17-26). La principal diferencia es que la entrada del directorio de páginas contiene la dirección física de una tabla de páginas, mientras que la entrada de la tabla de páginas contiene la dirección física de una página física de memoria de 4 Kbytes. La otra diferencia es el bit D (malo), el cual no tiene función en la entrada del directorio de páginas, pero en una entrada de la tabla de páginas indica que se ha realizado una operación de escritura en una página.

La figura 17-27 muestra el mecanismo de paginación en el microprocesador 80386. Aquí, la dirección lineal 00C03FFCH, generada por un programa, se convierte en la dirección física XXXXXFFCH, según la traducción realizada por el mecanismo de paginación. (Nota: XXXXX es cualquier dirección de página física de 4 Kbytes.) El mecanismo de paginación funciona de la siguiente manera:

1. El directorio de páginas de 4 Kbytes se almacena como la dirección física localizada por CR₃. A esta dirección se le conoce en general como la **dirección raíz**. En un sistema hay un directorio de páginas a la vez. En el modo 8086 virtual, cada tarea tiene su propio directorio de páginas, con lo cual varias áreas distintas de memoria física pueden asignarse a distintas tareas en el 8086 virtual.
2. Los 10 bits superiores de la dirección lineal (bits 31 al 22), según la determinación de los descriptores que ya vimos en este capítulo, o con base en una dirección real, se aplican al mecanismo de paginación para seleccionar una entrada en el directorio de páginas. Con esta operación se asigna la entrada en el directorio de páginas a los 10 bits de más a la izquierda de la dirección lineal.
3. La entrada almacenada en el directorio de páginas direcciona la tabla de páginas. Esto permite hasta 4 K tablas de páginas en un sistema lleno y traducido por completo.
4. Los siguientes 10 bits de la dirección lineal (bits 21 al 12) direccionan una entrada en la tabla de páginas.
5. La entrada de la tabla de páginas contiene la dirección física actual de la página de memoria de 4 Kbytes.
6. Los 12 bits de más a la derecha de la dirección lineal (bits 11 al 0) seleccionan una posición en la página de memoria.

El mecanismo de paginación permite asignar la memoria física a cualquier dirección lineal a través del mecanismo de paginación. Por ejemplo, suponga que un programa selecciona la dirección lineal 20000000H, pero esta ubicación no existe en el sistema de memoria física. El programa hace referencia a la página lineal de 4 Kbytes como las posiciones 20000000H-20000FFFH. Como esta sección de memoria no existe, el sistema operativo asignaría una página de memoria física existente, tal como 12000000H-12000FFFH a este rango de direcciones lineales.

En el proceso de traducción de direcciones, los 10 bits de más a la izquierda de la dirección lineal seleccionan la entrada 200H del directorio de páginas que se ubica en la dirección de desplazamiento



Notas: 1. Los rangos de direcciones que se ilustran en el directorio de páginas y en la tabla de páginas representan los rangos de direcciones lineales y no el contenido de estas tablas.

2. Las direcciones (XXXXX) que se listan en la página de memoria se seleccionan con base en la entrada de la tabla de páginas.

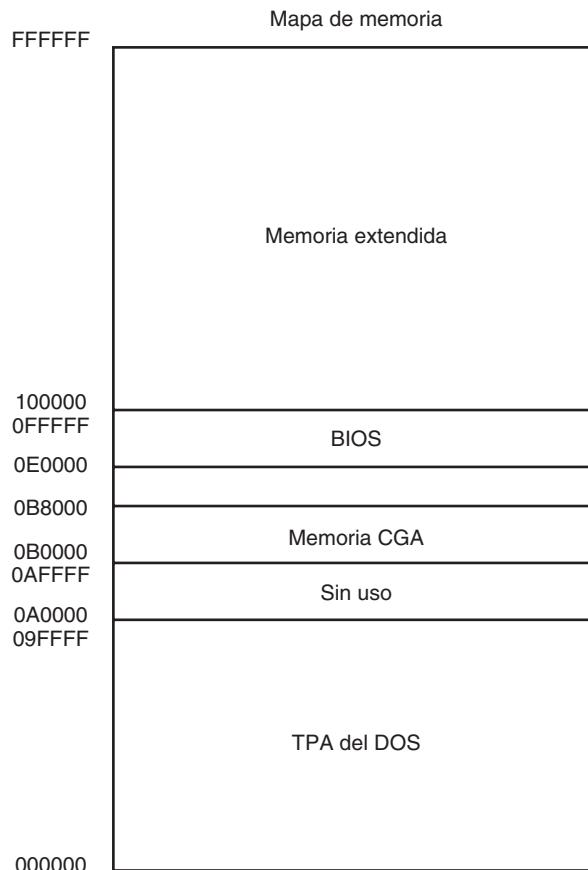
FIGURA 17-27 La traducción de la dirección lineal 00C03FFCH a la dirección de memoria física XXXXXFFCH. El valor de XXXXX se determina con base en la entrada de la tabla de páginas (que no se muestra aquí).

800H en el directorio de páginas. Esta entrada en el directorio de páginas contiene la dirección de la tabla de páginas para las direcciones lineales 20000000H-203FFFFFH. Los bits de dirección lineal (21-12) seleccionan una entrada en esta tabla de páginas, la cual corresponde a una página de memoria de 4 Kbytes. Para las direcciones lineales 20000000H-20000FFFH se selecciona la primera entrada (entrada 0) en la tabla de páginas. Esta primera entrada contiene la dirección física de la página de memoria actual, o 12000000H-12000FFFH en este ejemplo.

Vamos a tomar como ejemplo un sistema computacional típico basado en DOS. El mapa de memoria para el sistema aparece en la figura 17-28. En el mapa observamos que hay áreas de la memoria que no se utilizan, las cuales pueden paginarse hacia una ubicación distinta, con lo cual se proporciona más memoria a un programa de aplicación en modo real del DOS. El sistema de memoria normal del DOS comienza en la posición 00000H y se extiende hasta la posición 9FFFFH, para un total de 640 Kbytes de memoria. Por encima de la posición 9FFFFH encontramos secciones dedicadas a las tarjetas de video, las tarjetas de disco y la ROM del BIOS del sistema. En el ejemplo, una de las áreas de la memoria que está justo por encima de 9FFFFH se encuentra sin uso (A0000-AFFFFH). El DOS podría utilizar esta sección de la memoria, para que el área de memoria total para aplicaciones sea de 740 K en vez de 640 K. Tenga cuidado cuando maneje las posiciones A0000H-AFFFFH para RAM adicional, ya que la tarjeta de video utiliza dicha área para los gráficos de mapas de bits en los modos 12H y 13H.

Esta sección de memoria puede usarse si se asigna a la memoria extendida en las posiciones 1002000H-11FFFFH. En el ejemplo 17-4 se muestra el software para realizar esta traducción e inicializar el directorio de tablas de páginas, además de las tablas de páginas requeridas para establecer la

FIGURA 17-28 Mapa de memoria para un clon estilo AT.



memoria. Este procedimiento inicializa el directorio de tablas de páginas y una tabla de páginas; además carga CR₃. No cambia al modo protegido y habilita la paginación. Tome en cuenta que la paginación funciona en la operación con memoria en modo real.

EJEMPLO 17-4

```

.MODEL SMALL
.386P
.DATA

;directorio de páginas

PDIR DD 4

;tabla de páginas 0

TAB0 DD 1024 dup(?)
```

```

.CODE
.STARTUP
    MOV EAX, 0
    MOV AX, CS
    SHL EAX, 4
    ADD EAX, OFFSET TAB0
    AND EAX, 0FFFFF000H
    ADD EAX, 7
    MOV PDIR, EAX           ;direcciona el directorio de páginas
    MOV ECX, 256
    MOV EDI, OFFSET TAB0

```

```

MOV    AX, DS
MOV    ES, AX
MOV    EAX, 7
.REPEAT
    STOSD
    ADD   EAX, 4096
.UNTILCXZ
MOV    EAX, 102007H
MOV    ECX, 16
.REPEAT
    STOSD
    ADD   EAX, 4096
.UNTILCXZ
MOV    EAX, 0
MOV    AX, DS
SHL   EAX, 4
ADD   EAX, OFFSET PDIR      ;carga CR3
MOV    CR3, EAX

;software adicional para reasignar otras áreas de la memoria

END

```

17-7**INTRODUCCIÓN AL MICROPROCESADOR 80486**

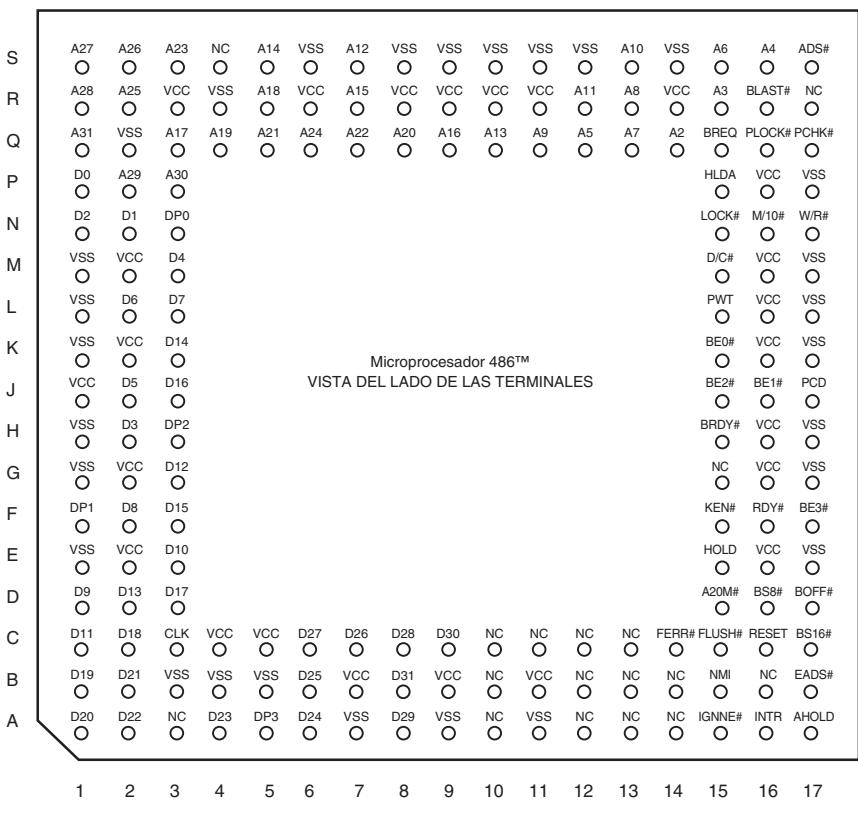
Este microprocesador es un dispositivo de alta integración, el cual contiene más de 1.2 millones de transistores. Dentro de este circuito de dispositivos hay una unidad de administración de memoria (MMU), un complejo coprocesador numérico compatible con el 80387, una memoria caché nivel 1 de alta velocidad que contiene 8 Kbytes de espacio y un microprocesador completo de 32 bits, compatible en forma ascendente con el microprocesador 80386. El 80486 está disponible en versiones de 25 MHz, 33 MHz, 50 MHz, 66 MHz o 100 MHz. Hay que tener en cuenta que la versión de 66 MHz opera al doble de la velocidad del reloj y que la versión de 100 MHz opera al triple. En 1990 Intel demostró el funcionamiento de una versión de 100 MHz (no la que opera al triple de velocidad de su reloj) del 80486 para la revista *Computer Design*, pero aún no ha salido al mercado. Advanced Micro Devices (AMD) produjo una versión de 40 MHz, que también está disponible en versiones de 80 MHz (al doble de velocidad de reloj) y 120 MHz (triple de velocidad de reloj). El 80486 está disponible como 80486DX o como 80486SX. La única diferencia entre estos dispositivos es que el 80486SX no contiene el coprocesador numérico, lo cual reduce su precio. El coprocesador numérico 80487SX está disponible por separado para el microprocesador 80486SX.

En esta sección explicaremos con detalle las diferencias entre los microprocesadores 80486 y 80386. Como veremos, tales diferencias son pocas. Las más notables se aplican al sistema de memoria caché y al generador de paridad.

Diagrama de terminales de los microprocesadores 80486SX y 80486DX

La figura 17-29 ilustra el diagrama de terminales del microprocesador 80486DX, en formato PGA de 168 terminales. No se muestra el 80486SX (que también viene empaquetado en un PGA de 168 terminales), ya que sólo hay unas cuantas diferencias. En el 80486DX la terminal B₁₅ es NMI y en el 80486SX la terminal A₁₅ es NMI. Las otras diferencias son que la terminal A₁₅ es IGNNE (ignora error numérico) en el 80486DX (no viene en el 80486SX); la terminal C₁₄ es FERR (error de punto flotante) en el 80486DX; y las terminales B₁₅ y C₁₄ en el 80386SX no están conectadas.

Al conectar el microprocesador 80486 se deben conectar todas las terminales V_{CC} y V_{SS} a la fuente de energía para su correcta operación. La fuente de energía tiene que ser capaz de suministrar 5.0 V ±10% con hasta 1.2 A de sobrecorriente para la versión de 33 MHz. La corriente de suministro promedio debe ser de 650 mA para la versión de 33 MHz. Intel también ha producido una versión de 3.3 V que requiere un promedio de 500 mA, a una triple velocidad de reloj de 100 MHz. Las salidas de 0 lógico permiten hasta 4.0 mA de corriente y las salidas de 1 lógico permiten hasta 1.0 mA. Si se requieren corrientes más grandes, como es común, entonces se deben usar búferes con el 80486. La figura



240440-2

FIGURA 17-29 Diagrama de terminales del 80486. (Cortesía de Intel Corporation.)

17-30 es de un sistema 80486DX con búferes. En el circuito que se muestra, sólo se utilizan búferes para las señales de dirección, de datos y paridad.

Definiciones de las terminales

- A₃₁-A₂** Las **salidas de dirección** A₃₁A₂ proporcionan la dirección a la memoria y la E/S durante la operación normal; durante una invalidación de línea de la caché, las terminales A₃₁-A₄ se utilizan para controlar el microprocesador.
- A20M** La **máscara del bit de dirección 20** hace que el 80486 envuelva su dirección de la posición 000FFFFFH a 00000000H, como el microprocesador 8086. Esto proporciona un sistema de memoria que funciona como el sistema de memoria real de 1 Mbyte del microprocesador 8086.
- ADS** El **estrobo de dirección de datos** se vuelve un 0 lógico para indicar que el bus de dirección contiene una dirección de memoria válida.
- AHOLD** La terminal **retención de dirección** hace que el microprocesador coloque sus conexiones del bus de direcciones en su estado de alta impedancia, mientras que el resto de los buses permanecen activos. Por lo general, otro maestro de bus utiliza dicha terminal para obtener el acceso para un ciclo de invalidación de la caché.
- BE3-BE0** Las salidas **habilitación de byte** seleccionan un banco del sistema de memoria cuando se transfiere información entre los microprocesadores y su memoria o su espacio de E/S. La señal **BE3** habilita las terminales D₃₁-D₂₄, **BE2** habilita D₂₃-D₁₆, **BE1** habilita D₁₅-D₈ y **BE0** habilita D₇-D₀.

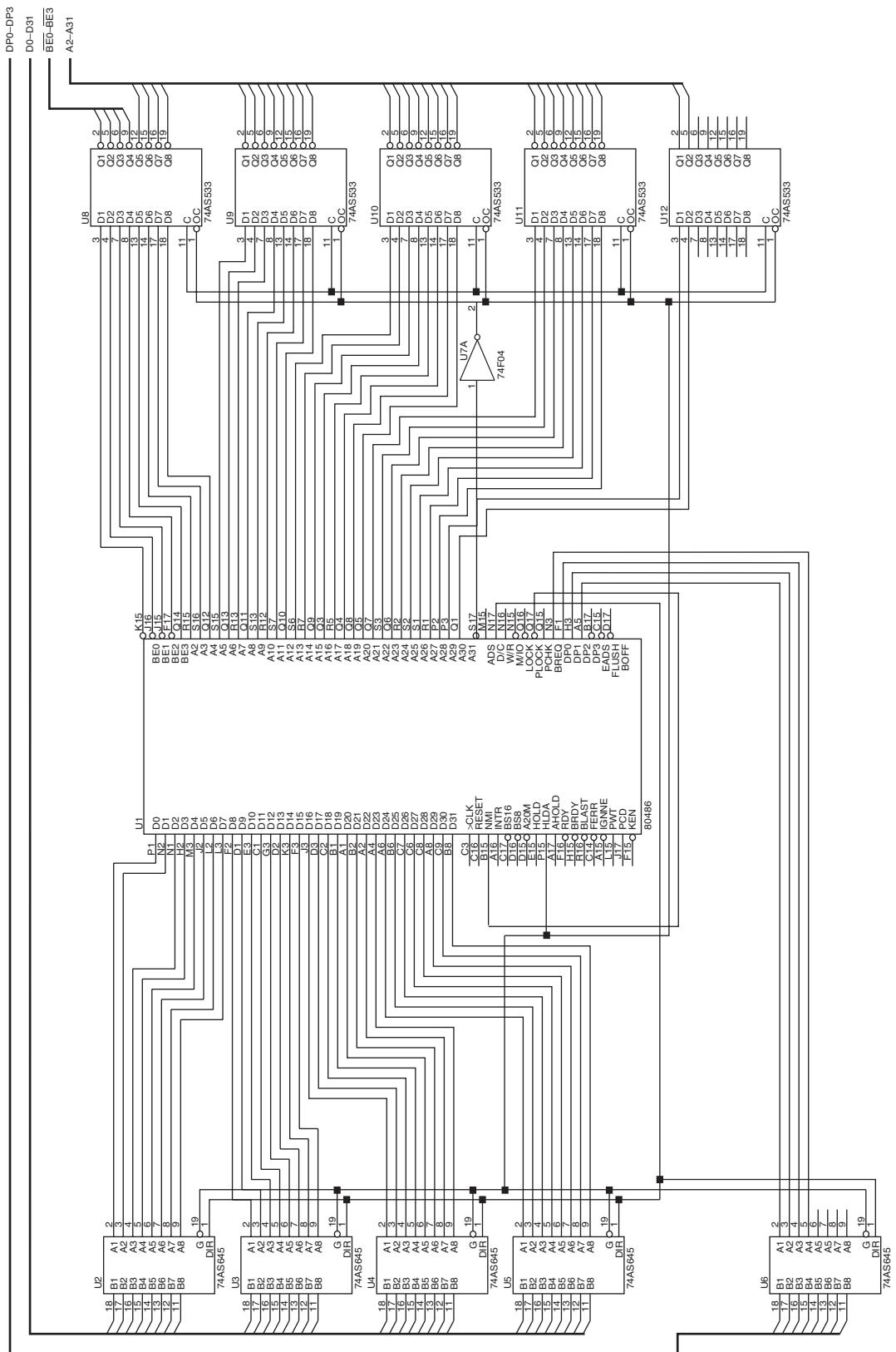


FIGURA 17-30 Un microprocesador 80486 que muestra el uso de búferes en los buses de direcciones, de datos y de paridad.

BLAST	La salida duración de ráfaga muestra que el ciclo de ráfaga del bus está completo en la siguiente activación de la señal $\overline{\text{BRDY}}$.
BOFF	La señal retiro hace que el microprocesador coloque sus buses en su estado de alta impedancia durante el siguiente ciclo de reloj. El microprocesador permanece en el estado de retención de bus hasta que la terminal $\overline{\text{BOFF}}$ se coloca en nivel de 1 lógico.
BRDY	La entrada ráfaga lista se utiliza para indicar al microprocesador que se ha completado un ciclo de ráfaga.
BREQ	La salida petición de bus indica que el 80486 ha generado una petición de bus interno.
BS8	La entrada bus tamaño 8 hace que el 80486 se estructure a sí mismo con un bus de datos de 8 bits para acceder a la memoria y a los componentes de E/S mediante bytes.
BS16	La entrada bus tamaño 16 hace que el 80486 se estructure a sí mismo con un bus de datos de 16 bits, para acceder a la memoria y a los componentes de E/S mediante palabras.
CLK	La entrada reloj proporciona al 80486 su señal de sincronización básica. La entrada de reloj es una entrada compatible con TTL de 25 MHz, para operar el 80486 a 25 MHz.
D₃₁-D₀	El bus de datos transfiere datos entre el microprocesador y sus sistemas de memoria y de E/S. Las conexiones D ₇ -D ₀ del bus de datos se utilizan también para aceptar el número de tipo de vector de interrupción durante un ciclo de reconocimiento de interrupción.
D/C	La salida datos/control indica si la operación actual es una transferencia de datos o un ciclo de control. En la tabla 17-3 consulte la función de D/C, M/IO y W/R.
DP₃-DP₀	Las terminales de E/S de paridad de datos proporcionan una paridad par para una operación de escritura y comprueban la paridad para una <u>operación de lectura</u> . Si se detecta un error de paridad durante una lectura, la salida $\overline{\text{PCHK}}$ se vuelve un 0 lógico para indicar un error de paridad. Si no se utiliza la paridad en un sistema, estas líneas deben llevarse al nivel alto, a +5.0 V o a 3.3 V en un sistema que utilice una fuente de 3.3 V.
EADS	La entrada estrobo de dirección externa se utiliza con AHOLD para indicar que se va a utilizar una dirección externa para realizar un ciclo de invalidación de caché.
FERR	La salida error de punto flotante indica que el coprocesador de punto flotante ha detectado una condición de error. Se utiliza para mantener la compatibilidad con el software para DOS.
FLUSH	La entrada vaciado de caché fuerza al microprocesador a borrar el contenido de su caché interna de 8 Kbytes.
HLDA	La salida reconocimiento de retención indica que la entrada HOLD está activa y que el microprocesador ha colocado sus buses en su estado de alta impedancia.

TABLA 17-3 Identificación del ciclo de bus.

<i>m/IO</i>	<i>D/C</i>	<i>W/R</i>	<i>Tipo de ciclo de bus</i>
0	0	0	Reconocimiento de interrupción
0	0	1	Alto/especial
0	1	0	Lectura de E/S
0	1	1	Escritura de E/S
1	0	0	Búsqueda de código de operación
1	0	1	Reservado
1	1	0	Lectura de memoria
1	1	1	Escritura de memoria

HOLD	La entrada retención solicita una acción de DMA. Hace que los buses de direcciones, de datos y de control se coloquen en su estado de alta impedancia y también, una vez que se reconoce, que HLDA se vuelva un 0 lógico.
IGNNE	La entrada ignora error numérico hace que el coprocesador ignore los errores de punto flotante y continúe procesando datos. Esta señal no afecta el estado de la terminal FERR.
INTR	La entrada p petición de interrupción solicita una interrupción enmascarable, de igual forma que en todos los demás miembros de la familia.
KEN	La entrada habilita caché hace que el bus actual se almacene en la caché interna.
LOCK	La salida bloqueo se vuelve un 0 lógico para cualquier instrucción en la que se utilice el prefijo de bloqueo.
M/IO	La terminal memoria/ES define si el bus de direcciones contiene una dirección de memoria o un número de puerto de E/S. También se combina con la señal W/R para generar señales de control de lectura y escritura de memoria y de E/S.
NMI	La entrada interrupción no enmascarable solicita una interrupción tipo 2.
PCD	La salida deshabilita caché de página refleja el estado del bit de atributo PCD en la entrada de la tabla de páginas o en la entrada del directorio de páginas.
PCHK	La salida comprobación de paridad indica que se detectó un error de paridad durante una operación de lectura en las terminales DP ₃ -DP ₀ .
PLOCK	La salida pseudo-bloqueo indica que la operación actual requiere más de un ciclo de bus para llevarse a cabo. Esta señal se vuelve un 0 lógico para las operaciones del coprocesador aritmético que accedan a datos en memoria de 64 o de 80 bits.
PWT	La salida paso de escritura de página indica el estado del bit de atributo PWT en la entrada de la tabla de páginas o en la entrada del directorio de páginas.
RDY	La entrada listo indica que se ha completado un ciclo de bus sin ráfaga. La señal RDY debe regresarse o, de lo contrario, el microprocesador colocará estados de espera en su sincronización hasta que se reconozca RDY .
RESET	La entrada reinicio inicializa el 80486, al igual que a los demás miembros de la familia. La tabla 17-4 muestra el efecto de la entrada RESET en el microprocesador 80486.
W/R	La terminal escritura/lectura indica si el ciclo de bus actual es una lectura o una escritura.

TABLA 17-4 Estado del microprocesador después de RESET.

Registro	Valor inicial con autoprueba	Valor inicial sin autoprueba
EAX	0000000H	?
EDX	00000400H + ID*	00000400H + ID*
EFLAGS	00000002H	00000002H
EIP	0000FFF0H	0000FFF0H
ES	0000H	0000H
CS	F000H	F000H
DS	0000H	0000H
SS	0000H	0000H
GS	0000H	0000H
FS	0000H	0000H
IDTR	Base = 0, límite = 3FFH	Base = 0, límite = 3FFH
CR0	60000010H	60000010H
DR7	0000000H	0000000H

*El número de ID de revisión lo suministra Intel para las revisiones al microprocesador.

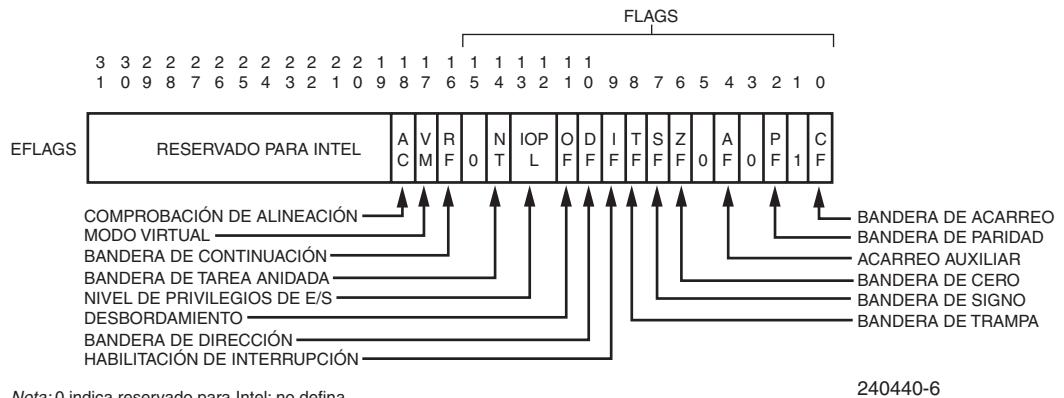


FIGURA 17-31 El registro EFLAGS del 80486. (Cortesía de Intel Corporation.)

Arquitectura básica del 80486

La arquitectura del 80486DX es casi idéntica a la del 80386. En el interior del 80486DX, además de la arquitectura del 80386 hay un coprocesador matemático y una memoria caché nivel 1 de 8 Kbytes. El 80486SX es casi idéntico a un 80386 con una caché de 8 Kbytes, pero sin coprocesador numérico.

En la figura 17-31 se muestra el registro de banderas extendido (EFLAGS). Al igual que con otros miembros de la familia, los bits de bandera de más a la derecha realizan las mismas funciones por cuestión de compatibilidad. El único bit de bandera nuevo es AC (**comprobación de alineación**), el cual se utiliza para indicar que el microprocesador ha accedido a una palabra en una dirección par, o a una doble palabra almacenada en un límite que no es de doble palabra. El software y la ejecución eficientes requieren que los datos se almacenen en límites de palabra o de doble palabra.

Sistema de memoria del 80486

El sistema de memoria para el 80486 es idéntico al del microprocesador 80386. El 80486 contiene 4 Gbytes de memoria, que comienzan en la posición 00000000H y terminan en la posición FFFFFFFFH. La principal modificación del sistema de memoria es interna para el 80486, en la forma de una memoria caché de 8 Kbytes, la cual agiliza la ejecución de las instrucciones y la adquisición de los datos. Otra adición es el comprobador/generador de paridad integrado en el microprocesador 80486.

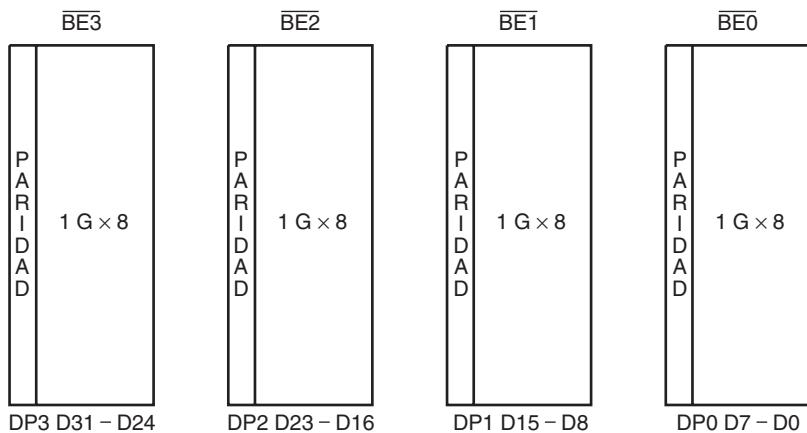
Comprobador/generador de paridad. A menudo se utiliza la paridad para determinar si los datos se leyeron correctamente de una posición de memoria. Para facilitar esta tarea, Intel ha incorporado un generador/detector de paridad interno. El 80486 genera la paridad durante cada ciclo de escritura. La paridad se genera como paridad par y se proporciona un bit de paridad para cada byte de memoria. Los bits de comprobación de paridad aparecen en las terminales DP₀-DP₃, que también son entradas de paridad además de salidas. Por lo general, se almacenan en memoria durante cada ciclo de escritura y se leen de memoria durante cada ciclo de lectura.

En una lectura, el microprocesador comprueba la paridad y genera un error de comprobación de paridad (en caso de que ocurra uno) en la terminal PCHK. Un error de paridad no produce ningún cambio en el procesamiento, a menos que el usuario aplique la señal PCHK a una entrada de interrupción. Por lo general, las interrupciones se utilizan para señalar un error de paridad en sistemas computacionales basados en DOS. La figura 17-32 muestra la organización de un sistema de memoria del 80486 que incluye el almacenamiento de la paridad. Esto es lo mismo que para el 80386, con excepción del almacenamiento del bit de paridad. Si no se utiliza la paridad, Intel recomienda que las terminales DP₀-DP₃ se lleven hasta +5.0 V.

Memoria caché. El sistema de memoria caché almacena los datos que utiliza un programa y también sus instrucciones. La caché está organizada como una caché asociativa de cuatro vías, en donde cada ubicación (línea) contiene 16 bytes o cuatro dobles palabras de datos. La caché opera como una caché de escritura directa. Observe que la caché cambia sólo cuando no hay ocurrencias. Esto significa que

FIGURA 17-32

La organización de la memoria del 80486; también se muestra la paridad.



los datos que se escriben en una posición de memoria que no estén almacenados ya en la caché no se escriben en la caché. En muchos casos, la mayor parte de la porción activa de un programa se encuentra dentro de la memoria caché. Lo anterior hace que la ejecución se realice a la velocidad de un ciclo de reloj para muchas de las instrucciones que se utilizan con frecuencia en un programa. Una forma (si no es que la única) de reducir la velocidad de estas eficientes instrucciones es cuando el microprocesador debe llenar una línea en la caché. Los datos también se almacenan en la caché, pero esto tiene un menor impacto en la velocidad de ejecución de un programa debido a que no se hace referencia a los datos con tanta frecuencia como a muchas porciones de un programa.

El registro de control 0 (CR_0) se utiliza para controlar la caché con dos nuevos bits de control que no están presentes en el microprocesador 80386. (En la figura 17-33 verá a CR_0 en el microprocesador 80486.) Los bits CD (deshabilita caché) y NW (sin caché de escritura directa) son nuevos para el 80486 y se utilizan para controlar la caché de 8 Kbytes. Si el bit CD es un 1 lógico, se inhiben todas las operaciones de la caché. Esta configuración se maneja sólo para el software de depuración y, por lo general, permanece en 0 lógico. El bit NW se utiliza para inhibir las operaciones de la caché de escritura directa. Al igual que con CD, la caché de escritura directa se inhibe sólo para fines de prueba. Para una operación normal de los programas, $CD = 0$ y $NW = 0$.

Debido a que la caché es nueva para el microprocesador 80486 y a que se llena mediante el uso de ciclos de ráfaga que no están en el 80386, se necesita cierto detalle para comprender los ciclos de llenado del bus. Cuando se llena una línea del bus, el 80486 debe adquirir cuatro números de 32 bits del sistema de memoria para llenar una línea en la caché. El llenado se lleva a cabo mediante un ciclo de ráfaga. Este ciclo es una memoria especial en la que se obtienen cuatro números de 32 bits del sistema de memoria en cinco periodos de reloj. Para esto, suponemos que la velocidad de la memoria es suficiente y que no se requieren estados de espera. Si la frecuencia de reloj del 80486 es de 33 MHz, llenamos una línea de la caché en 167 ns, lo cual es muy eficiente si consideramos que una operación de lectura de memoria de 32 bits normal, sin ráfaga, utiliza dos períodos de reloj.

Sincronización de lectura de memoria. La figura 17-34 muestra la sincronización de lectura del 80486 para una operación de memoria sin ráfaga. Observe que se utilizan dos períodos de reloj para transferir datos. El periodo de reloj T_1 proporciona la dirección de memoria y las señales de control, en tanto que el periodo de reloj T_2 está en donde se transfieren los datos entre la memoria y el microprocesador. Observe que \overline{RDY} debe volverse un 0 lógico para hacer que se transfieran datos y para terminar el ciclo de bus. Para determinar el tiempo requerido para un acceso sin ráfaga se toman dos períodos de reloj, menos el tiempo requerido para que la dirección aparezca en la conexión del bus de direcciones, menos un tiempo de preparación para las conexiones del bus de datos. Para la versión de 20 MHz del

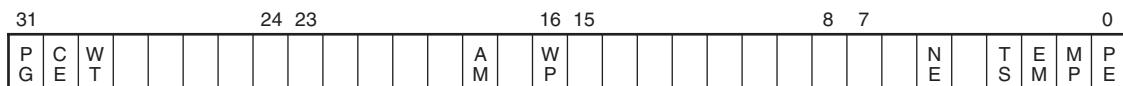
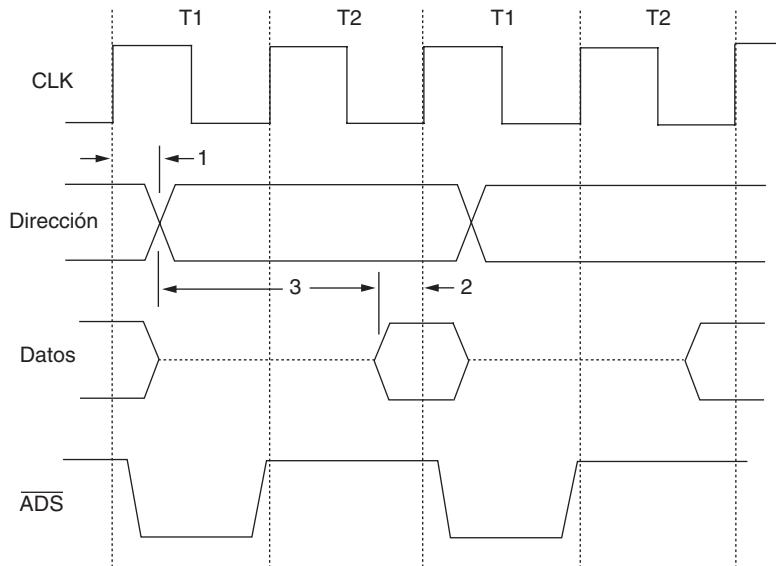
**FIGURA 17-33** El registro de control cero (CR_0) para el microprocesador 80486.

FIGURA 17-34

La sincronización de lectura sin ráfaga para el microprocesador 80486.

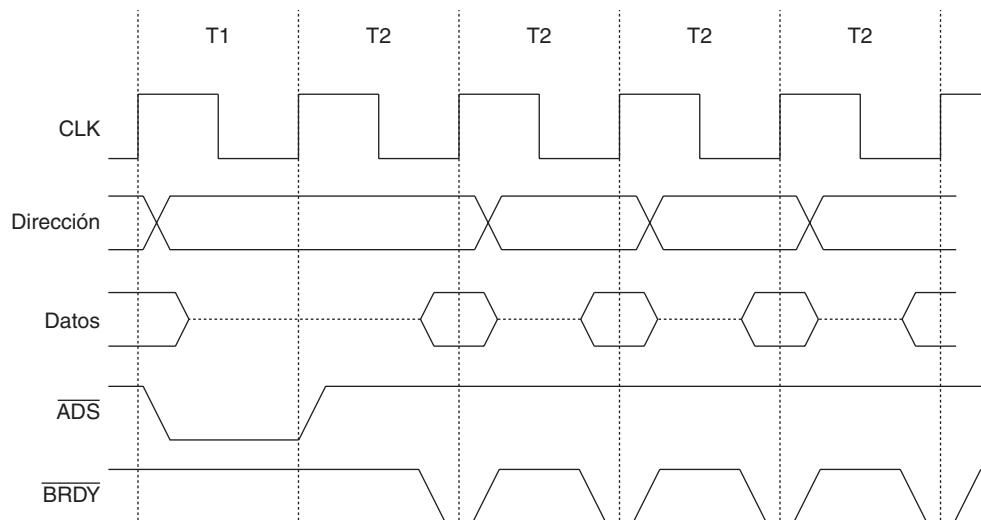


80486, dos períodos de reloj necesitan 100 ns menos 28 ns para el tiempo de preparación de la dirección y 6 ns para el tiempo de preparación de los datos. Esto da como resultado un tiempo de acceso sin ráfaga de 100 ns – 34 ns o 76 ns.

Desde luego que, si se incluyen los tiempos del decodificador y los retrasos, el tiempo de acceso permitido para la memoria es aún menor para una operación sin estados de espera. Si se utiliza una versión del 80486 de mayor frecuencia en un sistema, el tiempo de acceso a memoria es todavía menor.

Todos los procesadores 80486 de 33 MHz, 66 MHz y 100 MHz acceden a los datos del bus a una velocidad de 33 MHz. En otras palabras, el microprocesador puede operar a 100 MHz, pero el bus del sistema lo hace a 33 MHz. Observe que la sincronización de acceso sin ráfaga para el bus de sistema de 33 MHz permite 60 ns – 24 ns = 36 ns. Es obvio que se requieren estados de espera para operar con dispositivos de memoria DRAM estándar.

La figura 17-35 muestra el diagrama de sincronización para llenar una línea de caché con cuatro números de 32 bits mediante el uso de una ráfaga. Observe que las direcciones (A₃₁-A₄) aparecen durante T₁ y permanecen constantes durante todo el ciclo de ráfaga. Además, observe que A₂ y A₃ cambian durante cada T₁ después del primero para direccionar cuatro números consecutivos de 32 bits en el

**FIGURA 17-35** Un ciclo de ráfaga lee cuatro dobles palabras en cinco períodos de reloj.

sistema de memoria. Como se mencionó, en el proceso de llenar la caché mediante ráfagas se requieren sólo cinco periodos de reloj (un T_1 y cuatro T_2) para llenar una línea de caché con cuatro dobles palabras de datos. Si se utiliza una versión de 20 MHz del 80486, el tiempo de acceso para la segunda doble palabra y todas las dobles palabras subsiguientes es de 50 ns – 28 ns – 5 ns o 17 ns, suponiendo que no hay retrasos en el sistema. Para usar las transferencias en modo ráfaga necesitamos memoria de alta velocidad. Como los tiempos de acceso de la memoria DRAM son de 40 ns cuando mucho, nos vemos forzados a utilizar SRAM para las transferencias por ciclo de ráfaga. El sistema de 33 MHz permite un tiempo de acceso de 30 ns – 19 ns – 5 ns o 6 ns para el segundo byte y todos los bytes subsiguientes. Si se utiliza un contador externo en vez de los bits de dirección A_2 y A_3 podrán eliminarse los 19 ns y el tiempo de acceso será de 30 ns – 5 ns o 25 ns, lo cual es suficiente tiempo incluso hasta para la SRAM más lenta que se conecte al sistema como caché. A este circuito se le conoce por lo general como *caché sincrónica en modo ráfaga* si se utiliza una caché de SRAM con el sistema. La terminal \overline{BRDY} se usa para reconocer una transferencia de bus en vez de la terminal \overline{RDY} , que reconoce una transferencia normal de memoria.

El PWT controla la manera en que funciona la caché para una operación de escritura de la memoria caché externa; no controla la manera de escribir en la caché interna. El nivel lógico de este bit se encuentra en la terminal PWT del microprocesador 80486. Puede usarse de manera externa para dictar la política de escritura directa de la caché externa.

El bit PCD controla la caché dentro del chip. Si $PCD = 0$, la caché dentro del chip se habilita para la página actual de memoria. Tenga en cuenta que las entradas en la tabla de páginas del 80386 colocan un 0 lógico en la posición de bit PCD, con lo cual se habilita el uso de la caché. Si $PCD = 1$, se deshabilita la caché dentro del chip. El uso de la caché se deshabilita sin importar la condición de \overline{KEN} , CD y NW.

17-8

RESUMEN

1. El microprocesador 80386 es una versión mejorada del microprocesador 80286; incluye una unidad de administración de memoria mejorada para proporcionar la paginación de la memoria. El 80386 también trae registros extendidos de 32 bits, así como un bus de direcciones y datos de 32 bits. El microprocesador 80386SX es una versión reducida del 80386DX, el cual cuenta con un bus de datos de 16 bits y un bus de direcciones de 24 bits. El 80386EX es una computadora personal estilo AT completa en un chip.
2. El 80386 tiene un tamaño de memoria física de 4 Gbytes, los cuales pueden direccionarse como memoria virtual de hasta 64 Tbytes. La memoria del 80386 es de 32 bits y se direcciona en forma de bytes, palabras o dobles palabras.
3. Cuando el 80386 se opera en el modo de canalización, envía la dirección de la siguiente instrucción o de los datos en memoria al sistema de memoria, antes de completar la ejecución de la instrucción actual. Esto permite que el sistema de memoria empiece a buscar la siguiente instrucción o datos antes de que la actual se complete. De esta forma se incrementa el tiempo de acceso y, por consecuencia, se reduce la velocidad de la memoria.
4. Un sistema de memoria caché permite acceder a los datos que se leen con frecuencia en menos tiempo, ya que se almacenan en memoria de semiconductor de alta velocidad. Si se escriben datos en la memoria también se escriben en la caché, por lo que los datos más actuales siempre están en la caché.
5. La estructura de E/S del 80386 es casi idéntica al 80286, sólo que la E/S puede inhibirse cuando el 80386 se opera en el modo protegido a través del mapa de protección de bits de E/S que se almacena con el TSS.
6. En el microprocesador 80386 se expandieron las interrupciones para incluir interrupciones predefinidas adicionales en la tabla de vectores de interrupción. Estas interrupciones adicionales se utilizan con el sistema de administración de memoria.
7. El administrador de memoria del 80386 es similar al 80286, sólo que las direcciones físicas que se generan mediante la MMU son de 32 bits, en vez de 24 bits. La MMU del 80386 es capaz de paginar.

8. El 80386 se opera en el modo real (modo 8086) cuando se reinicia. El modo real permite al microprocesador direccionar datos en el primer 1 Mbyte de memoria. En el modo protegido, el 80386 direcciona cualquier ubicación en sus 4 Gbytes de espacio de direcciones físicas.
9. Un descriptor es una serie de ocho bytes que especifican la manera en que el 80386 debe utilizar un segmento de código o de datos. El descriptor se selecciona mediante un selector que se almacena en uno de los registros de segmento. Los descriptores se utilizan sólo en el modo protegido.
10. La administración de memoria se lleva a cabo mediante una serie de descriptores, los cuales se almacenan en tablas de descriptores. Para facilitar la administración de memoria, el 80386 utiliza tres tablas de descriptores: la tabla de descriptores globales (GDT), la tabla de descriptores locales (LDT) y la tabla de descriptores de interrupción (IDT). La GDT y la LDT almacenan hasta 8192 descriptores cada una; la IDT almacena hasta 256 descriptores. La GDT y la LDT describen tanto segmentos de código y de datos como tareas. La IDT describe los 256 niveles de interrupción distintos, mediante el uso de descriptores de compuertas de interrupción.
11. El TSS (segmento de estado de tareas) contiene información sobre la tarea actual y la anterior. Al final del TSS se adjunta un mapa de protección de bits de E/S, el cual inhibe las direcciones de puertos de E/S seleccionadas.
12. El mecanismo de paginación de memoria permite asignar cualquier página de memoria física a cualquier página de memoria lineal de 4 Kbytes. Por ejemplo, a la posición de memoria 00A00000H se le puede asignar la posición de memoria A0000000H mediante el mecanismo de paginación. Para asignar cualquier dirección física a cualquier dirección lineal se utilizan un directorio de páginas y tablas de páginas. El mecanismo de paginación puede utilizarse en el modo protegido o en el modo virtual.
13. El microprocesador 80486 es una versión mejorada del microprocesador 80386; contiene una caché de 8 bytes y un coprocesador aritmético 80387; ejecuta muchas instrucciones en un solo periodo de reloj.
14. El microprocesador 80486 ejecuta unas cuantas nuevas instrucciones que controlan la memoria caché interna; estas instrucciones permiten la suma (XADD) y la comparación (CMPXCHG) con un intercambio, además de una operación de intercambio de bytes (BSW AP). Aparte de estas instrucciones adicionales, el 80486 es 100% compatible en forma ascendente con el 80386 y el 80387.
15. El 80486 incluye una nueva característica conocida como BIST (autoprueba integrada), cuya función es evaluar el microprocesador, el coprocesador y la caché al momento de su reinicio. Si el 80486 pasa la prueba, EAX contiene un cero.

17-9**PREGUNTAS Y PROBLEMAS**

1. El microprocesador 80386 direcciona _____ bytes de memoria en el modo protegido.
2. El microprocesador 80386 direcciona _____ bytes de memoria virtual mediante la unidad de administración de memoria.
3. Describa las diferencias entre el 80386DX y el 80386SX.
4. Dibuje el mapa de memoria del 80386 cuando se opera en el (a) modo protegido; (b) modo real.
5. ¿Cuánta corriente hay disponible en las diversas conexiones de las terminales de salida del 80386? Compare estas corrientes con las corrientes disponibles en la conexión de terminal de salida de un microprocesador 8086.
6. Describa el sistema de memoria del 80386; explique el propósito y la operación de las señales de selección de banco.
7. Explique la acción de un reinicio de hardware en las conexiones del bus de direcciones del 80386.
8. Explique cómo la canalización extiende el tiempo de acceso para muchas referencias a memoria en el sistema basado en el microprocesador 80386.
9. Describa en forma breve cómo funciona el sistema de memoria caché.
10. En el 80386, los puertos de E/S comienzan en la dirección de E/S _____ y se extienden hasta la dirección de E/S _____.
11. ¿Qué puertos de E/S comunican datos entre el 80386 y su coprocesador 80387 complementario?

12. Compare y contraste las conexiones de memoria y de E/S que se encuentran en el 80386, con las de los microprocesadores anteriores.
13. Si el 80386 opera a 20 MHz, ¿qué frecuencia de reloj se aplica a la terminal CLK₂?
14. ¿Cuál es el propósito de la terminal BS16 en el microprocesador 80386?
15. Defina el propósito de cada uno de los registros de control (CR₀, CR₁, CR₂ y CR₃) que se encuentran dentro del 80386.
16. Defina el propósito de cada uno de los registros de depuración del 80386.
17. ¿Qué nivel de depuración producen los registros de depuración?
18. ¿Cuáles son los registros de prueba?
19. Seleccione una instrucción que copie el registro de control 0 en EAX.
20. Describa el propósito de PE en CR₀.
21. Forme una instrucción que acceda a los datos en el segmento FS, en la posición direccionada de manera indirecta por el registro DI. La instrucción deberá almacenar el contenido de EAX en esta posición de memoria.
22. ¿Qué es el direccionamiento de índice escalado?
23. ¿Es legal la siguiente instrucción: MOV AX,[EBX+ECX]?
24. Explique cómo las siguientes instrucciones calculan la dirección de memoria:
 - a) ADD [EBX+8*ECX],AL
 - b) MOV DATA[EAX+EBX],CX
 - c) SUB EAX,DATOS
 - d) MOV ECX,[EBX]
25. ¿Cuál es el propósito del tipo de interrupción número 7?
26. ¿Qué número de tipo de vector de interrupción se activa para una violación de privilegios de protección?
27. ¿Qué es un doble fallo de interrupción?
28. Si ocurre una interrupción en el modo protegido, ¿qué es lo que define a los vectores de interrupción?
29. ¿Qué es un descriptor?
30. ¿Qué es un selector?
31. ¿Cómo le hace el selector para seleccionar la tabla de descriptores locales?
32. ¿Qué registro se utiliza para direccionar la tabla de descriptores globales?
33. ¿Cuántos descriptores globales pueden almacenarse en la GDT?
34. Explique cómo el 80386 puede direccionar un espacio de memoria virtual de 64 Tbytes, cuando la memoria física sólo contiene 4 Gbytes de memoria.
35. ¿Cuál es la diferencia entre un descriptor de segmento y un descriptor de sistema?
36. ¿Qué es el segmento de estado de tareas (TSS)?
37. ¿Cómo se direcciona el TSS?
38. Describa cómo el 80386 cambia del modo real al modo protegido.
39. Describa cómo el 80386 cambia del modo protegido al modo real.
40. ¿En qué consiste la operación en modo 8086 virtual del microprocesador 808386?
41. ¿Cómo localiza el 80386 el directorio de páginas?
42. ¿Cuántos bytes hay en una página de memoria?
43. Explique cómo puede asignarse la dirección de memoria lineal D0000000H a la dirección de memoria física C0000000H mediante el uso de la unidad de paginación del 80386.
44. ¿Cuáles son las diferencias entre un microprocesador 80386 y un 80486?
45. ¿Cuál es el propósito de la terminal de entrada FLUSH en el microprocesador 80486?
46. Compare el conjunto de registros del 80386 con el microprocesador 80486.
47. ¿Qué diferencias hay en las banderas del 80486, si se le compara con el microprocesador 80386?
48. ¿Qué terminales se utilizan para la comprobación de la paridad en el microprocesador 80486?
49. El microprocesador 80486 utiliza la paridad _____.
50. La caché dentro del microprocesador 80486 es de _____ Kbytes.
51. Para llenar una línea de caché, se leen _____ bytes del sistema de memoria.
52. ¿Qué es una ráfaga del 80486?
53. Defina el término *caché de escritura directa*.
54. ¿Qué significa BIST?

CAPÍTULO 18

Los microprocesadores Pentium y Pentium Pro

INTRODUCCIÓN

El microprocesador Pentium representa una mejora a la arquitectura del microprocesador 80486. Las modificaciones incluyen una estructura de la caché mejorada, un bus de datos más amplio, un coprocesador numérico más veloz, un procesador de enteros dual y lógica de predicción de bifurcación. La caché se reorganizó para formar dos cachés, cada una de las cuales tiene 8 Kbytes. Una caché es para los datos y la otra para las instrucciones. La anchura del bus de datos se incrementó de 32 a 64 bits. El coprocesador numérico opera casi cinco veces más rápido que el coprocesador numérico 80486. A menudo, un procesador de enteros dual permite dos instrucciones por ciclo. Por último, la lógica de predicción de bifurcación permite que los programas con bifurcaciones se ejecuten con mayor eficiencia. Tenga en cuenta que tales modificaciones son internas para el Pentium, lo cual hace que el software para los microprocesadores Intel 80X86 anteriores sea compatible en forma ascendente. Poco después de que el Pentium salió al mercado se agregaron las instrucciones MMX.

El Pentium Pro es una versión aún más rápida del Pentium. Contiene una arquitectura interna modificada, que puede programar hasta cinco instrucciones para su ejecución y una unidad de punto flotante mucho más veloz. El Pentium Pro también contiene una caché de 256 Kbytes o de 512 Kbytes de nivel 2, además de la caché de 16 Kbytes (8 K para datos y 8 K para instrucciones) de nivel 1. Este microprocesador también incluye circuitos de corrección de errores (ECC) para corregir un error de un bit y señalar un error de dos bits. También se agregaron cuatro líneas de dirección adicionales para que el Pentium Pro tuviera acceso a la sorprendente cantidad de 64 Gbytes de espacio de memoria direccional en forma directa.

OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

1. Comparar el Pentium y el Pentium Pro con los microprocesadores 80386 y 80486.
2. Describir la organización y la interfaz del sistema de memoria de 64 bits del Pentium y sus variaciones.
3. Comparar las modificaciones en la unidad de administración de memoria y en la unidad de paginación con los microprocesadores 80386 y 80486.
4. Detallar el funcionamiento de las nuevas instrucciones que se incluyen en el microprocesador Pentium.
5. Explicar cómo las unidades enteras duales superescalares mejoran el rendimiento del microprocesador Pentium.
6. Describir la operación de la lógica de predicción de bifurcación.
7. Explicar con detalle las mejoras en el Pentium Pro, en comparación con el Pentium.
8. Explicar cómo funciona la arquitectura de ejecución dinámica del Pentium Pro.

18-1

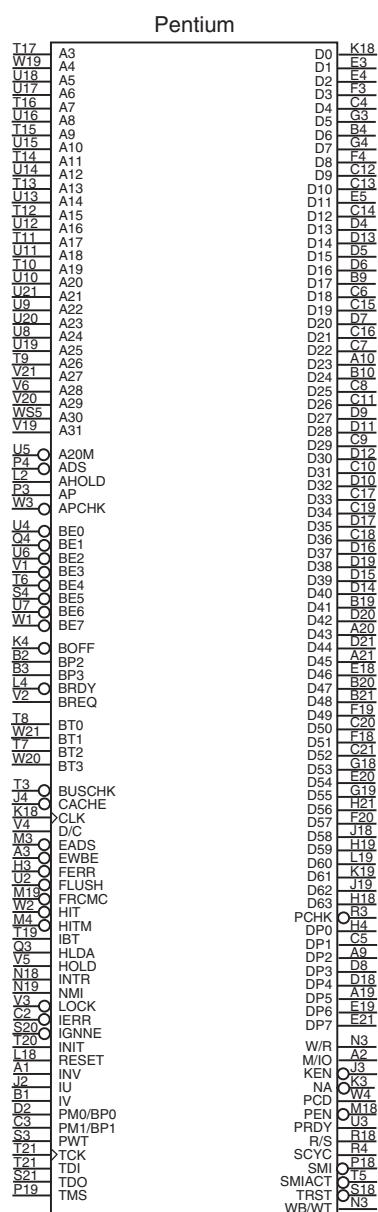
INTRODUCCIÓN AL MICROPROCESADOR PENTIUM

Antes de utilizar el microprocesador Pentium o cualquier otro en un sistema, se debe comprender el funcionamiento de cada una de las terminales. En esta sección veremos con detalle la operación de cada terminal, junto con las estructuras del sistema de memoria externa y de E/S del microprocesador Pentium.

La figura 18-1 muestra el diagrama de terminales del microprocesador Pentium, que viene encapsulado en un enorme PGA (matriz de rejillas de terminales) de 237 terminales. El Pentium está disponible en dos versiones: el Pentium verdadero y la versión P24T, llamada Pentium OverDrive. Esta última versión contiene un bus de datos de 32 bits, compatible para insertarse en equipos 80486 que contengan el zócalo (socket) para el P24T.

La versión P24T también viene con un ventilador integrado a la unidad. La diferencia más notable en el diagrama de terminales del Pentium, cuando se le compara con los primeros microprocesadores

FIGURA 18-1 Diagrama de terminales del microprocesador Pentium.



80486, es que hay 64 conexiones en el bus de datos en lugar de 32, lo cual requiere una huella física más grande.

Al igual que con las versiones anteriores de la familia de microprocesadores Intel, las primeras versiones del Pentium requieren una sola fuente de energía de +5.0 V para operar. El promedio de corriente de la fuente de energía es de 3.3 A para la versión de 66 MHz del Pentium y de 2.91 A para la versión de 60 MHz. Como estas corrientes son considerables, también lo son las disipaciones de potencia de dichos microprocesadores: 13 W para la versión de 66 MHz y 11.9 W para la versión de 60 MHz. Las versiones actuales del Pentium, de 90 MHz y superiores, utilizan una fuente de energía de 3.3 V con un consumo de corriente reducido. En la actualidad se requiere un buen disipador de calor con un flujo de aire considerable para mantener el Pentium fresco. Este microprocesador contiene varias conexiones V_{CC} y V_{SS} que deben conectarse a +5.0 V o a +3.3 V y a tierra para su correcta operación. Algunas de las terminales se identifican como N/C (sin conexión) y no deben conectarse. Las versiones más recientes del Pentium se han mejorado para reducir la disipación de potencia. Por ejemplo, el Pentium de 233 MHz requiere 3.4 A de corriente, una cantidad un poco mayor a los 3.3 A que requiere la primera versión de 66 MHz.

Cada una de las terminales de salida del Pentium es capaz de suministrar 4.0 mA de corriente en el nivel de 0 lógico y 2.0 mA en el nivel de 1 lógico. Esto representa un incremento en la corriente de control, en comparación con los 2.0 mA disponibles en las terminales de salida de los primeros microprocesadores 8086, 8088 y 80286. Cada terminal de entrada representa una pequeña carga que requiere sólo 15 µA de corriente. En algunos sistemas (excepto en los más pequeños), estos niveles de corriente requieren búferes de bus.

A continuación se describe la función de cada grupo de terminales del Pentium:

A20	La máscara de dirección A20 es una entrada que se impone en el modo real para indicar al Pentium que debe realizar un envolvimiento de direcciones, como en el microprocesador 8086, para el uso del controlador HIMEM.SYS.
A₃₁-A₃	Las conexiones del bus de direcciones direccionan cualquiera de las 512 K × 64 posiciones del sistema de memoria del Pentium. Observe que A ₀ , A ₁ y A ₂ se codifican en la habilitación del bus (BE₇-BE₀), que se describe más abajo, para seleccionar cualquiera de los ocho bytes (o todos) en una posición de memoria de 64 bits.
ADS	El estrobo de datos de dirección se activa cada vez que el Pentium envía una dirección válida de memoria o de E/S. Esta señal se combina con las señales W/R y M/IO para generar las señales de lectura y escritura separadas que se utilizan en los primeros sistemas basados en los microprocesadores 8086-80286.
AHOLD	La terminal retención de dirección es una entrada que hace que el Pentium retenga la dirección y las señales AP para el siguiente reloj.
APCHK	La terminal paridad de dirección proporciona una paridad par para la dirección de memoria en todas las transferencias de memoria y E/S iniciadas por el Pentium. La terminal AP también debe controlarse con información de paridad par en todos los ciclos de investigación en el mismo periodo de reloj que la señal EADS . La señal de comprobación de la paridad par se vuelve un 0 lógico cada vez que el Pentium detecta un error de paridad en la dirección.
BE₇-BE₀	Las señales habilitación de banco seleccionan el acceso de un byte, una palabra, una doble palabra o una palabra cuádruple de datos. El microprocesador genera estas señales en forma interna, a partir de los bits de dirección A ₀ , A ₁ y A ₂ .
BOFF	La entrada retroceso aborta todos los ciclos de bus pendientes y flota los buses del Pentium hasta que BOFF se niega. Después de negar BOFF , el Pentium reinicia todos los ciclos de bus que abortó en su totalidad.
BP₃-BP₀	Las terminales de punto de interrupción BP ₃ -BP ₀ indican una concordancia en el punto de interrupción cuando los registros se programan para monitorear las concordancias.
PM₁-PM₀	Las terminales monitoreo del rendimiento PM ₁ y PM ₀ indican la configuración de los bits de monitoreo del rendimiento en el registro de control en modo de depuración.

BRDY	La entrada ráfaga lista indica al Pentium que el sistema externo ha aplicado o extraído datos de las conexiones del bus de datos. Esta señal se utiliza para insertar estados de espera en la sincronización del Pentium.
BREQ	La salida p petición de bus indica que el Pentium ha generado una petición de bus.
BT₃-BT₀	Las salidas rastreo de ramificación proporcionan los bits 2-0 de la dirección lineal de destino de ramificación y el tamaño de operando predeterminado en BT ₃ . Estas salidas se hacen válidas durante un ciclo de mensaje especial de rastreo de ramificación.
BUSCHK	La entrada comprobación de bus permite que el sistema indique al Pentium que la transferencia del bus no ha tenido éxito.
CACHE	La salida de caché indica que el ciclo actual del Pentium puede colocar los datos en la caché.
CLK	La terminal reloj está controlada por una señal de reloj que define la frecuencia de operación del Pentium. Por ejemplo, para operar el Pentium a 66 MHz, aplique una señal de reloj de 66 MHz a dicha terminal.
D₆₃-D₀	Las conexiones del bus de datos transfieren datos tipo byte, palabra, doble palabra y palabra cuádruple entre el microprocesador y su sistema de memoria y de E/S.
D/C	La terminal datos/control indica que el bus de datos contiene datos que van hacia (o vienen desde) la memoria o la E/S cuando es un 1 lógico. Si D/C es un 0 lógico, el microprocesador se detiene o ejecuta un reconocimiento de interrupción.
DP₇-DP₀	La paridad de datos es generada por el Pentium y detecta sus ocho bancos de memoria a través de estas conexiones.
EADS	La entrada estrobo de dirección externo indica que el bus de direcciones contiene una dirección para un ciclo de exploración.
EWBE	La entrada vaciado de búfer de escritura externa indica que hay un ciclo de escritura pendiente en el sistema externo.
FERR	La señal error de punto flotante puede compararse con la línea ERROR en el 80386 y muestra que se ha producido un error en el coprocesador interno.
FLUSH	La entrada vaciado de caché hace que la caché vacíe todas las líneas de escritura aplazada y que invalide sus cachés internas. Si esta entrada es un 0 lógico durante una operación de reinicio, el Pentium entra en su modo de prueba.
FRCMC	La señal comprobación de redundancia funcional se muestrea durante un reinicio para configurar el Pentium en el modo maestro (1) o verificador (0).
HIT	La señal coincidencia indica que la caché interna contiene datos válidos en el modo de investigación.
HITM	La señal coincidencia modificada indica que el ciclo de exploración encontró una línea de caché modificada. Esta salida se utiliza para inhibir otras unidades maestras, de manera que no accedan a los datos hasta que la línea de caché se escriba en la memoria.
HOLD	La señal retención solicita una acción de DMA.
HLDA	La señal reconocimiento de retención indica que el Pentium se encuentra en una condición de retención.
IBT	La señal ramificación de instrucción tomada indica que el Pentium ha tomado una ramificación de una instrucción.
IERR	La salida error interno muestra que el Pentium ha detectado un error de paridad interno o un error de redundancia funcional.
IGNNE	La entrada ignora error numérico hace que el Pentium ignore un error del coprocesador numérico.
INIT	La entrada inicialización efectúa un reinicio sin inicializar las cachés, los búferes de escritura aplazada y los registros de punto flotante. Esto no puede usarse para reiniciar el microprocesador en lugar de RESET después de encender el equipo.

INTR	La p petición de interrupción es utilizada por circuitos externos para solicitar una interrupción.
INV	La entrada invalidación determina el estado de la línea de caché después de una investigación.
IU	La salida instrucción de canalización U completa muestra que se ha completado la instrucción en la canalización U.
IV	La salida instrucción de canalización V completa indica que se ha completado la instrucción en la canalización V.
KEN	La entrada habilita caché habilita la caché interna.
LOCK	Esta señal se vuelve un 0 lógico cada vez que se coloca el prefijo LOCK : en una instrucción. Esto se utiliza con más frecuencia durante los accesos de DMA.
M/IO	La señal memoria/ES selecciona un dispositivo de memoria cuando es un 1 lógico o un dispositivo de E/S cuando es un 0 lógico. Durante la operación de E/S, el bus de direcciones contiene una dirección de E/S de 16 bits en las conexiones de dirección A ₁₅ -A ₃ .
NA	La señal siguiente dirección indica que el sistema de memoria externa está listo para aceptar un nuevo ciclo de bus.
NMI	La señal interrupción no enmascarable solicita una interrupción no enmascarable, de igual forma que en las versiones anteriores del microprocesador.
PCD	La salida deshabilita caché de páginas muestra que el caché de páginas interno está deshabilitado cuando refleja el estado del bit PCD de CR ₃ .
PCHK	La salida comprobación de paridad indica un error de comprobación de paridad para los datos que se leen de memoria o de la E/S.
PEN	La entrada habilita paridad habilita la interrupción o excepción de comprobación del equipo.
PRDY	La salida sonda lista indica que se ha entrado al modo de sonda para la depuración.
PWT	La salida escritura directa de página muestra el estado del bit PWT en CR ₃ . Dicha terminal se proporciona para usarse con el puerto de depuración Intel y provoca una interrupción.
RESET	La señal reinicio inicializa el Pentium, con lo cual empieza a ejecutar software en la posición de memoria FFFFFFFF0H. El Pentium se reinicia en el modo real y las 12 conexiones de dirección de más a la izquierda permanecen como 1s (FFFH) hasta que se ejecuta un salto lejano o una llamada lejana. Lo anterior permite la compatibilidad con los microprocesadores anteriores. Consulte en la tabla 18-1 el estado del Pentium después de un reinicio de hardware.
SCYC	La salida ciclo dividido indica un ciclo de bus bloqueado (LOCK) mal alineado.
SMI	La entrada interrupción de administración del sistema hace que el Pentium entre al modo de operación de administración del sistema.
SMIACT	La salida interrupción de administración del sistema activa muestra que el Pentium está operando en el modo de administración del sistema.
TCK	La entrada reloj de testabilidad selecciona la función del reloj de acuerdo con la interfaz de exploración de límites IEEE 1149.1.
TDI	La entrada de datos de prueba se utiliza para probar los datos que se envían al Pentium mediante la señal TCK.
TDO	La salida de datos de prueba se utiliza para recopilar los datos de prueba y las instrucciones que se desplazan fuera del Pentium mediante TCK.
TMS	La entrada selección de modo de prueba controla la operación del Pentium en el modo de prueba. La entrada de reinicio de prueba permite reiniciar el modo de prueba.
W/R	La señal escritura/lectura indica que el ciclo de bus actual es una escritura cuando es un 1 lógico o una lectura cuando es un 0 lógico.

TABLA 18-1 Estado del Pentium después de un reinicio (RESET).

Registro	Valor RESET	Valor RESET + BIST
EAX	0	0 (si pasa la prueba)
EDX	0500XXXXH	0500XXXXH
EBX, ECX, ESP, EBP, ESI Y EDI	0	0
EFLAGS	2	2
EIP	0000FFF0H	0000FFF0H
CS	F000H	F000H
DS, ES, FS, GS y SS	0	0
GDTR y TSS	0	0
CR ₀	60000010H	60000010H
CR ₂ , CR ₃ y CR ₄	0	0
DR ₀ -DR ₃	0	0
DR ₆	FFFF0FF0H	FFFF0FF0H
DR ₇	00000400H	00000400H

WB/WT La señal **escritura aplazada/escritura directa** selecciona la operación para la caché de datos del Pentium.

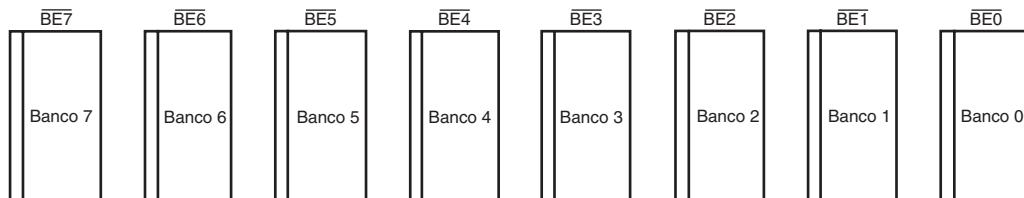
El sistema de memoria

El sistema de memoria para el microprocesador Pentium es de 4 Gbytes, igual que en los microprocesadores 80386DX y 80486. La diferencia está en la anchura del bus de datos de memoria. El Pentium utiliza un bus de datos de 64 bits para direccionar la memoria organizada en ocho bancos, cada uno de los cuales contiene 512 Mbytes de datos. En la figura 18-2 ve la organización del sistema de memoria física del Pentium.

El sistema de memoria del Pentium se divide en ocho bancos que almacenan cada uno un byte de datos con un bit de paridad. Al igual que el 80486, el Pentium emplea la generación de paridad y la lógica de comprobación de manera interna para la información del bus de datos del sistema de memoria. (Tenga en cuenta que la mayoría de los sistemas Pentium no utilizan comprobaciones de paridad, aunque esté disponible.) La memoria de 64 bits es importante para los datos de punto flotante con doble precisión. Recuerde que un número de punto flotante con doble precisión es de 64 bits. Debido al cambio de un bus de datos a uno de 64 bits, el Pentium puede extraer datos de punto flotante con un ciclo de lectura, en lugar de dos, como ocurre en el 80486. Esto hace que el Pentium funcione a mayor velocidad que un 80486. Al igual que con los microprocesadores Intel de 32 bits anteriores, el sistema de memoria se enumera en bytes, desde el byte 00000000H hasta el byte FFFFFFFFH.

La selección de memoria se lleva a cabo mediante las señales de habilitación de banco (\overline{BE}_7 - \overline{BE}_0). Estos bancos de memoria separados permiten que el Pentium tenga acceso a cualquier byte individual, palabra, doble palabra o palabra cuádruple con un solo ciclo de transferencia de memoria. Al igual que con la lógica de selección de memoria anterior, se generan ocho estrobos de escritura separados para escribir en el sistema de memoria.

Una nueva característica que se agregó al Pentium es su capacidad para comprobar y generar la paridad para el bus de dirección (A_{31} - A_5) durante ciertas operaciones. La terminal AP proporciona la in-

**FIGURA 18-2** Los bancos de memoria de 8 bytes del microprocesador Pentium.

formación de paridad al sistema y $\overline{\text{APCHK}}$ indica una comprobación de paridad errónea para el bus de direcciones. El Pentium no realiza ninguna acción cuando se detecta un error de paridad en la dirección. El sistema debe hacerse cargo del error tomando la acción apropiada (una interrupción), si así lo desea.

Sistema de entrada/salida

El sistema de entrada/salida del Pentium es 100% compatible con los microprocesadores Intel anteriores. El número de puerto de E/S aparece en las líneas de dirección A₁₅-A₃ y las señales de habilitación de banco se utilizan para seleccionar los bancos de memoria actuales que se utilizan para la transferencia de E/S.

Desde el microprocesador 80386, la información de privilegios de E/S se agrega al segmento TSS cuando el Pentium se opera en el modo protegido. Recuerde que esto permite inhibir los puertos de E/S en forma selectiva. Si se accede a la ubicación de E/S bloqueada, el Pentium genera una interrupción tipo 13 para indicar una violación de privilegios de E/S.

Sincronización del sistema

Al igual que con cualquier microprocesador, hay que comprender el funcionamiento de las señales de sincronización del sistema para conectar el microprocesador. En esta parte del libro hablaremos con detalles sobre la operación del Pentium mediante sus diagramas de sincronización y mostraremos cómo determinar los tiempos de acceso a la memoria.

El ciclo de memoria sin canalización básico del Pentium consiste en dos períodos de reloj: T₁ y T₂. En la figura 18-3 vea el ciclo de lectura sin canalización básico. En el diagrama de sincronización observamos que el Pentium de 66 MHz es capaz de realizar 33 millones de transferencias de memoria por segundo. Para ello, se supone que la memoria puede operar a esa velocidad.

Observe también en el diagrama de sincronización que la señal W/R se vuelve válida si $\overline{\text{ADS}}$ es un 0 lógico en el borde positivo del reloj (fin de T₁). Este reloj debe usarse para calificar el ciclo como una lectura o una escritura.

Durante T₁, el microprocesador envía las señales $\overline{\text{ADS}}$, W/R, de dirección y M/I $\overline{\text{O}}$. Para calificar la señal W/R y generar las señales $\overline{\text{MRDC}}$ y $\overline{\text{MWTC}}$ apropiadas, utilizaremos un flip-flop para generar la señal W/R. Después, un multiplexor de dos líneas a una línea generará las señales de control de memoria

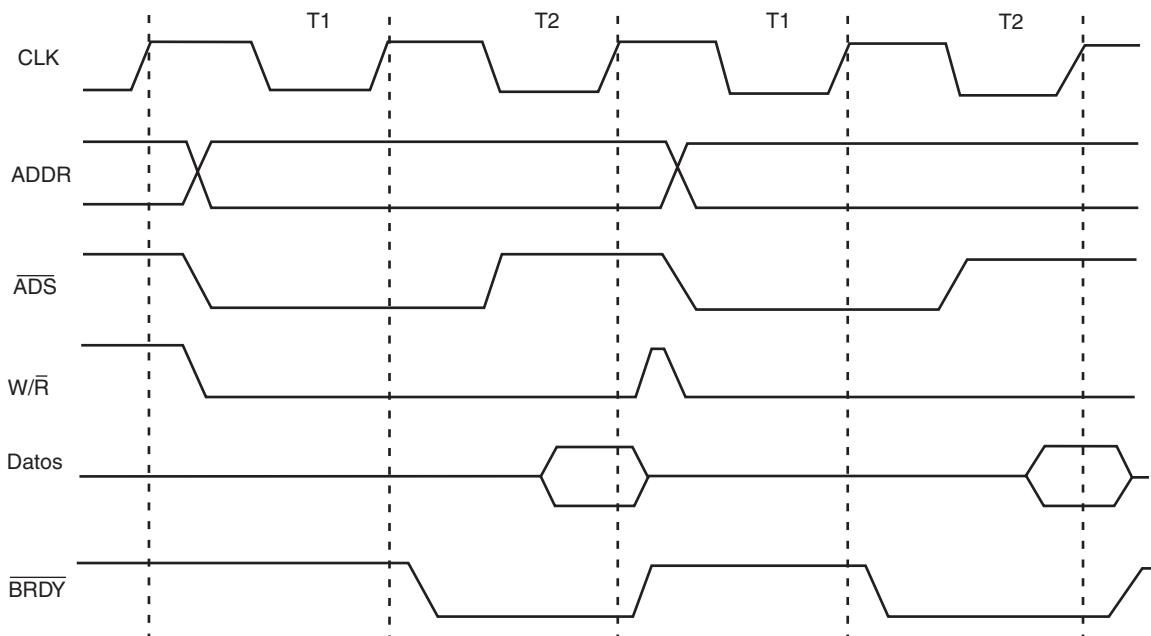


FIGURA 18-3 El ciclo de lectura sin canalización para el microprocesador Pentium.

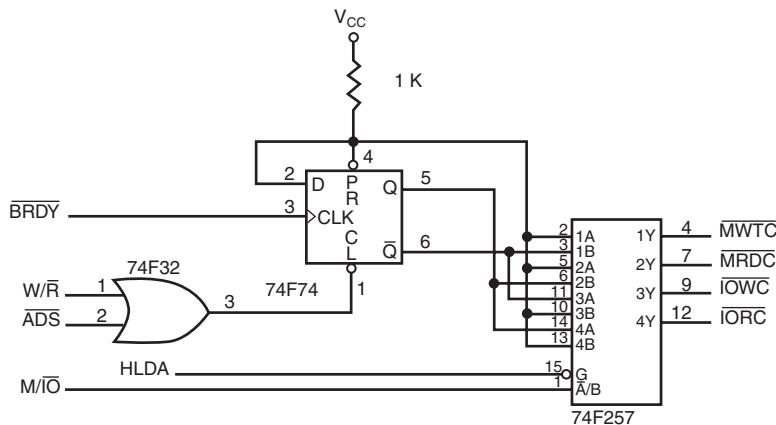


FIGURA 18-4 Un circuito que genera las señales de control de la memoria y de la E/S.

y de E/S. En la figura 18-4 verá un circuito que genera las señales de control de memoria y de E/S para el microprocesador Pentium.

Durante T_2 , el bus de datos se muestrea en sincronización con el fin de T_2 en la transición positiva del pulso de reloj. El tiempo de preparación antes del reloj se da como 3.8 ns y el tiempo de retención después del reloj se da como 2.0 ns. Esto significa que la ventana de datos alrededor del reloj es de 5.8 ns. La dirección aparece en el máximo de 8 ns después del inicio de T_1 . Esto significa que el microprocesador Pentium que opera a 66 MHz permite 30.3 ns (dos períodos de reloj), menos el tiempo de retraso de dirección de 8.0 ns y el tiempo de preparación de los datos de 3.8 ns. El tiempo de acceso a memoria sin usar estados de espera sería de $30.3 - 8.0 - 3.8$ o 18.5 ns. Esto es tiempo suficiente para permitir el acceso a una SRAM, pero no para cualquier DRAM sin insertar estados de espera en la sincronización. Por lo general, la SRAM se encuentra en la forma de una caché externa de nivel 2.

Los estados de espera se insertan en la sincronización mediante el control de la entrada BRDY que va hacia el Pentium. Esta señal debe volverse un 0 lógico antes de que termine T_2 o se insertarán estados de espera adicionales a T_2 en la sincronización. En la figura 18-5 se observa el diagrama de sincronización del ciclo de lectura, el cual contiene estados de espera para memoria lenta. El efecto de insertar estados de espera en la sincronización es para extender la sincronización, lo cual permite un tiempo adicional para que la memoria acceda a los datos. En el diagrama de sincronización que se muestra, se ha extendido el tiempo de acceso de manera que pueda usarse una DRAM estándar en un

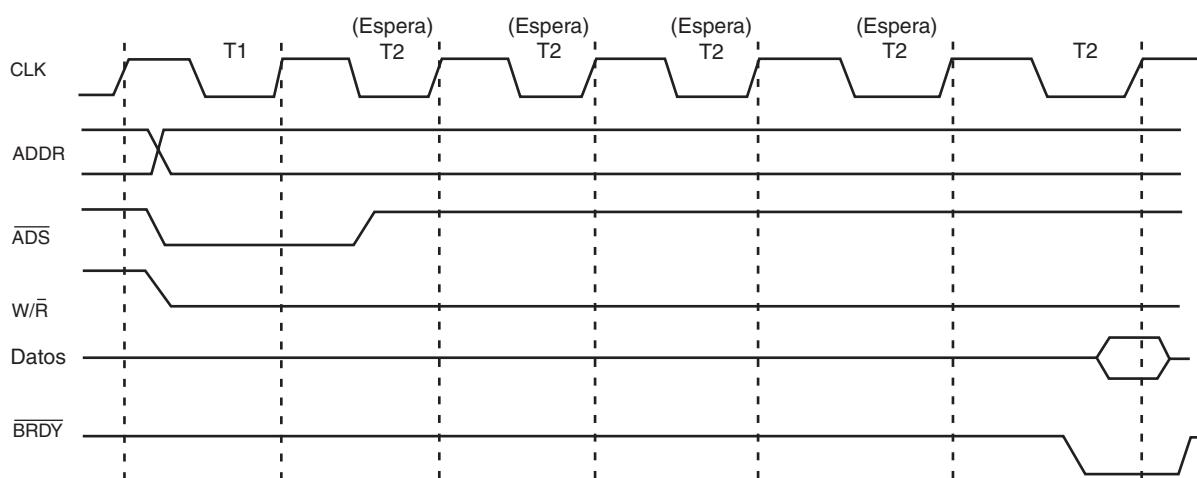


FIGURA 18-5 Diagrama de sincronización del Pentium con cuatro estados de espera insertados, para un tiempo de acceso de 79.5 ns.

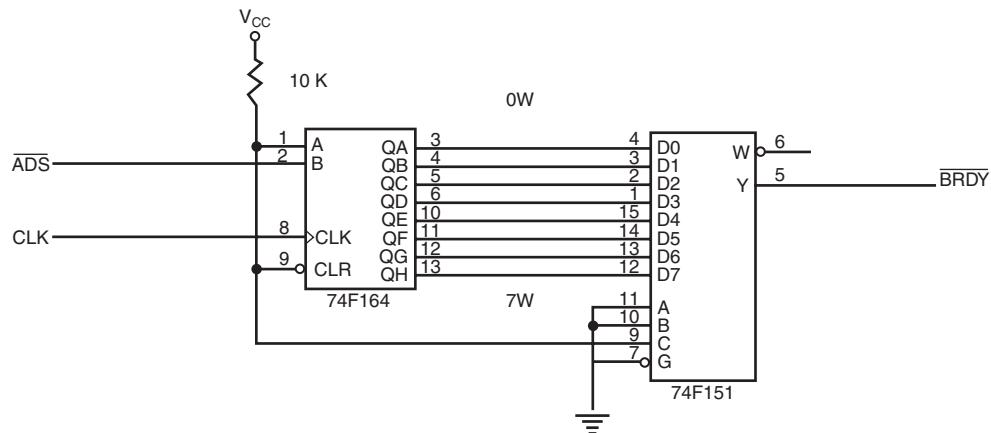


FIGURA 18-6 Un circuito que genera estados de espera mediante el retraso de ADS. El circuito está cableado para generar cuatro estados de espera.

sistema. Para esto se requiere la inserción de cuatro estados de espera de 15.2 ns (un periodo de reloj) cada uno, para extender el tiempo de acceso hasta 79.5 ns. El anterior es el tiempo suficiente para que funcionen la DRAM y cualquier decodificador en el sistema.

La señal $\overline{\text{BRDY}}$ es una señal sincrónica que se genera mediante el uso del reloj del sistema. La figura 18-6 muestra un circuito que puede usarse para generar $\overline{\text{BRDY}}$ para insertar cualquier número de estados de espera en el diagrama de sincronización del Pentium. Tal vez recuerde un circuito similar que inserta estados de espera en el diagrama de sincronización del microprocesador 80386. La señal $\overline{\text{ADS}}$ se retraza entre 0 y 7 períodos de reloj mediante el registro de desplazamiento 74F161 para generar la señal $\overline{\text{BRDY}}$. El multiplexor 74F151 de ocho líneas a una línea selecciona el número exacto de estados de espera. En este ejemplo, el multiplexor selecciona la salida de cuatro estados de espera del registro de desplazamiento.

Un método más eficiente para leer datos de la memoria es a través del ciclo de ráfaga. Este ciclo en el Pentium transfiere cuatro números de 64 bits por cada ciclo de ráfaga en cinco períodos de reloj. Una ráfaga sin estados de espera requiere que el sistema de memoria transfiera datos cada 15.2 ns. Si se cuenta con una caché de nivel 2, esta velocidad no representará un problema siempre y cuando los datos se lean desde la caché. Si la caché no contiene los datos, deberán insertarse estados de espera, lo cual reducirá la producción de datos de salida. En la figura 18-7 verá la transferencia del ciclo de ráfaga del Pentium sin estados de espera. Como antes, es posible insertar estados de espera para permitir que el sistema de memoria tenga más tiempo para los accesos.

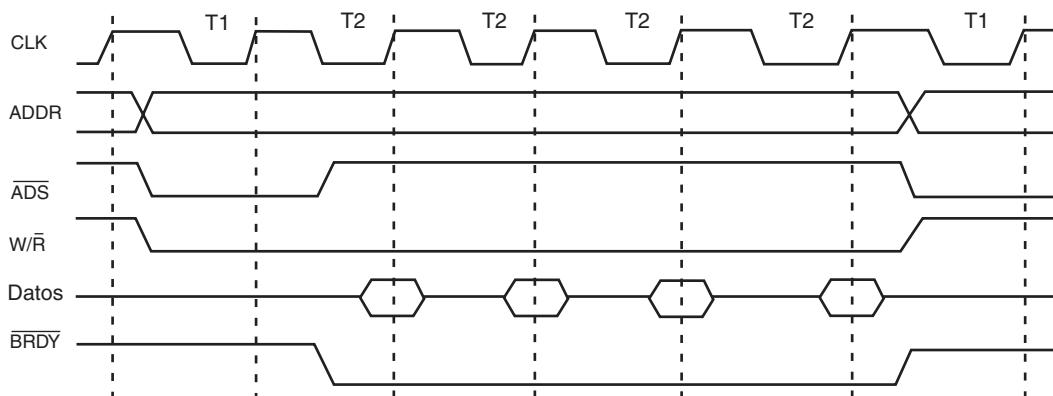


FIGURA 18-7 La operación del ciclo de ráfaga del Pentium que transfiere cuatro datos de 64 bits entre el microprocesador y la memoria.

Lógica de predicción de ramificación

El microprocesador Pentium utiliza una lógica de predicción de ramificación para reducir el tiempo requerido para una ramificación producida por retrasos internos. Estos retrasos se minimizan debido a que cuando se encuentra una instrucción de ramificación (sólo corta o cercana), el microprocesador comienza la búsqueda previa de la instrucción en la dirección de ramificación. Las instrucciones se cargan en la caché de instrucciones, por lo que cuando se produce la ramificación las instrucciones están presentes y permiten que la ramificación se ejecute en un periodo de reloj. Si por cualquier razón se produce un error en la lógica de predicción de ramificación, la ramificación requerirá tres períodos de reloj adicionales para ejecutarse. En la mayoría de los casos, la predicción de ramificación es correcta y no se producen retrasos.

Estructura de la caché

La caché en el Pentium se modificó, en comparación con la que se encuentra en el microprocesador 80486. El Pentium contiene dos memorias caché de 8 Kbytes, en lugar de una como en el 80486. Hay una caché de datos de 8 Kbytes y una caché de instrucciones de 8 Kbytes. La caché de instrucciones sólo almacena instrucciones, mientras que la caché de datos almacena los datos que utilizan las instrucciones.

En el 80486 con su caché unificada, un programa que utilizaba datos en forma intensiva llenaba la caché con rapidez y dejaba muy poco espacio para las instrucciones. Lo anterior reducía la velocidad de ejecución del microprocesador 80486. En el Pentium, esto no llega a ocurrir debido a la caché de instrucciones separada.

Arquitectura superescalar

El microprocesador Pentium está organizado con tres unidades de ejecución. Una ejecuta las instrucciones de punto flotante; las otras dos (canalizaciones U y V) ejecutan instrucciones de enteros. Lo anterior significa que es posible ejecutar tres instrucciones a la vez. Por ejemplo, la instrucción FADD ST, ST(2), la instrucción MOV EAX,10H y la instrucción MMOV EBX,12H pueden ejecutarse al mismo tiempo, ya que ninguna de tales instrucciones depende de las otras. El coprocesador ejecuta la instrucción FADD ST,ST(2); la canalización U ejecuta la instrucción MOV EAX,10H y la canalización V ejecuta la instrucción MOV EBX,12H. Como la unidad de punto flotante también se utiliza para las instrucciones MMX (si están disponibles), el Pentium puede ejecutar dos instrucciones con enteros y una instrucción MMX de manera simultánea.

Se debe escribir software para aprovechar esta característica mediante un análisis de las instrucciones en un programa y luego mediante su modificación, cuando se descubran casos en los que instrucciones dependientes puedan separarse mediante instrucciones no dependientes. Estas modificaciones llegan a producir un aumento de hasta el 40% de la velocidad de ejecución en cierto software. Asegúrese de que cualquier compilador reciente u otro paquete de aplicación aprovechen esta nueva característica superescalar del Pentium.

18-2

REGISTROS ESPECIALES DEL PENTIUM

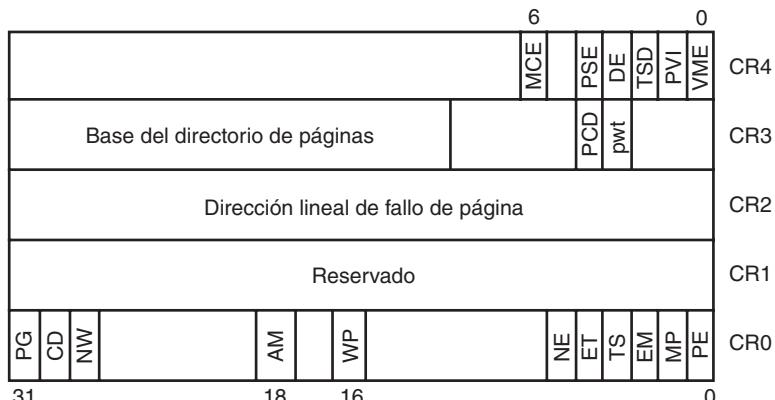
En esencia, el Pentium es el mismo microprocesador que el 80386 y el 80486, sólo que se han agregado ciertas características adicionales y algunas modificaciones al conjunto de registros de control. En esta sección resaltaremos las diferencias entre la estructura de los registros de control del 80386 y el registro de banderas.

Registros de control

La figura 18-8 muestra la estructura de los registros de control para el microprocesador Pentium. Observe que se agregó un nuevo registro de control (CR₄) al arreglo de registros de control.

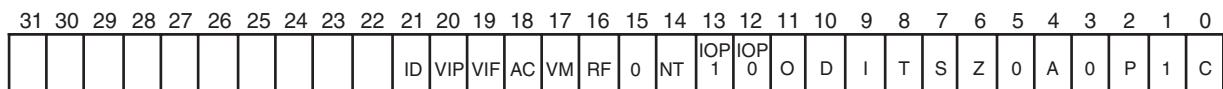
En esta sección sólo explicaremos los nuevos componentes del Pentium en los registros de control. En la figura 17-4 encontrará una descripción y una ilustración de los registros de control del 80386. A continuación aparece una descripción de los nuevos bits de control y del nuevo registro de control CR₄:

FIGURA 18-8 La estructura de los registros de control del Pentium.



- CD** El bit **deshabilita caché** controla la caché interna. Si CD = 1, la caché no se llenará con nuevos datos cuando no estén incluidos, pero continuará funcionando para las coincidencias de la caché. Si CD = 0, las omisiones harán que la caché se llene con datos nuevos.
- NW** El bit **sin escritura directa** selecciona el modo de operación para la caché de datos. Si NW = 1, la caché de datos se inhibe de la escritura directa.
- AM** El bit **máscara de alineación** permite la comprobación de la alineación cuando está activo. Tenga en cuenta que la comprobación de la alineación sólo se lleva a cabo, para la operación en modo protegido, cuando el usuario se encuentra en el nivel de privilegio 3.
- WP** El bit **protección contra escritura** protege las páginas a nivel de usuario contra las operaciones de escritura a nivel de supervisor. Cuando WP = 1, el supervisor puede escribir en los segmentos a nivel de usuario.
- NE** El bit **error numérico** habilita la detección de errores estándar del coprocesador numérico. Si NE = 1, la terminal FERR se activa para un error del coprocesador numérico. Si NE = 0, se ignora cualquier error del coprocesador.
- VME** El bit **extensión de modo virtual** permite el soporte para la bandera de interrupciones virtuales en modo protegido. Si VME = 0, se deshabilita el soporte para interrupciones virtuales.
- PVI** El bit **interrupción virtual en modo protegido** habilita el soporte para la bandera de interrupciones virtuales en modo protegido.
- TSD** El bit **deshabilita estampa de tiempo** controla la instrucción RDTSC.
- DE** El bit **extensión de depuración** habilita las extensiones de depuración de punto de interrupción de E/S cuando se activa.
- PSE** El bit **extensión de tamaño de página** habilita las páginas de memoria de 4 Mbytes cuando se activa.
- MCE** El bit **habilita comprobación de equipo** habilita la interrupción de comprobación de equipo.

El Pentium contiene nuevas características que se controlan mediante CR₂ y unos cuantos bits en CR₀. En secciones posteriores del libro explicaremos estas nuevas características.



Nota: Los bits en blanco en el registro de bandera están reservados para usos futuros que no están definidos.

FIGURA 18-9 La estructura del registro EFLAG del Pentium.

Registro EFLAG

El registro de banderas extendido (EFLAG) se ha modificado en el microprocesador Pentium. La figura 18-9 muestra el contenido del registro EFLAG. Observe que se han agregado cuatro nuevos bits de bandera a este registro para controlar o indicar condiciones sobre algunas de las nuevas características en el Pentium. A continuación se muestra una lista de las cuatro nuevas banderas y la función de cada una de ellas:

- ID** La **bandera de identificación** se utiliza para comprobar la instrucción CPUID. Si un programa puede activar y borrar la bandera ID, significa que el procesador soporta la instrucción CPUID.
- VIP** **Interrupción virtual pendiente** indica que hay una interrupción virtual pendiente.
- VIF** **Interrupción virtual** es la imagen de la bandera de interrupciones IF que se utiliza con VIP.
- AC** **Comprobación de alineación** indica el estado del bit AM en el registro de control 0.

Autoprueba integrada (BIST)

Para acceder a la autoprueba integrada (BIST), al encender el equipo se coloca un 1 lógico en INIT, mientras que la terminal RESET cambia de 1 a 0. La BIST evalúa el 70% de la estructura interna del Pentium en un tiempo aproximado de 150 µs. Al completarse la BIST, el Pentium reporta el resultado en el registro EAX. Si EAX = 0, el Pentium pasó la prueba BIST y está listo para operar. Si EAX contiene cualquier otro valor, el Pentium tiene una falla.

18-3

ADMINISTRACIÓN DE MEMORIA DEL PENTIUM

La unidad de administración de memoria dentro del Pentium es compatible en forma ascendente con los microprocesadores 80386 y 80486. Muchas de las características de estos microprocesadores no se modificaron en el Pentium. Los principales cambios son la unidad de paginación y un nuevo modo de administración de memoria del sistema.

Unidad de paginación

El mecanismo de paginación funciona con páginas de memoria de 4 Kbytes, o con una nueva extensión disponible para el Pentium con páginas de memoria de 4 Mbytes. Como se describió en los capítulos 1 y 17, el tamaño de la estructura de tablas de paginación puede crecer de manera considerable en un sistema que contenga una memoria grande. Recuerde que para realizar una repaginación completa de 4 Gbytes de memoria, el microprocesador requiere un poco más de 4 Mbytes de memoria sólo para las tablas de páginas. En el Pentium, con la nueva característica de paginación de 4 Mbytes, esto se reduce de una forma dramática a sólo un directorio de página individual y sin tablas de paginación. El bit PSE en el registro de control 0 selecciona los nuevos tamaños de página de 4 Mbytes.

La principal diferencia entre la paginación de 4 K y la paginación de 4 M es que en el esquema de paginación de 4 M no hay entrada de tabla de página en la dirección lineal. En la figura 18-10 verá el sistema de paginación 4 M en el microprocesador Pentium. Ponga mucha atención a la manera en que se utiliza la dirección lineal con este esquema. Observe que los 10 bits de más a la izquierda de la dirección lineal seleccionan una entrada en el directorio de páginas (justo igual que con las páginas de 4 K). A diferencia de las páginas de 4 K, no hay tablas de páginas; en lugar de ello, el directorio de páginas dirige una página de memoria de 4 Mbytes.

Modo de administración de memoria

El modo de administración de memoria del sistema (SMM) está en el mismo nivel que el modo protegido, el modo real y el modo virtual, sólo que se proporciona para funcionar como administrador. El SMM no se diseñó para usarse como característica a nivel de aplicación o de sistema, sino para funciones del sistema de alto nivel tales como la administración de la energía y la seguridad, elementos que la mayoría de los Pentium utilizan durante su operación.

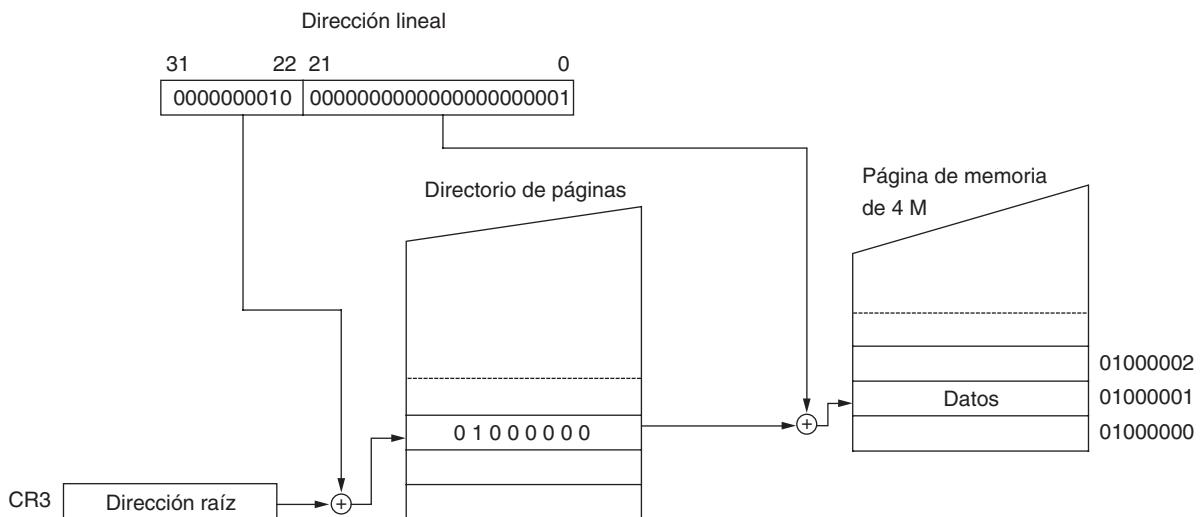


FIGURA 18-10 La dirección lineal 00200001H se repagina hacia la posición de memoria 01000002H en páginas de 4 Mbytes. Observe que no hay tablas de páginas.

El acceso al SMM se logra a través de una nueva interrupción de hardware externa que se aplica a la terminal SMI en el Pentium. Cuando se activa la interrupción SMM, el procesador comienza a ejecutar software a nivel de sistema en un área de memoria conocida como RAM de administración del sistema o SMMRAM (registro de vaciado de estado del SMM). La interrupción SMI deshabilita todas las demás interrupciones que por lo general se manejan mediante las aplicaciones de usuario y el sistema operativo. Para regresar de la interrupción SMM se utiliza una nueva instrucción llamada RSM. Esta instrucción regresa de la interrupción del modo de administración de memoria hacia el programa que se paralizó en el punto de la interrupción.

La interrupción SMM llama al software, que en un principio se almacena en la posición de memoria 38000H, mediante el uso de CS = 3000H y EIP = 8000H. Este estado inicial puede modificarse con el uso de un salto hacia cualquier posición dentro del primer 1 Mbyte de la memoria. Mediante la interrupción del modo de administración se entra a un entorno similar al direccionamiento de memoria en modo real, pero es distinto porque en lugar de poder direccionar el primer 1 Mbytes de memoria, el modo SMM permite que el Pentium trate al sistema de memoria como un sistema plano de 4 Gbytes.

Además de ejecutar software que empieza en la posición 38000H, la interrupción SMM también almacena el estado del Pentium en lo que se conoce como **registro de vaciado**. Este registro se almacena en las posiciones de memoria 3FFA8H a 3FFFFH, con un área en las posiciones 3FE00H a 3FEF7H reservada por Intel. El registro de vaciado permite que un sistema basado en el Pentium entre a un modo de inactividad y se reactive en el punto de la interrupción del programa. Para ello se requiere que la SMMRAM reciba energía durante el periodo de inactividad. Muchas computadoras portátiles (laptops) tienen una batería separada para energizar la SMMRAM durante muchas horas en el modo de inactividad. En la tabla 18-2 se muestra el contenido del registro de vaciado.

Las opciones Detención de reinicio automático y Reinicio de trampa de E/S se utilizan cuando se sale del modo SMM mediante la instrucción RSM. Estos datos permiten que la instrucción RSM regrese al estado de detención o a la instrucción de E/S de la interrupción. Si no se efectúa una detención o una operación de E/S al momento de entrar al modo SMM, la instrucción RSM vuelve a cargar el estado del equipo desde el vaciado de estado y regresa al punto de la interrupción.

El sistema hace posible usar el modo SMM antes de que se coloque el sistema operativo normal en la memoria y se ejecute. También puede utilizarse en forma periódica para administrar el sistema, siempre y cuando no haya software normal en las posiciones 38000H-3FFFFH. Si el sistema reubica la SMMRAM antes de iniciar el sistema operativo normal, se vuelve disponible para usarse además del sistema normal.

Para cambiar la dirección base de la SMMRAM en modo SMM, se modifica el valor en el registro de dirección base de vaciado de estado (posiciones 3FEF8H a 3F3FBH) después de la primera interrupción

TABLA 18-2 Registro de vaciado de estado del SMM del Pentium.

<i>Dirección de desplazamiento</i>	<i>Registro</i>
FFFCH	CR0
FFF8H	CR3
FFF4H	EFLAGS
FFF0H	EIP
FFECH	EDI
FFE8H	ESI
FFE4H	EBP
FFE0H	ESP
FFDCH	EBX
FFD8H	EDX
FFD4H	ECX
FFD0H	EAX
FFCCH	DR6
FFC8H	DR7
FFC4H	TR
FFC0H	LDTR
FFBCH	GS
FFB8H	FS
FFB4H	DS
FFB0H	SS
FFA8H	CS
FF04H-FFA7H	ES
FF02H	Reservado.
FF00H	Detención de inicio automático.
FEFCH	Reinicio de trampa de E/S.
FED8H	Identificación de revisión de SMM.
FE00H-FEF7H	Base de vaciado de estado.
	Reservado.

Nota: Al principio las direcciones de desplazamiento se ubican en la dirección base 00003000H.

en modo de administración de memoria. Cuando se ejecuta la primera instrucción RSM, la cual devuelve el control al sistema que se interrumpió, el nuevo valor de tales ubicaciones modifica la dirección base de la interrupción SMM para todos los usos futuros. Por ejemplo, si la dirección base de vaciado de estado se modifica a 000E8000H, todas las interrupciones SMM subsiguientes utilizarán las posiciones E8000H-EFFFFH para el vaciado de estado del Pentium. Estas posiciones son compatibles con DOS y Windows.

18-4

INSTRUCCIONES NUEVAS DEL PENTIUM

El Pentium contiene sólo una nueva instrucción que funciona con el software de sistema normal; el resto de las nuevas instrucciones se agregaron para controlar la característica del modo de administración de memoria y las instrucciones de serialización. La tabla 18-3 muestra las nuevas instrucciones que se agregaron al conjunto de instrucciones del Pentium.

La instrucción CMPXCHG8B es una extensión de la instrucción CMPXCHG que se agregó al conjunto de instrucciones del 80486. La instrucción CMPXCHG8B compara el número de 64 bits almacenado en EDX y EAX con el contenido de una posición de memoria de 64 bits o de un par de registros. Por ejemplo, la instrucción CMPXCHG8B DATOS2 compara los ocho bytes que se almacenan en la posición de memoria DATOS2 con el número de 64 bits en EDX y EAX. Si DATOS2 es igual a EDX:EAX, el número de 64 bits almacenado en ECX:EBX se almacena en la posición de datos DATOS2. Si no son iguales, el contenido de DATOS2 se almacena en EDX:EAX. Observe que el bit de bandera cero indica que el contenido de EDX:EAX fue o no igual a DATOS2.

TABLA 18-3 Instrucciones nuevas del Pentium.

<i>Instrucción</i>	<i>Función</i>
CMPXCHG8B	Compara y cambia ocho bytes.
CPUID	Regresa al CPU a código de identificación.
RDTSC	Contador de lectura de estampa de tiempo.
RDMSR	Registro de lectura de modelo específico.
WRMSR	Registro de escritura de modelo específico.
RSM	Interrupción del retorno del sistema administrador.

TABLA 18-4 Información de la instrucción CPUID.

<i>Valor de entrada (EAX)</i>	<i>Resultado después de CPUID</i>
0	EAX = 1 para todos los microprocesadores. EBX-EDX-ECX = Información del fabricante.
1	EAX (bits 3–0) = ID de escalonamiento. EAX (bits 7–4) = Modelo. EAX (bits 11–8) = Familia. EAX (bits 13–12) = Tipo. EAX (bits 31–14) = Reservado. EDX (bit 0) = CPU contiene FPU. EDX (bit 1) = Soporte para modo virtual 8086 mejorado. EDX (bit 2) = Soporte para puntos de interrupción de E/S. EDX (bit 3) = Soporte para extensiones de tamaño de página. EDX (bit 4) = Soporte para contador de estampa de tiempo. EDX (bit 5) = Soporte para MSR estilo Pentium. EDX (bit 6) = Reservado. EDX (bit 7) = Soporte para excepción de comprobación de equipo. EDX (bit 8) = Soporte para CMPXCHG8B. EDX (bit 9) = Microprocesador de 3.3V. EDX (bit 10–31) = Reservado.

La instrucción CPUID lee el código de identificación del CPU y demás información del Pentium. La tabla 18-4 muestra distinta información devuelta de la instrucción CPUID para diversos valores de entrada para EAX. Para utilizar la instrucción CPUID, primero cargue EAX con el valor de entrada y después ejecute CPUID. La información se devolverá en los registros que se indican en la tabla.

Si se coloca un 0 en EAX antes de ejecutar la instrucción CPUID, el microprocesador devuelve la identificación del fabricante en EBX, EDX y ECX. Por ejemplo, el Intel Pentium devuelve “Genuine-Intel” en código ASCII con la parte “Genu” en EBX, “ineI” en EDX y “ntel” en ECX. El registro EDX devuelve información si EAX se carga con un 1 antes de ejecutar la instrucción CPUID.

El ejemplo 18-1 muestra un programa corto que lee la información del fabricante con la instrucción CPUID. Este software se colocó en la sección TODO: de la función OnInitDialog de una aplicación de cuadro de diálogo simple. Después viene el mensaje en la pantalla de vídeo en una etiqueta ActiveX, como se indica en la figura 18-11. La instrucción CPUID funciona tanto en modo real como en modo protegido, y puede usarse en cualquier aplicación Windows.

EJEMPLO 18-1

```
CString temp;
int a, b, c;
__asm
{
    mov    eax, 0
    cpuid
    mov    a,ebx
    mov    b,edx
    mov    c,ecx
}
```

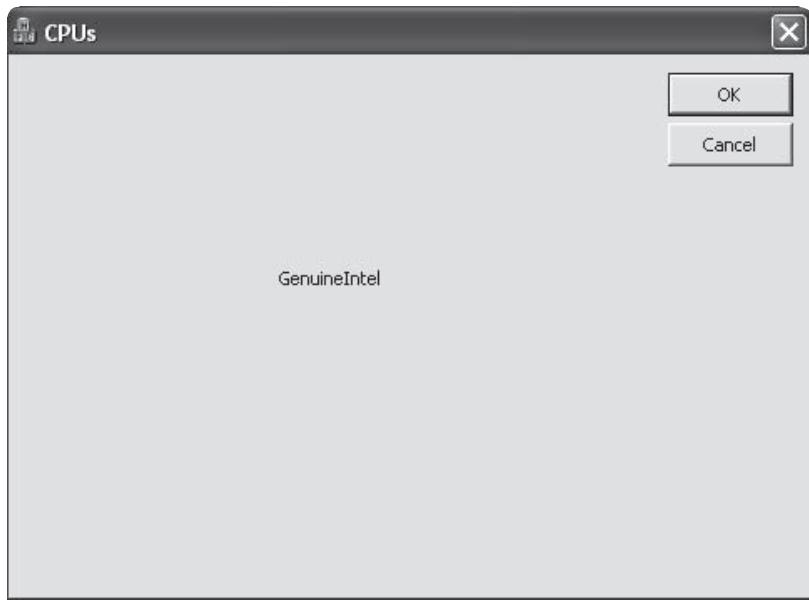


FIGURA 18-11 Captura de pantalla del programa del ejemplo 18-1, en donde se utiliza la instrucción CPUID.

```

}
for (int d = 0; d < 4; d++)
{
    temp += (char)a;
    a >>= 8;
}
for (d = 0; d < 4; d++ )
{
    temp += (char)b;
    b >>= 8;
}
for (d = 0; d < 4; d++ )
{
    temp += (char)c;
    c >>= 8;
}
Etiqueta1.put_Caption(temp);

```

La instrucción RDTSC lee el contador de estampa de tiempo y lo coloca en EDX:EAX. Este contador cuenta los ciclos de reloj del CPU desde el tiempo en que se reinicia el microprocesador, donde el contador de estampa de tiempo se inicializa con un conteo desconocido. Como éste es un conteo de 64 bits, un microprocesador de 1 GHz puede acumular un conteo de más de 580 años antes de que el contador de estampa de tiempo de otra vuelta. Esta instrucción sólo funciona en modo real o con el nivel de privilegio 0 en modo protegido.

El ejemplo 18-2 muestra una clase escrita para Windows que proporciona funciones miembro para retrasos de tiempo precisos y funciones miembro para medir los tiempos de ejecución del software. Para agregar esta clase, se hace clic con el botón derecho sobre el nombre del proyecto y se inserta una clase MFC genérica llamada TiempoD. Esta clase contiene tres funciones miembro llamadas Inicio, Paro y Retraso.

La función Inicio() se utiliza para iniciar una medición y Paro(), para detener una medición de tiempo. La función Paro() devuelve un valor de punto flotante que representa la cantidad de tiempo en microsegundos entre Inicio() y Paro().

La función Retraso produce un retraso de tiempo de precisión con base en el contador de estampa de tiempo. El parámetro que se transfiere a la función Retraso está en milisegundos. Lo anterior significa que Retraso(1000) produce la cantidad exacta de 1000 ms de retraso.

Cuando se inicializa TiempoD en un programa, lee la frecuencia del microprocesador en MHz del archivo del registro de Windows, mediante el uso de la función RegQueryValueEx después de abrirlo con la función RegOpenKeyEx. En la variable de clase MicroFrecuencia se devuelve la frecuencia del reloj del microprocesador.

EJEMPLO 18-2

```
#include "StdAfx.h"
#include ".\tiempod.h"

int MicroFrecuencia;           //frecuencia en MHz
_int64 Conteo;

TiempoD::TiempoD(void)
{
    HKEY hKey;
    DWORD tamanioDatos;
        // Obtiene la frecuencia del procesador

    if ( RegOpenKeyEx (HKEY_LOCAL_MACHINE,
                       "Hardware\\Description\\System\\CentralProcessor\\0",
                       0, KEY_QUERY_VALUE, &hKey) == ERROR_SUCCESS )
    {
        RegQueryValueEx (hKey, _T("~MHz"), NULL, NULL,
                         (LPBYTE)&MicroFrecuencia, &tamanioDatos);
        RegCloseKey (hKey);
    }
}

TiempoD::~TimeD(void)
{
}

void TiempoD::Inicio(void)
{
    __asm
    {
        rdtsc          ;obtiene y almacena TSC
        mov dword ptr Conteo,eax
        mov dword ptr Conteo+4,edx
    }
}

double TiempoD::Paro(void)
{
    __asm
    {
        rdtsc
        sub eax,dword ptr Conteo
        mov dword ptr Conteo,eax
        sbb edx,dword ptr Conteo+4
        mov dword ptr Conteo+4,edx
    }
    return (double)Conteo/MicroFrecuencia;
}

void TiempoD::Retraso(_int64 milisegundos)
{
    milisegundos *= 1000;           //convierte a milisegundos
    milisegundos *= MicroFrecuencia; //convierte en conteo puro
    __asm {
        mov ebx, dword ptr milisegundos      ;retraso de 64 bits en ms
        mov ecx, dword ptr milisegundos+4
        rdtsc          ;obtiene conteo
        add ebx, eax
        adc ecx, edx           ;avanza conteo en base a retraso
    }
}
```

```

Retraso_CICLO1:           ;espera a que conteo se ponga al corriente

    rdtsc
    cmp    edx, ecx
    jb     Retraso_CICLO1
    cmp    eax, ebx
    jb     Retraso_CICLO1
}
}

```

Si necesita un Retraso adicional, podría agregarse a la clase para provocar retrasos en microsegundos, pero se debe hacer una restricción para que no sea menor que 2 o 3 microsegundos, debido al tiempo que se requiere para sumar el tiempo al conteo del contador de estampa de tiempo.

El ejemplo 18-3 muestra una aplicación de cuadro de diálogo de ejemplo que utiliza Retraso() para esperar un segundo después de hacer clic en el botón antes de modificar el color de primer plano de una Etiqueta ActiveX. Lo que no aparece en el ejemplo es que al principio de la clase del cuadro de diálogo se muestra una instrucción #include "TiempoD.h". El software en sí está en la sección TODO: de la función OnInitDialog.

EJEMPLO 18-3

```

void CRDTSCDlg::OnBnClickedButton1()
{
    TiempoD temporizador;
    temporizador.Retraso(1000);
    Etiqueta1.put_ForeColor(0xff0000);
}

```

Las instrucciones RDMSR y WRMSR permiten la lectura y la escritura en los registros específicos para el modelo. Estos registros son únicos para el Pentium y se utilizan para rastrear, comprobar el rendimiento, evaluar y comprobar errores del equipo. Ambas instrucciones utilizan ECX para llevar el número de registro al microprocesador y EDX:EAX para la lectura o escritura de 64 bits. Observe que las direcciones de los registros son 0H-13H. En la tabla 18-5 verá un lista de los registros específicos para el modelo Pentium y su contenido. Al igual que con la instrucción RDTSC, estos registros específicos para el modelo operan en el modo real o en el nivel de privilegio 0 del modo protegido.

TABLA 18-5
Los registros específicos para el modelo Pentium.

Dirección (ECX)	Tamaño	Función
00H	64 bits	Dirección de excepción de comprobación de equipo.
01H	5 bits	Tipo de excepción de comprobación de equipo.
02H	14 bits	Registro de prueba de inversión de paridad TR ₁ .
03H	—	—
04H	4 bits	Bits de fin de caché de instrucción TR ₂ .
05H	32 bits	Datos de caché TR ₃ .
06H	32 bits	Etiqueta de caché TR ₄ .
07H	15 bits	Control de caché TR ₄ .
08H	32 bits	Comando TLB TR ₆ .
09H	32 bits	Datos TLB TR ₇ .
0AH	—	—
0BH	32 bits	Etiqueta BTB TR ₉ .
0CH	32 bits	Destino BTB TR ₁₀ .
0DH	12 bits	Control BTB TR ₁₁ .
0EH	10 bits	Control de nueva característica TR ₁₂ .
0FH	—	—
10H	64 bits	Contador de estampa de tiempo (puede escribirse en él).
11H	26 bits	Selección y control de contador de eventos.
12H	40 bits	Contador de eventos 0.
13H	40 bits	Contador de eventos 1.

Nunca debe utilizar un valor indefinido en ECX antes de las instrucciones RDMSR o WRMSR. Si ECX = 0, antes de la instrucción de lectura o escritura del registro específico del equipo, el valor devuelto (EDX:EAX) será la dirección de excepción de comprobación de equipo. (EDX:EAX es en donde residen todos los datos cuando se escriben o se leen de los registros específicos para el modelo.) Si ECX = 1, el valor es el tipo de excepción de comprobación del equipo; si ECX = 0EH, se accede al registro de prueba 12 (TR₁₂). Tenga en cuenta que éstos son registros internos diseñados para pruebas internas. El contenido de tales registros es propiedad de Intel, por lo que no deben utilizarse durante la programación normal.

18-5**INTRODUCCIÓN AL MICROPROCESADOR PENTIUM PRO**

Antes de utilizar este microprocesador o cualquier otro en un sistema, hay que comprender la función de cada una de sus terminales. En esta sección veremos los detalles acerca de la operación de cada terminal, junto con las estructuras del sistema de memoria externa y de E/S del microprocesador Pentium Pro. En la figura 18-2 se muestra el diagrama de terminales del microprocesador Pentium Pro, el cual viene encapsulado en un inmenso PGA (matriz de rejilla de terminales) de 387 terminales. Hasta el momento, el Pentium Pro está disponible en dos versiones. Una versión contiene una caché de nivel 2 de 256 K; la otra contiene una cache de nivel 2 de 512 K. La diferencia más notable en el diagrama de terminales del Pentium Pro, en comparación con el Pentium, es que hay provisiones para un bus de direcciones de 36 bits, el cual permite el acceso a 64 Gbytes de memoria. Esto es para uso futuro, ya que no hay un sistema que contenga algo parecido a esa cantidad de memoria.

Al igual que con las versiones más recientes del microprocesador Pentium, el Pentium Pro requiere una sola fuente de energía de +3.3 V o +2.7 V para su operación. La corriente de la fuente de energía es un máximo de 9.9 A para la versión de 150 MHz del Pentium Pro, que también tiene una disipación máxima de potencia de 26.7 W. Se requiere un buen disipador de calor con un flujo de aire considerable para mantener el Pentium Pro fresco. Al igual que el Pentium, el Pentium Pro contiene varias conexiones V_{CC} y V_{SS} que deben conectarse para su correcta operación. El Pentium Pro contiene terminales V_{CCP} (V_{CC} primario) que se conectan a +3.1 V, terminales V_{CCS} (V_{CC} secundario) que se conectan a +3.3 V y terminales V_{CC5} (V_{CC} estándar) que se conectan a +5.0 V. Algunas terminales se identifican como N/C (sin conexión) y no deben conectarse.

Cada una de las terminales de salida del Pentium Pro es capaz de suministrar 48.0 mA de corriente en un nivel de 0 lógico. Esto representa un considerable aumento en la corriente de control, en comparación con los 2.0 mA disponibles en las terminales de salida de los microprocesadores anteriores. Cada terminal de entrada representa una pequeña carga que requiere de sólo 15 µA de corriente. Debido a los 48.0 mA de corriente de control disponible en cada salida, sólo un sistema muy grande requiere búferes de bus.

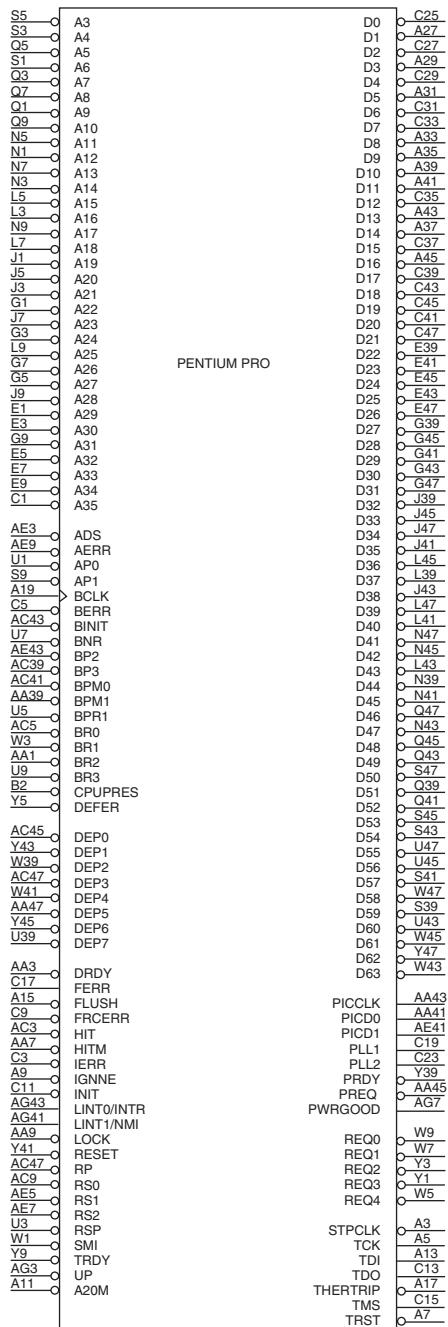
Estructura interna del Pentium Pro

La estructura del Pentium Pro es distinta a la de los primeros microprocesadores. Éstos contienen una unidad de ejecución y una unidad de interfaz de bus con una pequeña caché que actúa como búfer entre la unidad de ejecución y la unidad de interfaz de bus. En los siguientes microprocesadores se modificó esta estructura, pero sólo eran etapas adicionales dentro de los microprocesadores. La arquitectura del Pentium también es una modificación, pero más considerable que los microprocesadores anteriores. La figura 18-13 muestra un diagrama de bloques de la estructura interna del microprocesador Pentium Pro.

Los buses del sistema, que se comunican con la memoria y la E/S, se conectan a una caché de nivel 2 interna, que por lo general se encuentra en la tarjeta principal en la mayoría de los otros sistemas de microprocesadores. La caché de nivel 2 en el Pentium Pro es de 256 Kbytes o de 512 Kbytes. La integración de esta caché de nivel 2 agiliza el procesamiento y reduce el número de componentes en un sistema.

La unidad de interfaz de bus (BIU) controla el acceso a los buses del sistema, a través de la caché de nivel 2, como lo hace en la mayoría de los otros microprocesadores. De nuevo, la diferencia es que

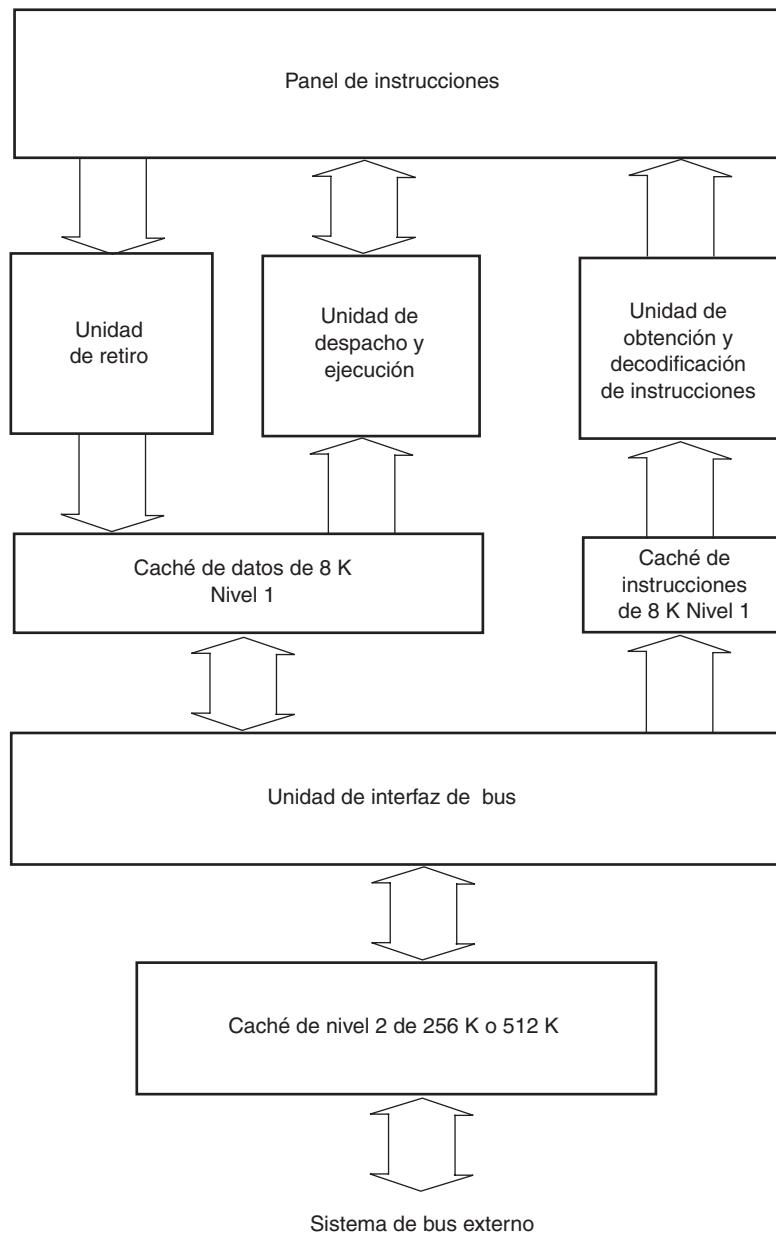
FIGURA 18-12 Diagrama de terminales del microprocesador Pentium Pro.



la caché de nivel 2 está integrada. La BIU genera la dirección de memoria y las señales de control; además, pasa y obtiene datos o instrucciones hacia/desde una caché de datos de nivel 1 o una caché de instrucciones de nivel 1. Cada una de estas cachés es de 8 Kbytes y tal vez se hagan más grandes en futuras versiones del microprocesador. Las versiones anteriores del microprocesador Intel contenían una caché unificada que almacenaba tanto instrucciones como datos. La implementación de cachés separadas mejora el rendimiento, ya que los programas que hacen un uso intensivo de los datos ya no llenan la caché sólo con datos.

La caché de instrucciones está conectada a la unidad de obtención y decodificación de instrucciones (IFDU). Aunque no se muestra, la IFDU contiene tres decodificadores de instrucciones separados,

FIGURA 18-13 La estructura interna del microprocesador Pentium Pro.

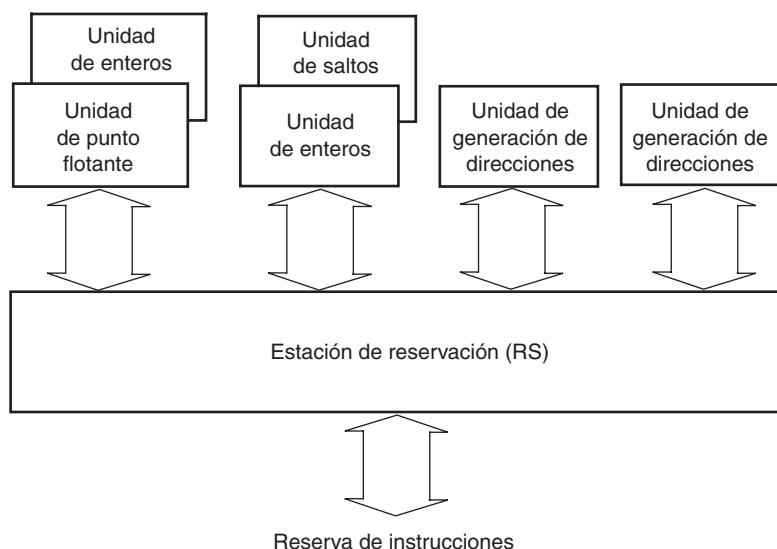


los cuales decodifican tres instrucciones de manera simultánea. Una vez decodificadas, las salidas de los tres decodificadores se pasan a la reserva de instrucciones, en donde permanecen hasta que la unidad de despacho y ejecución o la unidad de retiro las obtiene. Dentro de la IFDU también se incluye una sección de lógica de predicción de ramificación, la cual busca por adelantado en secuencias de código que contienen instrucciones de salto condicional. Si se localiza un salto condicional, la lógica de predicción de ramificación trata de determinar la siguiente instrucción en el flujo de un programa.

Una vez que las instrucciones decodificadas se pasan a la reserva de instrucciones, se retienen para su procesamiento. La reserva de instrucciones es una memoria que puede direccionarse por contenido, pero Intel nunca declara su tamaño en la literatura.

La unidad de despacho y ejecución (DEU) recupera las instrucciones decodificadas de la reserva de instrucciones cuando están completas y luego las ejecuta. En la figura 18-14 se muestra la estructura interna de la DEU. Observe que contiene tres unidades de ejecución de instrucciones: dos para procesar instrucciones con enteros y una para las instrucciones de punto flotante. Esto significa que el Pentium

FIGURA 18-14 La unidad de despacho y ejecución (DEU) del Pentium Pro.



Pro puede procesar dos instrucciones con enteros y una instrucción de punto flotante de manera simultánea. El Pentium también contiene tres unidades de ejecución, pero la arquitectura es distinta debido a que el Pentium no contiene una unidad de ejecución de saltos ni unidades de generación de direcciones, como el Pentium Pro. La estación de reservación (RS) logra programar hasta cinco eventos para su ejecución y procesar cuatro en forma paralela. Observe que hay dos componentes de la estación conectados a una de las unidades de generación de direcciones que no aparecen en la ilustración de la figura 18-14.

La última estructura interna del Pentium Pro es la unidad de retiro (RU). Esta unidad revisa la reserva de instrucciones y elimina las instrucciones decodificadas que ya se hayan ejecutado. La RU puede quitar tres instrucciones decodificadas por cada pulso de reloj.

Conexiones de las terminales

El número de terminales aumentó de 237 terminales en el Pentium a 387 en el Pentium Pro. A continuación se muestra una descripción de cada terminal o agrupación de terminales:

- A20M** La **máscara de dirección A20** es una entrada que se impone en el modo real para indicar al Pentium que debe realizar un envolvimiento de direcciones, como en el microprocesador 8086, para el uso del controlador HIMEM.SYS.
- A₃₅-A₃** Las conexiones del **bus de direcciones** direccionan cualquiera de las $8\text{ G} \times 64$ posiciones del sistema de memoria del Pentium Pro.
- ADS** El **estrobo de datos de dirección** se activa cada vez que el Pentium envía una dirección válida de memoria o de E/S.
- AP1, AP0** Las terminales **paridad de dirección** proporcionan una paridad par para la dirección de memoria en todas las transferencias de memoria y E/S iniciadas por el Pentium Pro. La salida **AP0** proporciona la paridad para las conexiones de dirección A₂₃-A₃ y la salida AP1 proporciona la paridad para las conexiones de dirección A₃₅-A₂₄.
- ASZ1, ASZ0** Las entradas **tamaño de dirección** se controlan para seleccionar el tamaño del acceso a la memoria. La tabla 18-6 muestra el tamaño del acceso a la memoria para los patrones de bits binarios en estas dos entradas para el Pentium Pro.
- BCLK** La terminal **reloj de bus** determina la frecuencia de operación del microprocesador Pentium Pro. Por ejemplo, si BLCK es de 66 MHz, se seleccionan diversas velocidades de reloj internas con base en los niveles lógicos que se aplican a las terminales en la tabla 18-7. Una frecuencia de BCLK de 66 MHz opera el bus del sistema a 66 MHz.

TABLA 18-6 Tamaño de memoria indicado por las terminales ASZ.

<u>ASZ1</u>	<u>ASZ0</u>	<i>Tamaño de memoria</i>
0	0	0-4 G
0	1	4 G-64 G
1	X	Reservado

TABLA 18-7 La señal BCLK y su efecto sobre la velocidad de reloj del Pentium.

<u>LINT1/NMI</u>	<u>LINT0/INTR</u>	<u>IGNNE</u>	<u>A20M</u>	<i>Proporción</i>	<i>BCLK=50Mhz</i>	<i>BCLK=66Mhz</i>
0	0	0	0	2	100 MHz	133 MHz
0	0	0	1	4	200 MHz	266 MHz
0	0	1	0	3	150 MHz	200 MHz
0	0	1	1	5	250 MHz	333 MHz
0	1	0	0	5/2	125 MHz	166 MHz
0	1	0	1	9/2	225 MHz	300 MHz
0	1	1	0	7/2	175 MHz	233 MHz
0	1	1	1	11/2	275 MHz	366 MHz
1	1	1	1	2	100 MHz	133 MHz

BERR	La entrada/salida error de bus indica un error de bus o se impone por un dispositivo externo para producir una interrupción de comprobación de equipo, o una interrupción no enmascarable.
BINIT	La señal inicialización de bus se activa cuando se enciende el equipo para inicializar el sistema de bus.
BNR	La señal bloqueo de siguiente petición se utiliza para detener el sistema cuando hay varios microprocesadores.
BP3-BP2	Las salidas estado de punto de interrupción indican el estado de los puntos de interrupción del Pentium Pro.
BPM1, BPM0	Las salidas monitor de punto de interrupción indican el estado de los puntos de interrupción y de los contadores programables.
BPR1	La peticIÓN de bus de agente de prioridad es una entrada que hace que el microprocesador cese las peticiones de bus.
BR3-BR0	Las entradas petición de bus permiten que coexistan hasta cuatro microprocesadores Pentium Pro en el mismo bus de sistema.
BREQ3-BREQ0	Las señales de petición de bus se utilizan para varios microprocesadores en el mismo bus de sistema.
D63-D0	Las conexiones del bus de datos transfieren datos tipo byte, palabra, doble palabra y palabra cuádruple entre el microprocesador y su sistema de memoria y de E/S.
DBSY	Bus de datos ocupado se impone para indicar que el bus de datos está ocupado con una transferencia de datos.
DEFER	La entrada aplazamiento se impone durante la fase de husmeo para indicar que no se es posible garantizar que la transacción se complete en orden.
DEN	La señal aplaza habilitación se dirige al bus en la segunda fase de una fase de petición.
DEP7-DEP0	Las señales de protección ECC del bus de datos proporcionan códigos de corrección de error para corregir un error de un solo bit y detectar un error de doble bit.
FERR	La señal error de punto flotante puede compararse con la línea ERROR en el 80386 y muestra que se ha producido un error en el coprocesador interno.

FLUSH	La entrada vaciado de caché hace que la caché vacíe todas las líneas de escritura aplazada y que invalide sus cachés internas. Si esta entrada es un 0 lógico durante una operación de reinicio, el Pentium Pro entra en su modo de prueba.
FRCERR	La señal error de comprobación de redundancia funcional se utiliza si dos microprocesadores Pentium Pro están configurados en un par.
HIT	La señal coincidencia indica que la caché interna contiene datos válidos en el modo de investigación.
HITM	La señal coincidencia modificada indica que el ciclo de investigación encontró una línea de caché modificada. Esta salida se utiliza para inhibir otras unidades maestras de manera que no accedan a los datos hasta que la línea de caché se escriba en la memoria.
IERR	La salida error interno muestra que el Pentium Pro ha detectado un error de paridad interno o un error de redundancia funcional.
IGNNE	La entrada ignora error numérico hace que el Pentium Pro ignore un error del co-procesador numérico.
INIT	La entrada inicialización efectúa un reinicio sin inicializar las cachés, los búferes de escritura aplazada y los registros de punto flotante. Esto no puede usarse para reiniciar el microprocesador en lugar de RESET después de encender el equipo.
INTR	La peticIÓN de interrupción es utilizada por circuitos externos para solicitar una interrupción.
LEN1, LEN0	Las señales de longitud (bits 0 y 1) indican el tamaño de la transferencia de datos, como se muestra en la tabla 18-8.
LINT1, LINT0	Las entradas de interrupción local funcionan como NMI e INTR; además, establecen la frecuencia divisoria del reloj en el reinicio.
LOCK	Esta señal se vuelve un 0 lógico cada vez que se coloca el prefijo LOCK: en una instrucción. Esto se utiliza con más frecuencia durante los accesos de DMA.
NMI	La señal interrupción no enmascarable solicita una interrupción no enmascarable, de igual forma que en las versiones anteriores del microprocesador.
PICCLK	La entrada de señal de reloj se utiliza para las transferencias sincrónicas de datos.
PICD	Los datos seriales de interfaz del procesador se utilizan para transferir mensajes seriales bidireccionales entre microprocesadores Pentium Pro.
PWRGOOD	La entrada energía apropiada se coloca en nivel de 1 lógico cuando la fuente de energía y el reloj se han estabilizado.
REQ4-REQ0	Las señales de petición (bits 0-4) definen el tipo de operación de transferencia de datos, como se muestra en las tablas 18-9 y 18-10.
RESET	La señal reinicio inicializa el Pentium Pro, con lo cual empieza a ejecutar software en la posición de memoria FFFFFFFFOH. El Pentium Pro se reinicia en el modo real y las 12 conexiones de dirección de más a la izquierda permanecen como 1s (FFFH) hasta que se ejecuta un salto lejano o una llamada lejana. Esto permite la compatibilidad con los microprocesadores anteriores.
RP	La señal peticIÓN de paridad proporciona un medio para solicitar que el Pentium Pro compruebe la paridad.

TABLA 18-8 Los bits de LEN y el tamaño de los datos.

<i>LEN1</i>	<i>LEN0</i>	<i>Tamaño de transferencia de dato</i>
0	0	0-8 bytes
0	1	16 bytes
1	0	32 bytes
1	1	Reservado

TABLA 18-9 Función de las entradas de petición en el primer pulso de reloj.

<i>REQ4</i>	<i>REQ3</i>	<i>REQ2</i>	<i>REQ1</i>	<i>REQ0</i>	Función
0	0	0	0	0	Respuesta aplazada.
0	0	0	0	1	Reservada.
0	1	0	0	0	Caso 1*.
0	1	0	0	1	Caso 2*.
1	0	0	0	0	Lectura de E/S.
1	0	0	0	1	Escritura de E/S.
X	X	0	1	0	Lectura de memoria.
X	X	0	1	1	Escritura en memoria.
X	X	1	0	0	Lectura de código en memoria.
X	X	1	1	0	Lectura de datos en memoria.
X	X	1	X	1	Escritura en memoria.

*En la tabla 18-10 podrá consultar el segundo pulso de reloj para los casos 1 y 2.

TABLA 18-10 Función de las entradas de petición para los casos 1 y 2.

Caso	<i>REQ4</i>	<i>REQ3</i>	<i>REQ2</i>	<i>REQ1</i>	<i>REQ0</i>	Función
1	X	X	X	0	0	Reconocimiento de interrupción.
1	X	X	X	0	1	Transacción especial.
1	X	X	X	1	X	Reservada.
2	X	X	X	0	0	Mensaje de rastreo de ramificación.
2	X	X	X	0	1	Reservada.
2	X	X	X	1	X	Reservada.

RS2-RS0 Las entradas **estado de respuesta** hacen que el Pentium Pro realice las funciones que se listan en la tabla 18-11.

RSP La entrada **paridad de respuesta** aplica una señal de error de paridad de un verificador de paridad externo.

SMI La entrada **interrupción de administración del sistema** hace que el Pentium Pro entre al modo de operación de administración del sistema.

SMMEM La señal **modo de administración de memoria del sistema** se vuelve un 0 lógico cada vez que el Pentium se ejecuta en el espacio de interrupciones y direcciones del modo de administración de memoria del sistema.

SPLCK La señal **bloqueo dividido** se coloca en el nivel de 0 lógico para indicar que la transferencia contendrá cuatro transacciones bloqueadas.

SPCLK La señal **detención de reloj** hace que el Pentium Pro entre al estado de apagado cuando se coloca en el nivel de 0 lógico.

TABLA 8-11 Operación de las entradas de estado de respuesta.

<i>RS2</i>	<i>RS1</i>	<i>RS0</i>	Función	<i>HITM</i>	<i>DEFER</i>
0	0	0	Estado inactivo	X	X
0	0	1	Reintenta	0	1
0	1	0	Aplaza	0	1
0	1	1	Reservada	0	1
1	0	0	Falla de hardware	X	X
1	0	1	Normal, sin datos	0	0
1	1	0	Escritura aplazada implícita	1	X
1	1	1	Normal, con datos	0	0

TCK	La entrada reloj de testabilidad selecciona la función del reloj de acuerdo con la interfaz de Exploración de límites IEEE 1149.1.
TDI	La entrada de datos de prueba se utiliza para probar los datos que se envían al Pentium Pro mediante la señal TCK.
TDO	La salida de datos de prueba se utiliza para recopilar los datos de prueba y las instrucciones que se desplazan fuera del Pentium mediante TCK.
TMS	La entrada selección de modo de prueba controla la operación del Pentium Pro en el modo de prueba.
TRDY	La entrada destino listo se impone cuando el destino está listo para una operación de transferencia de datos.

El sistema de memoria

El sistema de memoria para el microprocesador Pentium Pro es de 4 Gbytes, igual que en los microprocesadores 80386DX-Pentium, pero se hace posible el acceso a un área entre 4 G y 64 G gracias a las señales de dirección adicionales A₃₂-A₃₅. El Pentium Pro utiliza un bus de datos de 64 bits para direccionar la memoria organizada en ocho bancos, cada uno de los cuales contiene 8 Gbytes de datos. La memoria adicional se habilita con la posición de bit 5 de CR4 y sólo está accesible cuando se habilita la paginación de 2 M. Además, la paginación de 2 M es nueva para el Pentium Pro y permite acceder a la memoria que está por encima de los 4 Gbytes. Más adelante, en este capítulo, presentaremos información adicional sobre la paginación del Pentium Pro. En la figura 18-15 verá la organización del sistema de memoria física del Pentium Pro.

El sistema de memoria del Pentium Pro se divide en ocho bancos, que almacenan cada uno un byte de datos con un bit de paridad. La mayoría de los sistemas basados en los microprocesadores Pentium y Pentium Pro no utilizan el bit de paridad. Al igual que el 80486, el Pentium Pro emplea la generación de paridad y la lógica de comprobación de manera interna para la información del bus de datos del sistema de memoria. La memoria de 64 bits es importante para los datos de punto flotante con doble precisión. Recuerde que un número de punto flotante con doble precisión es de 64 bits. Al igual que con los microprocesadores Intel anteriores, el sistema de memoria se enumera en bytes, desde el byte 00000000H hasta el byte FFFFFFFFH. Esta dirección hexadecimal de nueve dígitos se emplea en un sistema que direcciona 64 G de memoria.

La selección de memoria se lleva a cabo mediante las señales de habilitación de banco (\overline{BE}_7 - \overline{BE}_0). En el microprocesador Pentium Pro, las señales de habilitación de banco están presentes en el bus de direcciones (A₁₅-A₈) durante el segundo ciclo de reloj de un acceso a memoria o de E/S. Estas señales deben extraerse del bus de direcciones para acceder a los bancos de memoria. Estos bancos de memoria separados permiten que el Pentium Pro tenga acceso a cualquier byte individual, palabra,

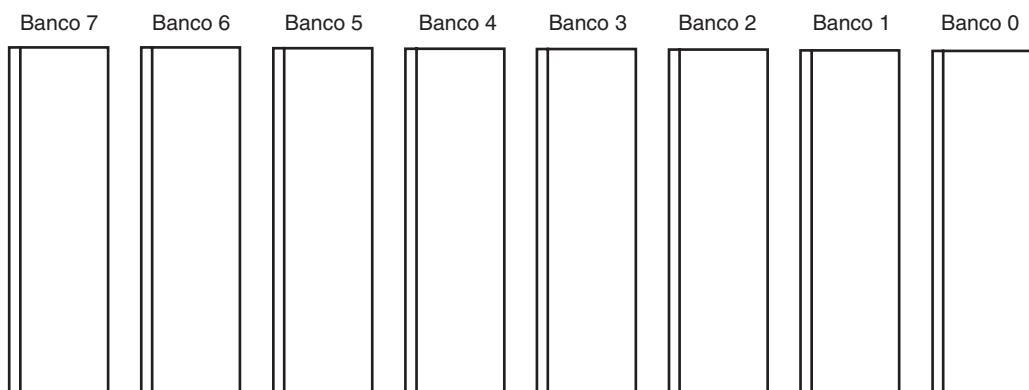


FIGURA 18-15 Los ocho bancos de memoria en el sistema Pentium Pro. Cada banco es de 8 bits de ancho y de 8 G de largo si se habilita el direccionamiento de 36 bits.

doble palabra o palabra cuádruple con un solo ciclo de transferencia de memoria. Al igual que con la lógica de selección de memoria anterior, se generan ocho estrobos de escritura separados para escribir en el sistema de memoria. La información de escritura en memoria se proporciona en las líneas de petición del microprocesador durante la segunda fase de reloj de un acceso a memoria o a la E/S.

Una nueva característica que se agregó al Pentium y al Pentium Pro es la capacidad de comprobar y generar la paridad para el bus de dirección durante ciertas operaciones. La terminal AP (Pentium) o las terminales AP1 y APO (Pentium Pro) proporcionan la información de paridad al sistema, en tanto que la terminal APCHK (Pentium) o las terminales AP (Pentium Pro) señalan una comprobación de paridad errónea para el bus de direcciones. El Pentium Pro no realiza ninguna acción cuando se detecta un error de paridad en la dirección. El sistema debe hacerse cargo del error tomando la acción apropiada (una interrupción), si así lo desea.

Una novedad en el Pentium Pro es el circuito de corrección de errores integrado (ECC), el cual permite corregir un error de un bit y detectar un error de dos bits. Para lograr tales detección y corrección de errores, el sistema de memoria debe tener espacio para un número adicional de 8 bits, que se almacena con cada número de 64 bits. Los ocho bits adicionales se utilizan para almacenar un código de corrección de errores que permite al Pentium Pro corregir cualquier error de un solo bit en forma automática. Una SDRAM de $1\text{ M} \times 64$ es una SDRAM de 64 M sin ECC y una SDRAM de $1\text{ M} \times 72$ es una SDRAM con soporte ECC. El código ECC es mucho más confiable que el antiguo esquema de la paridad, que se utiliza en muy raras ocasiones en los sistemas modernos. La única desventaja del esquema ECC es el costo adicional de la SDRAM de 72 bits.

Sistema de entrada/salida

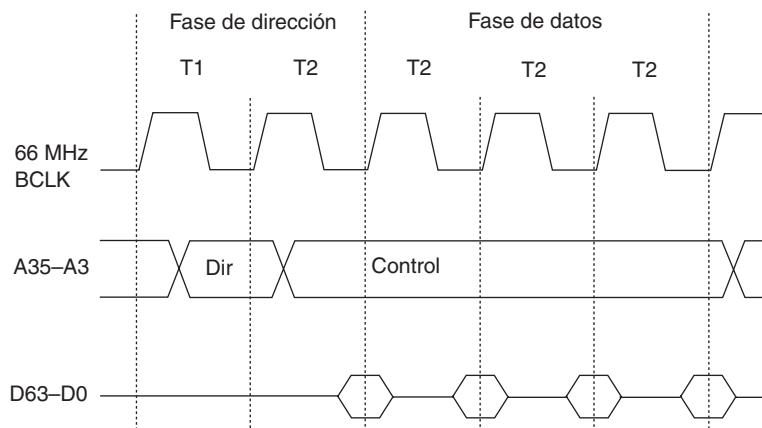
El sistema de entrada/salida del Pentium Pro es 100% compatible con los microprocesadores Intel anteriores. El número de puerto de E/S aparece en las líneas de dirección $A_{15}-A_3$ y las señales de habilitación de banco se utilizan para seleccionar los bancos de memoria actuales que se utilizan para la transferencia de E/S.

Sincronización del sistema

Al igual que con cualquier microprocesador, hay que comprender el funcionamiento de las señales de sincronización del sistema para conectar el microprocesador. En esta parte del libro hablaremos con detalle sobre la operación del Pentium Pro a través de sus diagramas de sincronización y mostraremos cómo determinar los tiempos de acceso a la memoria.

El ciclo de memoria básico del Pentium Pro consiste en dos secciones: la fase de dirección y la fase de datos. Durante la fase de dirección, el Pentium Pro envía la dirección (T_1) al sistema de memoria y de E/S, junto con las señales de control (T_2). Las señales de control incluyen las líneas ATTR ($A_{31}-A_{24}$), las líneas DID ($A_{23}-A_{16}$), las señales de habilitación de banco ($A_{15}-A_8$) y las líneas EXF (A_7-A_3). En la figura 18-16 podrá consultar el ciclo de sincronización básico. El tipo de ciclo de memoria aparece

FIGURA 18-16 Ciclo de sincronización básico del Pentium Pro.



en las terminales de petición. Durante la fase de datos, se obtienen o se escriben en memoria cuatro números de 64 bits. Esta operación es muy común, ya que los datos de la memoria principal se transfieren entre la caché de escritura aplazada interna de 256 K o 512 K y el sistema de memoria. Las operaciones que escriben un byte, una palabra o una doble palabra, tales como las transferencias de E/S, utilizan las señales de selección de banco y sólo tienen un ciclo de reloj en la fase de transferencia de datos. En el diagrama de sincronización observamos que el Pentium Pro de 66 MHz es capaz de realizar 33 millones de transferencias por segundo (esto si suponemos que la memoria logra operar a esa velocidad).

El tiempo de preparación antes del ciclo de reloj se da como 5.0 ns y el tiempo de retención después del ciclo de reloj se da como 1.5 ns. Esto significa que la ventana de datos alrededor del reloj es de 6.5 ns. La dirección aparece en el máximo de 8.0 ns después del inicio de T_1 . Esto significa que el microprocesador Pentium Pro que opera a 66 MHz permite 30 ns (dos períodos de reloj) menos el tiempo de retraso de dirección de 8.0 ns y menos el tiempo de preparación de los datos de 5.0 ns. El tiempo de acceso a memoria sin usar estados de espera sería de $30 - 8.0 - 5.0 = 17.0$ ns. Esto es tiempo suficiente para permitir el acceso a una SRAM, pero no para cualquier DRAM sin insertar estados de espera en la sincronización.

Los estados de espera se insertan en la sincronización mediante el control de la entrada **TRDY** que va hacia el Pentium Pro. Esta señal debe volverse un 0 lógico antes de que termine T_2 , o se insertarán estados de espera T_2 adicionales en la sincronización. Tenga en cuenta que la DRAM de 60 ns requiere la inserción de cuatro estados de espera de 15 ns (un período de reloj) cada uno para extender el tiempo de acceso a 77 ns. Esto es tiempo suficiente para que funcione la DRAM y cualquier decodificador en el sistema. Como muchas EPROM o dispositivos de memoria Flash requieren un tiempo de acceso de 100 ns, hay que agregar siete estados de espera para extender el tiempo de acceso hasta 122 ns.

18-6

CARACTERÍSTICAS ESPECIALES DEL PENTIUM PRO

En esencia, el Pentium Pro es el mismo microprocesador que el 80386, el 80486 y el Pentium, sólo que se han agregado ciertas características y modificaciones en el conjunto de registros de control. En esta sección resaltaremos las diferencias entre la estructura de los registros de control del 80386 y el registro de control del Pentium Pro.

Registro de control 4

La figura 18-17 muestra el registro de control 4 del microprocesador Pentium Pro. Observe que CR₄ tiene dos nuevos bits de control que se agregaron al arreglo de registros de control.

En esta sección explicaremos sólo los dos nuevos componentes del Pentium Pro en el registro de control 4. (En la figura 18-8 verá una descripción y una ilustración de los registros de control del Pentium.) A continuación se muestra una descripción de los bits del registro CR₄ del Pentium y los nuevos bits de control del Pentium Pro en el registro de control CR₄:

- | | |
|------------|---|
| VME | El bit de extensión de modo virtual habilita el soporte para la bandera de interrupciones virtuales en el modo protegido. Si VME = 0, se deshabilita el soporte de interrupciones virtuales. |
| PVI | El bit de interrupción virtual en modo protegido habilita el soporte para la bandera de interrupciones virtuales en el modo protegido. |
| TSD | El bit deshabilita estampa de tiempo controla la instrucción RDTSC. |
| DE | El bit extensión de depuración habilita las extensiones de depuración de puntos de interrupción de E/S cuando se activa. |

FIGURA 18-17 El nuevo registro de control 4 (CR₄) en el microprocesador Pentium Pro.

31	7	6	5	4	3	2	1	0
	PGE	MCE	PAE	PSE	DE	TSD	PVI	VME

PSE	El bit extensión de tamaño de página habilita las páginas de memoria de 4 Mbytes cuando se activa en el Pentium o las páginas de 2 Mbytes cuando se activa en el Pentium Pro, siempre y cuando también se active PSE.
PAE	El bit extensión de dirección de página habilita las líneas de dirección A ₃₅ -A ₃₂ cada vez que se habilita para el Pentium Pro un nuevo modo de direccionamiento especial, controlado por PGE.
MCE	El bit habilita comprobación de equipo habilita la interrupción de comprobación de equipo.
PGE	El bit extensión de páginas controla el nuevo modo de direccionamiento más extenso de 64 G, cada vez que se activa junto con PAE y PSE.

18-7**RESUMEN**

1. El microprocesador Pentium es casi idéntico a los microprocesadores 80386 y 80486. La principal diferencia es que el Pentium se ha modificado en su interior para que contenga una caché dual (instrucciones y datos) y una unidad de enteros dual. El Pentium también opera a una velocidad de reloj más alta de 66 MHz.
2. El Pentium de 66 MHz requiere 3.3A de corriente y la versión de 60 MHz requiere 2.91 A. La fuente de energía debe ser de +5.0 V con una regulación de $\pm 5\%$. Las versiones más recientes del Pentium requieren una fuente de energía de 3.3 V o de 2.7 V.
3. El bus de datos en el Pentium es de 64 bits y contiene ocho bancos de memoria de un byte cada uno, los cuales se seleccionan mediante las señales de habilitación de banco (\overline{BE}_7 - \overline{BE}_0).
4. El tiempo de acceso a la memoria, sin estados de espera, es de tan sólo 18 ns en el Pentium de 66 MHz. En muchos casos, este tiempo de acceso tan corto requiere estados de espera que se introducen mediante el control de la entrada \overline{BRDY} que va al Pentium.
5. La estructura superescalar del Pentium contiene tres unidades de procesamiento independientes: un procesador de punto flotante y dos unidades de procesamiento de enteros que son identificados como U y V por Intel.
6. La estructura de la caché del Pentium se modificó para incluir dos cachés. Una caché de 8 K \times 8 fue diseñada como caché de instrucciones; la otra caché de 8 K \times 8 es una caché de datos. La caché de datos puede operarse ya sea como caché de escritura directa o de escritura aplazada.
7. Se ha agregado al Pentium un nuevo modo de operación llamado modo de administración de memoria de sistema (SMM). Para acceder al modo SMM se utiliza la interrupción de administración de memoria, la cual se aplica a la terminal de entrada SMI. En respuesta a SMI, el Pentium comienza a ejecutar el software que se encuentra en la posición de memoria 38000H.
8. Las nuevas instrucciones son CMPXCHG8B, RSM, RDMSR, WRMSR y CPUID. La instrucción CMPXCHG8B es similar a la instrucción CMPXCHG del 80486. La instrucción RSM regresa de la interrupción de administración de memoria del sistema. Las instrucciones RDMSR y WRMSR leen o escriben en los registros específicos del equipo. La instrucción CPUID lee el código de identificación de CPU del Pentium.
9. La autoprueba integrada (BIST) permite probar el Pentium cuando se aplica energía por primera vez al sistema. Un reinicio por encendido normal activa la entrada RESET que va al Pentium. Un reinicio por encendido con BIST activa INIT y luego desactiva la terminal RESET. EAX es igual a 0000000H cuando el Pentium pasa la prueba de la BIST.
10. Una nueva modificación propietaria de Intel a la unidad de paginación permite el uso de páginas de 4 Mbytes de memoria en lugar de las páginas de 4 Kbytes. Esto se logra mediante el uso del directorio de páginas para direccionar 1024 tablas de páginas, cada una de las cuales contiene 4 Mbytes de memoria.
11. El Pentium Pro es una versión mejorada del microprocesador Pentium, la cual no sólo contiene las cachés de nivel 1 que incluye el Pentium, sino que también incluye una caché de nivel 2 de 256 K o 512 Kbytes, la cual se encuentra en la mayoría de las tarjetas principales (o tarjetas madre).

12. El Pentium Pro opera mediante el uso de la misma velocidad de bus de 66 MHz que el Pentium y el 80486. Utiliza un generador de reloj interno para multiplicar la velocidad del bus por varios factores para obtener velocidades de ejecución internas más altas.
13. La única diferencia considerable de software entre el Pentium Pro y los microprocesadores anteriores es la adición de las instrucciones FCMOV y CMOV.
14. La única diferencia de hardware entre el Pentium Pro y los microprocesadores anteriores es la adición de la paginación de 2 M y cuatro líneas de dirección adicionales que permiten el acceso a un espacio de direcciones de memoria de 64 Gbytes.
15. Se ha agregado código de corrección de errores al Pentium Pro, el cual corrige cualquier error de un solo bit y detecta cualquier error de dos bits.

18-8**PREGUNTAS Y PROBLEMAS**

1. ¿Cuánta memoria está accesible para el microprocesador Pentium?
2. ¿Cuánta memoria está accesible para el microprocesador Pentium Pro?
3. La anchura del bus de datos de memoria es de _____ en el Pentium.
4. ¿Cuál es el propósito de las terminales DP₀-DP₇ en el Pentium?
5. Si el Pentium opera a 66 MHz, ¿qué frecuencia de señal de reloj se aplica a la terminal CLK?
6. ¿Cuál es el propósito de la terminal BRDY en el Pentium?
7. ¿Cuál es el propósito de la terminal AP en el Pentium?
8. ¿Cuánto tiempo de acceso a memoria permite el Pentium, sin estados de espera, cuando se opera a 66 MHz?
9. ¿Qué terminal del Pentium se utiliza para insertar estados de espera en la sincronización?
10. Un estado de espera es un periodo de reloj _____ adicional.
11. Explique cómo dos unidades de enteros permiten que el Pentium ejecute dos instrucciones no dependientes en forma simultánea.
12. ¿Cuántas cachés tiene el Pentium y cuáles son sus tamaños?
13. ¿Qué tan amplia es la ventana de muestreo de datos de memoria del Pentium para una operación de lectura de memoria?
14. ¿Puede el Pentium ejecutar tres instrucciones en forma simultánea?
15. ¿Cuál es el propósito de la terminal SMI?
16. ¿Qué es el modo de operación de administración de memoria del sistema para el Pentium?
17. ¿Cómo se puede salir del modo de administración de memoria del sistema?
18. ¿En dónde comienza el Pentium a ejecutar software para una entrada de interrupción SMI?
19. ¿Cómo puede modificarse la dirección de vaciado de la unidad de administración de memoria del sistema?
20. Explique la operación de la instrucción CMPXCHG8B.
21. ¿Qué información se devuelve en el registro EAX después de ejecutar la instrucción CPUID con un valor inicial de 0 en EAX?
22. ¿Cuáles son los nuevos bits de bandera que se agregaron al microprocesador Pentium?
23. ¿Qué nuevo registro de control se agregó al microprocesador Pentium?
24. Describa la manera en que el Pentium accede a páginas de 4 M.
25. Explique cómo funciona el contador de estampa de tiempo y cómo puede usarse para medir eventos.
26. Compare el microprocesador Pentium con el Pentium Pro.
27. ¿En dónde se encuentran las señales de habilitación de banco en el microprocesador Pentium Pro?
28. ¿Cuántas líneas de dirección se encuentran en el sistema Pentium Pro?
29. ¿Qué modificaciones se han realizado a CR₄ en el Pentium Pro y para qué fin?
30. Compare los tiempos de acceso en el sistema Pentium con los del sistema Pentium Pro.
31. ¿Qué significa ECC?
32. ¿Qué tipo de SDRAM debe comprarse para utilizar ECC?

CAPÍTULO 19

Los microprocesadores Pentium II, Pentium III y Pentium 4

INTRODUCCIÓN

Es muy probable que los microprocesadores Pentium II, Pentium III y Pentium 4 marquen el fin de la evolución de la arquitectura de 32 bits con la llegada de los microprocesadores Itanium¹ e Itanium II de Intel. El Itanium es un microprocesador con arquitectura de 64 bits. Las arquitecturas Pentium II, Pentium III y Pentium 4 son extensiones de la arquitectura Pentium Pro, con algunas diferencias. La más notable es que en el Pentium II se separó del microprocesador la caché interna de la arquitectura del Pentium Pro. Otro de los cambios importantes es que el Pentium II no está disponible en forma de circuito integrado, sino que viene en una pequeña tarjeta de circuitos insertable (cartucho) junto con el chip de caché nivel 2. Hay varias versiones del Pentium II disponibles. El Celeron² es una versión del Pentium II que no contiene la caché de nivel 2 en la tarjeta de circuitos del Pentium II. El Xeon³ es una versión mejorada del Pentium II que contiene una caché de hasta 2 Mbytes en la tarjeta de circuitos.

Al igual que el Pentium II, los primeros microprocesadores Pentium III venían empaquetados en un cartucho, en lugar de en un circuito integrado. Las versiones más recientes, como el Coppermine, se empaquetan de nuevo en un circuito integrado (370 terminales). Al igual que el Pentium Pro, el Pentium III Coppermine contiene una caché interna. El Pentium 4 viene empaquetado en un circuito integrado más grande, con 423 o 478 terminales. El Pentium 4 también utiliza transistores cuyo tamaño físico es más pequeño, lo cual lo hace mucho más pequeño y rápido que el Pentium III. A la fecha, Intel ha sacado al mercado versiones del Pentium 4 que operan a frecuencias mayores de 3 GHz, con un límite de tal vez 10 GHz en algún momento en el futuro. El modelo Extreme (P4E) es un modelo especial del Pentium 4 con una caché de 1 Mbyte. También está disponible el modelo Pentium 4 Extreme Edition (P4EE), con una caché de 2 Mbytes. Estas dos versiones (P4E y P4EE) están disponibles en el nuevo formato de 90 mm (0.09 micrones), en comparación con los microprocesadores P4 anteriores, que utilizan el formato de 0.13 micrones.

OBJETIVOS DEL CAPÍTULO

Al terminar este capítulo podrá:

1. Explicar con detalle las diferencias entre los microprocesadores Pentium II, Pentium III y Pentium 4 y los anteriores microprocesadores Intel.
2. Explicar cómo las arquitecturas del Pentium II, Pentium III y Pentium 4 mejoran la velocidad del sistema.

¹ Itanium es una marca registrada de Intel Corporation.

² Celeron es una marca registrada de Intel Corporation.

³ Xeon es una marca registrada de Intel Corporation.

3. Explicar cómo ha cambiado la arquitectura básica del sistema computacional mediante el uso de los microprocesadores Pentium II, Pentium III y Pentium 4.
4. Explicar con detalle las modificaciones a la instrucción CPUID y los registros específicos para el modelo.
5. Describir la operación de las instrucciones SYSENTER y SYSEXIT.
6. Describir la operación de las instrucciones FXSAVE y FXRSTOR.

19-1**INTRODUCCIÓN AL MICROPROCESADOR PENTIUM II**

Antes de utilizar el Pentium II o cualquier otro microprocesador en un sistema, se debe comprender la función de cada una de las terminales. En esta sección hablaremos con detalle sobre la operación de cada terminal, junto con las estructuras del sistema de memoria externa y de E/S del microprocesador Pentium II.

La figura 19-1 muestra el esquema básico de la ranura 1 (slot 1) conector del microprocesador Pentium II y la señales que se utilizan para conectarse con el conjunto de chips. La figura 19-2 muestra un diagrama simplificado de los componentes en el cartucho del Pentium II y su colocación junto con los componentes del bus en el sistema Pentium II común. Hay 242 terminales en la ranura 1 conector del microprocesador. (Ésta es una reducción en comparación con el número de terminales en los microprocesadores Pentium y Pentium II.) El Pentium II viene empaquetado en una tarjeta de circuitos impresos, en lugar de los circuitos integrados de los microprocesadores Intel anteriores. La caché de nivel 1 es de 32 Kbytes como en el Pentium Pro, pero la caché de nivel 2 ya no está dentro del circuito integrado. Intel modificó la arquitectura de manera que pudiera colocarse una caché de nivel 2 muy cerca del microprocesador. Este cambio reduce el precio del microprocesador y además la caché de nivel 2 sigue operando con eficiencia. Esta caché opera a la mitad de la frecuencia del reloj del microprocesador, en lugar de los 66 MHz del microprocesador Pentium. La caché de un Pentium II de 400 MHz tiene una velocidad de 200 MHz. El Pentium II está disponible en tres versiones. La primera es el Pentium II verdadero, que viene siendo el Pentium II para la ranura 1 tipo conector. La segunda es el Celeron, que es como el Pentium II, sólo que la ranura 1 tipo tarjeta de circuitos no contiene una caché de nivel 2; ésta se encuentra en la tarjeta principal y opera a 66 MHz. La versión más reciente es el Xeon que, como usa una caché de nivel 2 de 512 K, 1 M o 2 M, representa una mejora considerable en velocidad, en comparación con el Pentium II. La caché de nivel 2 del Xeon opera a la frecuencia del reloj del microprocesador. La caché de nivel 2 de un Xeon de 400 MHz tiene una velocidad de caché de 400 MHz, que es el doble de la velocidad del Pentium II normal.

Las primeras versiones del Pentium II requieren una fuente de energía de 5.0 V, 3.3V y de voltaje variable para su operación. Los voltajes principales de la fuente de energía varían desde 3.5V hasta un valor mínimo de 1.8 V en el microprocesador. A éste se le conoce como el voltaje básico del microprocesador. La corriente de la fuente de energía varía en un promedio de 14.2 A a 8.4 A, dependiendo de la frecuencia de operación y del voltaje del Pentium II. Como estas corrientes son considerables, también lo es la disipación de potencia de estos microprocesadores. Hoy se requiere un buen disipador de calor con un flujo de aire considerable para que el Pentium II se mantenga fresco. Por fortuna, el disipador de calor y el ventilador están integrados en el cartucho del Pentium II. Las versiones más recientes de este microprocesador se han mejorado para reducir la disipación de potencia.

Cada una de las terminales de salida del cartucho del Pentium II es capaz de suministrar, cuando menos, 36 mA de corriente a un nivel de 0 lógico en las conexiones de las señales. Algunas de las señales de control de salida sólo suministran 14 mA de corriente. Otra de las modificaciones en el Pentium II es que las salidas son de colector abierto y requieren una resistencia elevadora externa para operar en forma apropiada.

A continuación se muestra la función de cada grupo de terminales del Pentium II:

A20

La **máscara de dirección A20** es una entrada que se impone en el modo real para indicar al Pentium II que debe realizar un envolvimiento de direcciones, como en el microprocesador 8086, para el uso del controlador HIMEM.SYS.

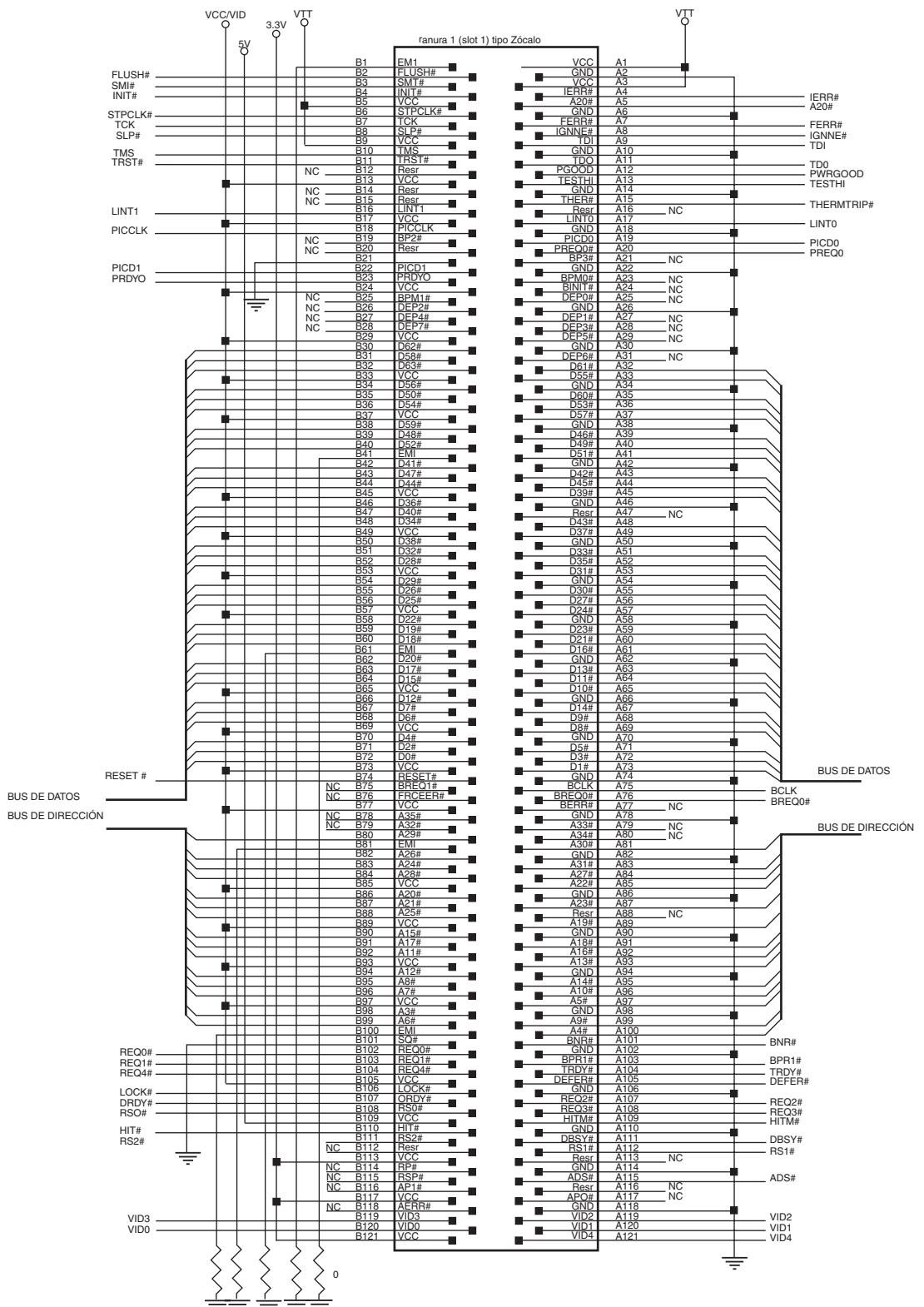
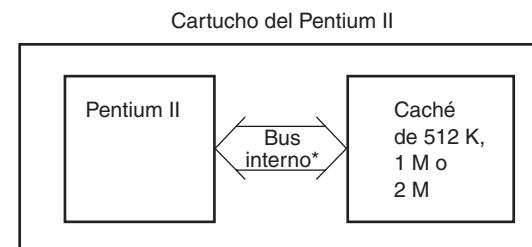
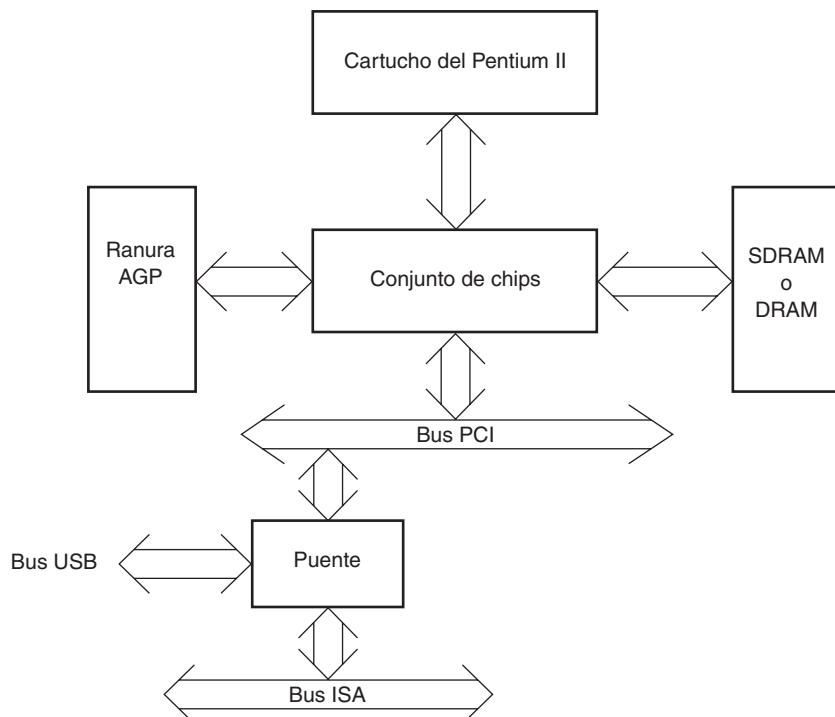


FIGURA 19-1 Diagrama de terminales de la ranura 1 (slot 1) tipo conector, en el que se muestran las conexiones para el sistema.

FIGURA 19-2 La estructura del cartucho del Pentium II y la estructura del sistema Pentium II.



* La velocidad del bus es 1/2 la velocidad del Pentium o la misma velocidad del Pentium en el Xeon.



A35-A3

Las conexiones del **bus de direcciones**, que son conexiones activas en nivel bajo, direccionan cualquiera de las posiciones del sistema de memoria del Pentium II. Observe que A_0 , A_1 y A_2 se codifican en las señales de habilitación del bus ($\overline{BE7-BE0}$) que genera el conjunto de chips para seleccionar cualquiera de los ocho bytes en una posición de memoria de 64 bits.

ADS

El **estrobo de datos de dirección** es una entrada que se activa para indicar al Pentium II que el sistema está listo para realizar una operación de memoria o de E/S. Esta señal hace que el microprocesador proporcione la dirección al sistema.

AERR

La señal **error de dirección** es una entrada que se utiliza para hacer que el Pentium II compruebe un error de paridad de dirección, si está activada.

AP1, AP0

Las entradas **paridad de dirección** indican un error en la paridad de la dirección.

BCLK

La señal **reloj del bus** es una entrada que establece la frecuencia del reloj del bus. En el Pentium II es de 66 MHz o de 100 MHz.

BERR

La señal **error de bus** se impone para indicar que ha ocurrido un error en el bus del sistema.

BINIT	La señal inicialización de bus es un 0 lógico durante el reinicio o la inicialización del sistema. Es una entrada para indicar que ha ocurrido un error de bus y que el sistema necesita reinicializarse.
BNR	La señal bus no está listo es una entrada que se utiliza para insertar estados de espera en la sincronización para el Pentium II. Si se coloca un 0 lógico en esta entrada, el Pentium II entrará en estados de suspensión (stall) o de espera.
BP3, BP2 PM1/BP1 y PM0/BP0	Las terminales punto de interrupción $\overline{BP3-BP0}$ indican una coincidencia de punto de interrupción cuando los registros de depuración se programan para monitorear las coincidencias. Las terminales monitoreo de rendimiento PM1 y PM0 indican la configuración de los bits de rendimiento en el registro de control de modo de depuración.
BPRI	La entrada de petición de prioridad del bus se utiliza para solicitar el bus del sistema al Pentium II.
BR1 y BR0	Las señales petición de bus indican que el Pentium II ha generado una petición del bus. Durante la inicialización, debe activarse la terminal $\overline{BR0}$.
BSEL	Hasta hoy, el Pentium II no ha utilizado la señal selección de bus ; debe conectarse a tierra para que el Pentium II opere en forma apropiada.
D63-D0	Las conexiones del bus de datos transfieren datos tipo byte, palabra, doble palabra y palabra cuádruple entre el microprocesador y su sistema de memoria y de E/S.
DEFER EP7-EP0	La señal aplaza indica que el sistema externo no puede completar el ciclo del bus. Las terminales ECC de datos se utilizan en el esquema de corrección de errores del Pentium II y por lo general se conectan a una sección de memoria de 8 bits adicionales. Esto significa que los módulos de memoria ECC son de 72 bits, en lugar de 64 bits.
DRDY	La señal datos listos se activa para indicar que el sistema presenta datos válidos al Pentium II.
EMI	La señal interferencia electromagnética debe aterrizar para evitar que el Pentium II genere o reciba ruido.
FERR	La señal error de punto flotante se puede comparar con la línea ERROR en el 80386 y muestra que el coprocesador interno tuvo un error.
FLUSH	La entrada vaciado de caché hace que la caché vacíe todas las líneas de escritura aplazada e invalide sus cachés internas. Si la entrada \overline{FLUSH} es un 0 lógico durante una operación de reinicio, el Pentium entra en su modo de prueba.
FRCERR	La señal comprobación de redundancia funcional se muestrea durante un reinicio para configurar el Pentium II en los modos maestro (1) o verificador (0).
HIT	La señal coincidencia indica que la caché interna contiene datos válidos en el modo de investigación (o exploración).
HITM	La señal coincidencia modificada indica que el ciclo de investigación encontró una línea de caché modificada. Esta salida se utiliza para inhibir otras unidades maestras de manera que no accedan a los datos hasta que la línea de caché se escriba en la memoria.
IERR	La salida error interno muestra que el Pentium II ha detectado un error interno o un error de redundancia funcional.
IGNNE	La entrada ignora error numérico hace que el Pentium II ignore un error del coprocesador numérico.
INIT	La entrada inicialización efectúa un reinicio sin inicializar las cachés, los búferes de escritura aplazada y los registros de punto flotante. Esta entrada no puede usarse para reiniciar el microprocesador en lugar de RESET, después de encender el equipo.
INTR	La petición de interrupción es utilizada por circuitos externos para solicitar una interrupción.

LINT₁, LINT₀	Las señales interrupción APIC local deben conectarse a las terminales apropiadas de todos los agentes de bus APIC. Cuando APIC se deshabilita, la señal LINT ₀ se convierte en INTR, una señal de petición de interrupción enmascarable; LINT ₁ se convierte en NMI, una interrupción no enmascarable.
LOCK	Esta señal se vuelve un 0 lógico cada vez que se coloca el prefijo LOCK en una instrucción. Su uso más común es durante los accesos DMA.
NMI	La señal interrupción no enmascarable solicita una interrupción no enmascarable, de igual forma que en las versiones anteriores del microprocesador.
PICCLK	Esta señal de reloj debe ser $\frac{1}{4}$ de la frecuencia de $\overline{\text{BCLK}}$.
PICD₁, PICD₀	Se utiliza para los mensajes en serie entre el Pentium II y APIC.
PRDY	La salida sonda lista indica que se ha entrado al modo de sonda para la depuración.
PREQ	La señal petición de sonda se utiliza para solicitar la depuración.
PWRGOOD	La entrada energía adecuada indica que la fuente de energía del sistema es operacional.
REQ4-REQ0	Las señales de petición comunican comandos entre los controladores de bus y el Pentium II.
RESET	La señal reinicio inicializa el Pentium II, con lo cual empieza a ejecutar software en las posiciones de memoria FFFFFFFF0H o 000FFFF0H. Los bits de dirección A ₃₅ -A ₃₂ se establecen como 0 lógicos durante la operación de reinicio. El Pentium II se reinicia en el modo real y las 12 conexiones de dirección de más a la izquierda permanecen como 1 lógicos (FFFH) hasta que se ejecuta un salto lejano o una llamada lejana. Esto permite la compatibilidad con los microprocesadores anteriores. Consulte en la tabla 19-1 el estado del Pentium II después de un reinicio de hardware.
RP	La entrada petición de paridad se utiliza para solicitar paridad.
RS2-RS0	Las entradas petición de estado se utilizan para solicitar el estado actual del Pentium II.
RSP	La entrada paridad de respuesta se activa para solicitar la paridad.
SLOTOCC	La salida ranura ocupada es un 0 lógico si la ranura cero contiene un Pentium II o un terminador falso.

TABLA 19-1 Estado del Pentium II después de un reinicio.

Registro	Reinicio	Reinicio + BIST
EAX	0	0 (si pasa la prueba)
EDX	0500XXXXH	0500XXXXH
EBX, ECX, ESP, EBP, ESI y EDI	0	0
EFLAGS	2	2
EIP	0000FFF0H	0000FFF0H
CS	F000H	F000H
DS, ES, FS, GS y SS	0	0
GDTR y TSS	0	0
CR ₀	60000010H	60000010H
CR ₂ , CR ₃ y CR ₄	0	0
DR ₃ -DR ₀	0	0
DR ₆	FFFF0FF0H	FFFF0FF0H
DR ₇	00000400H	00000400H

Notas: BIST = autoprueba integrada, XXXX = número de versión de Pentium II.

SLP	Cuando la entrada suspensión (sleep) se inserta en el estado de stop-grant, el Pentium II entra en el estado de suspensión.
SMI	La entrada interrupción de administración del sistema hace que el Pentium II entre al modo de operación de administración del sistema.
STPCLK	La entrada detener reloj hace que el Pentium II entre en el estado stop-grant de bajo consumo de energía.
TCK	La entrada reloj de testabilidad selecciona la función del reloj de acuerdo con la interfaz de exploración de límites IEEE 1149.1.
TDI	La entrada de datos de prueba se utiliza para probar los datos que se envían al Pentium II mediante la señal TCK.
TDO	La salida de datos de prueba se utiliza para recopilar los datos de prueba y las instrucciones que se desplazan fuera del Pentium II mediante TCK.
TESTHI	La entrada prueba en nivel alto debe conectarse a +2.5V por medio de una resistencia de 1 K-10 KΩ para que el Pentium II opere en forma apropiada.
THERMTRIP	La salida activa sensor término se vuelve cero cuando la temperatura del Pentium II excede los 130 °C.
TMS	La entrada selección de modo de prueba controla la operación del Pentium II en modo de prueba.
TRDY	La señal destino listo es una entrada que se utiliza para hacer que el Pentium II realice una operación de escritura aplazada.
VID4-VID0	Las terminales de salida datos de voltaje son señales abiertas o aterrizadas que indican el voltaje de suministro que requiere el Pentium II en ese momento. La fuente de energía debe aplicar al Pentium II el voltaje solicitado, como se muestra en la tabla 19-2.

El sistema de memoria

El sistema de memoria para el microprocesador Pentium II es de 64 Gbytes, igual que el microprocesador Pentium Pro. Ambos microprocesadores direccionan un sistema de memoria de 64 bits, con un bus de direcciones de 36 bits. La mayoría de los sistemas utilizan SDRAM que opera a 66 MHz o a 100 MHz para el Pentium II. La SDRAM para el sistema de 66 MHz tiene un tiempo de acceso de 10 ns y la SDRAM para el sistema de 100 MHz tiene un tiempo de acceso de 8 ns. En este capítulo no mostraremos el sistema de memoria, que se conecta al conjunto de chips. En los capítulos anteriores vio la organización de un sistema de memoria de 64 bits sin ECC.

El sistema de memoria del Pentium II se divide en ocho o nueve bancos que almacenan cada uno un byte de datos. Si está presente el noveno bit, almacena un código de comprobación de errores (ECC). Al igual que los microprocesadores 80486-Pentium Pro, el Pentium II emplea la generación de paridad y la lógica de comprobación de manera interna para la información del bus de datos del sistema de memoria. (Tenga en cuenta que la mayoría de los sistemas Pentium II no utilizan comprobaciones de paridad, pero está disponible.) Si se utilizan comprobaciones de paridad, cada banco de memoria contiene un noveno bit. La memoria de 64 bits es importante para los datos de punto flotante con doble precisión. Recuerde que un número de punto flotante con doble precisión es de 64 bits. Al igual que en el Pentium Pro, el sistema de memoria se enumera en bytes, desde el byte 00000000H hasta el byte FFFFFFFFHF. Tome en cuenta que ninguno de los conjuntos de chips actuales soporta más de 1 Gbyte de memoria de sistema, por lo que las conexiones de dirección adicionales son para una expansión futura. En la figura 19-3 se muestra el mapa de memoria básico para el sistema Pentium II, mediante el uso del AGP para la tarjeta de video.

El mapa de memoria para el sistema Pentium II es similar al mapa que mostramos en capítulos anteriores, sólo que se utiliza un área de la memoria para el área del AGP. Esta área permite que la tarjeta de video y Windows accedan a la información de video en un espacio de direcciones lineales. Esto es distinto a la ventana de 128 Kbytes en el área de DOS para una tarjeta de video VGA estándar.

TABLA 19-2 Voltajes de la fuente de energía para el Pentium II, según lo requerido por las terminales VID.

VID4	VID3	VID2	VID1	VID0	V _{CC}
0	0	0	0	0	2.05 V
0	0	0	0	1	2.00 V
0	0	0	1	0	1.95 V
0	0	0	1	1	1.90 V
0	0	1	0	0	1.85 V
0	0	1	0	1	1.80 V
0	0	1	1	0	—
0	0	1	1	1	—
0	1	0	0	0	—
0	1	0	0	1	—
0	1	0	1	0	—
0	1	0	1	1	—
0	1	1	0	0	—
0	1	1	0	1	—
0	1	1	1	0	—
0	1	1	1	1	—
1	0	0	0	0	3.5 V
1	0	0	0	1	3.4 V
1	0	0	1	0	3.3 V
1	0	0	1	1	3.2 V
1	0	1	0	0	3.1 V
1	0	1	0	1	3.0 V
1	0	1	1	0	2.9 V
1	0	1	1	1	2.8 V
1	1	0	0	0	2.7 V
1	1	0	0	1	2.6 V
1	1	0	1	0	2.5 V
1	1	0	1	1	2.4 V
1	1	1	0	0	2.3 V
1	1	1	0	1	2.2 V
1	1	1	1	0	2.1 V
1	1	1	1	1	—

El beneficio son las actualizaciones de video más rápidas, ya que la tarjeta de video no necesita realizar paginaciones a través de la memoria de video de 128 K del DOS:

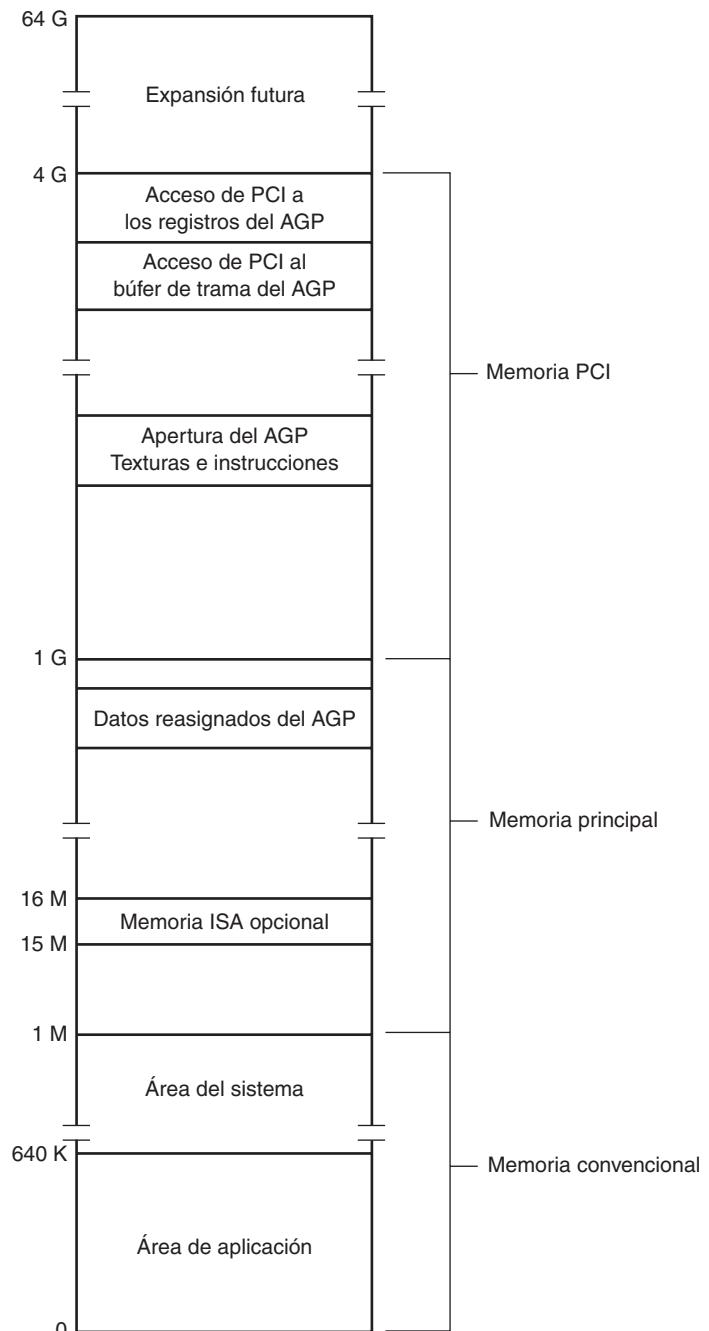
Las transferencias entre el Pentium II y el sistema de memoria se controlan mediante el conjunto de chips 440 LX o 440 BX. Las transferencias de datos entre el Pentium II y el conjunto de chips son de ocho bytes. El conjunto de chips se comunica con el microprocesador a través de las cinco señales REQ, como se muestra en la tabla 19-3. En esencia, el conjunto de chips controla el Pentium II, lo cual es completamente distinto del método tradicional en el que un microprocesador se conecta al sistema en forma directa con la memoria.

El Pentium II sólo se conecta en forma directa con la caché, que se encuentra en el cartucho del Pentium II. Como se mencionó, la caché del Pentium II opera a la mitad de la frecuencia del reloj del microprocesador. Por lo tanto, la caché de un Pentium II de 400 MHz opera a 200 MHz. La caché del Pentium II Xeon trabaja a la misma frecuencia que el microprocesador, lo cual significa que el XEON, con su caché de 512 K, 1 M o 2 M, supera en rendimiento al Pentium II estándar.

Sistema de entrada/salida

El sistema de entrada/salida del Pentium II es 100% compatible con los primeros microprocesadores Intel. El número de puerto de E/S aparece en las líneas de dirección A₁₅-A₃ y las señales de habilitación

FIGURA 19-3 El mapa de memoria de un sistema computacional basado en el Pentium II.



de banco se utilizan para seleccionar los bancos de memoria actuales para la transferencia de E/S. Las transferencias se controlan mediante el conjunto de chips, lo cual es completamente distinto de la arquitectura estándar de los microprocesadores anteriores al Pentium II.

Desde el microprocesador 80386, la información de privilegios de E/S se agrega al segmento TSS cuando el Pentium II se opera en el modo protegido. Recuerde que esto permite inhibir los puertos de E/S en forma selectiva. Si se accede a la ubicación de E/S bloqueada, el Pentium II genera una interrupción tipo 13 para indicar una violación de privilegios de E/S.

TABLA 19-3 Las señales REQ para el Pentium II.

<i>REQ4–REQ0</i>	<i>Nombre</i>	<i>Comentario</i>
00000	Contestación aplazada	Se envían respuestas aplazadas para las transacciones que se aplazaron antes.
00001	Reservada	—
00010	Lectura de memoria reservada e invalidación	Lectura de memoria de DRAM o escritura de PCI hacia DRAM desde PCI.
00011	Reservada	—
00100	Lectura de código de memoria	Ciclo de lectura de memoria.
00101	Escritura aplazada en memoria	Ciclo de escritura aplazada en memoria.
00110	Lectura de datos de memoria	Ciclo de lectura de memoria.
00111	Escritura en memoria	Ciclo normal de escritura en memoria.
01000	Reconocimiento de interrupción o ciclo especial	Reconocimiento de interrupción para el bus PCI.
01001	Reservada	—
10000	Lectura de E/S	Operación de lectura de E/S.
10001	Escritura de E/S	Operación de escritura de E/S.

Sincronización del sistema

Al igual que con cualquier microprocesador, hay que comprender el funcionamiento de las señales de sincronización del sistema para conectar el microprocesador, por lo menos así era antes. Como el Pentium II fue diseñado para que el conjunto de chips lo controlen, las señales de sincronización entre el microprocesador y el conjunto de chips son propiedad de Intel.

19-2

MODIFICACIONES DE SOFTWARE PARA EL PENTIUM II

El núcleo del microprocesador Pentium II es un Pentium Pro. Esto significa que el Pentium II y el Pentium Pro son en esencia el mismo dispositivo para el software. En esta sección veremos las modificaciones a la instrucción CPUID y a las instrucciones SYSENTER, SYSEXIT, FXSAVE y FXRSTORE (las únicas modificaciones al software).

Instrucción CPUID

La tabla 19-4 muestra los valores que se pasan entre el Pentium II y la instrucción CPUID. Hay modificaciones en comparación con las versiones anteriores del microprocesador Pentium.

La versión de la información , que se devuelve después de ejecutar la instrucción CPUID con un 0 lógico en EAX, se regresa en EAX. El ID de la familia se devuelve en los bits 8 al 11; el ID del modelo devuélvelo hace en los bits 4 al 7. El ID de escalonamiento, en los bits del 0 al 3. Para el Pentium II, el número de modelo es 6 y el ID de la familia es 3. El número de escalonamiento hace referencia a un número de actualización; entre mayor sea el número de escalonamiento, más reciente será la versión.

Las características se indican en el registro EDX tras ejecutar la instrucción CPUID con un cero en EAX. Sólo se devuelven dos nuevas características en EDX para el Pentium II. La posición de bit 11 indica si el microprocesador soporta o no las dos nuevas instrucciones de llamada rápida: SYSENTER y SYSEXIT. La posición de bit 23 indica si el microprocesador soporta el conjunto de instrucciones MMX que vimos en el capítulo 14. El resto de los bits es idéntico a las versiones anteriores del microprocesador, por lo que no los describiremos. El bit 16 indica si el microprocesador soporta la tabla de atributos de página, PAT. El bit 17 indica si el microprocesador soporta la extensión del tamaño de las páginas que se utiliza en los microprocesadores Pentium Pro y Pentium II. La extensión del tamaño de las páginas permite direccionar la memoria por encima de los 4 G, hasta 64 G. Por último, el bit 24

TABLA 19-4 La instrucción CPUID para el Pentium II.

<i>Entrada en EAX</i>	<i>Registro de salida</i>	<i>Contenido</i>
0	EAX	Entrada máxima permitida en EAX para CPUID.
0	EBX	“uneG”.
0	ECX	“Inei”.
0	EDX	“letn”.
1	EAX	Número de versión.
1	EDX	Información de las características.
2	EAX	Datos en caché.
2	EBX	Datos en caché.
2	ECX	Datos en caché.
2	EDX	Datos en caché.

indica si se van a implementar las instrucciones de almacenamiento de punto flotante (FXSAVE) y restauración (FXRSTOR).

Instrucciones SYSENTER y SYSEXIT

Estas instrucciones utilizan la característica de llamada rápida que se introdujo en el microprocesador Pentium II. Tenga en cuenta que tales instrucciones sólo funcionan en el anillo 0 (nivel de privilegio 0) en modo protegido. Windows opera en el anillo 0, pero no permite que las aplicaciones tengan acceso al anillo 0. Las nuevas instrucciones fueron diseñadas para el software del sistema operativo, ya que no funcionarán en ningún otro nivel de privilegios.

La instrucción SYSENTER utiliza algunos de los registros específicos del modelo para almacenar CS, EIP y ESP para ejecutar una llamada rápida a un procedimiento definido por el registro específico del modelo. La llamada rápida es distinta a una llamada regular debido a que no mete la dirección de retorno en la pila, como lo hace una llamada regular. La tabla 19-5 muestra el registro específico del modelo que se utiliza con SYSENTER y SYSEXIT. Tenga en cuenta que los registros específicos del modelo se leen mediante la instrucción RDMSR y se escribe en ellos mediante la instrucción WRMSR.

Para utilizar las instrucciones RDMSR o WRMSR, coloque el número de registro en el registro ECX. Si se utiliza WRMSR, ponga los nuevos datos para el registro en EDX:EAX. Para la instrucción SYSENTER sólo necesita utilizar el registro EAX, pero debe colocar un cero en EDX. Si se utiliza la instrucción RDMSR en un programa, los datos se devuelven en los registros EDX:EAX.

Para utilizar la instrucción SYSENTER, primero debe cargar los registros específicos del modelo con la dirección del punto de entrada al sistema en los registros SYSENTER_CS, SYSENTER_ESP y SYSENTER_EIP. En general, sería la dirección de entrada y el área de la pila de un sistema operativo tal como Windows 2000 o Windows XP. La instrucción fue diseñada como instrucción de sistema para acceder al código o software en el anillo 0. El registro del segmento de pila se carga con el valor que se coloca en SYSENTER_CS más 8. En otras palabras, el par de selectores direccionado por el valor del selector SYSENTER_CS se carga en CS y SS. El valor del desplazamiento de la pila se carga en SYSENTER_ESP.

TABLA 19-5 Los registros específicos del modelo que se utilizan con SYSENTER y SYSEXIT.

<i>Nombre</i>	<i>Número</i>	<i>Función</i>
SYSENTER_CS	174H	SYSENTER segmento de código de destino.
SYSENTER_ESP	175H	SYSENTER apuntador de pila de destino.
SYSENTER_EIP	176H	SYSENTER apuntador de instrucción de destino.

TABLA 19-6 Selectores direccionados por el valor de SYSENTER_CS.

SYSENTER_CS(MSR 174H)	Función
SYSENTER_CS valor	SYSENTER selector del segmento de código.
SYSENTER_CS valor + 8	SYSENTER selector del segmento de pila.
SYSENTER_CS valor + 16	SYSEXIT selector del segmento de código.
SYSENTER_CS valor + 24	SYSEXIT selector del segmento de pila.

La instrucción SYSEXIT carga CS y SS con el par de selectores direccionados por SYSENTER_CS más 16 y 24. La tabla 19-6 muestra los selectores de la tabla de selectores globales, según el direccionamiento realizado por SYSENTER_CS. Además de los selectores del segmento de código y de pila, así como de los segmentos de memoria que representan, la instrucción SYSEXIT pasa el valor en EDX al registro EIP, y el valor en ECX al registro ESP. La instrucción SYSEXIT devuelve el control de regreso al anillo 3 de aplicaciones. Como se mencionó, estas instrucciones parecen haber sido diseñadas para una rápida entrada y un rápido retorno de los sistemas operativos Windows o Windows NT en la computadora personal.

Para utilizar SYSENTER y SYSEXIT, la instrucción SYSENTER debe pasar la dirección de retorno al sistema. Esto se logra mediante la operación de cargar el registro EDX con el desplazamiento de retorno y mediante la colocación de la dirección de segmento en la tabla de descriptores globales en la posición SYSENTER_CS+16. El segmento de pila se transfiere mediante la operación de cargar el selector del segmento de pila en SYSENTER_CS+24 y el registro ESP en ECX.

Instrucciones FXSAVE y FXRSTOR

Las dos instrucciones recién agregadas al microprocesador Pentium II son FXSAVE y FXRSTOR, que son casi idénticas a las instrucciones FSAVE y FRSTOR, que vimos en el capítulo 14. La principal diferencia es que la instrucción FXSAVE fue diseñada para almacenar en forma apropiada el estado del equipo MMX, mientras que FSAVE almacena de manera adecuada el estado del coprocesador de punto flotante. La instrucción FSAVE almacena el campo de etiqueta completo, en tanto que la instrucción FXSAVE sólo almacena los bits válidos del campo de etiqueta. El campo válido de etiqueta se utiliza para reconstruir el campo de etiqueta de restauración cuando se ejecuta la instrucción FXRSTOR. Esto significa que si se almacena el estado MMX del equipo, se utiliza la instrucción FXSAVE; si se almacena el estado de punto flotante, se maneja la instrucción FSAVE. Para aplicaciones nuevas, se recomienda utilizar las instrucciones FXSAVE y FXRSTOR para almacenar el estado MMX y el estado de punto flotante del equipo. No utilice las instrucciones FSAVE y FRSTOR en aplicaciones nuevas.

19-3

EL PENTIUM III

El microprocesador Pentium III es una versión mejorada del microprocesador Pentium II. Aun cuando es más reciente que el Pentium II, de todas formas se basa en la arquitectura del Pentium Pro.

Hay dos versiones del Pentium III. Una está disponible con una caché de 512 K sin bloqueo y se empaqueta en el cartucho 1 tipo ranura; la otra está disponible con una caché de transferencia avanzada de 256 Kbytes y se empaqueta en un circuito integrado. La ranura 1 versión caché opera a la mitad de la velocidad del procesador y la versión con caché integrada lo hace a la frecuencia del reloj del microprocesador. Como se muestra en la mayoría de las pruebas de rendimiento de la caché, si se aumenta el tamaño de la caché de 256 Kbytes a 512 Kbytes sólo se mejorará el rendimiento en un pequeño porcentaje.

Conjuntos de chips

El conjunto de chips para el Pentium III es distinto del que se utiliza en el Pentium II. El Pentium III maneja un conjunto de chips Intel 810, 815 u 820. El 815 se encuentra con más frecuencia en sistemas más recientes que utilizan el Pentium III. Hay unos cuantos conjuntos de chips de otros fabricantes disponibles, pero se han reportado problemas con los controladores para nuevos dispositivos, como las

tarjetas de video. También se desarrolló un conjunto de chips 840 para el Pentium III, pero Intel no lo sacó al mercado.

Bus

La versión Coppermine del Pentium III aumenta la velocidad del bus a 100 MHz o 133 MHz. La versión más rápida permite transferencias entre el microprocesador y la memoria a velocidades mayores. La última versión del Pentium III que salió al mercado fue un microprocesador de 1 GHz con un bus de 133 MHz.

Suponga que tiene un microprocesador de 1 GHz que utiliza un bus de memoria de 133 MHz. Podría pensar que la velocidad del bus de memoria debería ser mayor para mejorar el rendimiento, algo en lo cual estamos de acuerdo. Sin embargo, las conexiones entre el microprocesador y la memoria impiden el uso de una velocidad mayor para la memoria. Si decidíramos usar una velocidad de bus de 200 MHz, tendríamos que reconocer que una longitud de onda a 200 MHz es de $300,000,000/200,000,000$ o 3/2 metros. Una antena mide 1/4 de una longitud de onda. A 200 MHz, una antena es de 14.8 pulgadas. Como no es conveniente irradiar energía a 200 MHz, necesitamos mantener el tamaño de las conexiones de la tarjeta de circuitos impresos menor que 1/4 de longitud de onda. En la práctica, habría que mantener el tamaño de las conexiones a no más de 1/10 de 1/4 de longitud de onda. Lo anterior significa que las conexiones en un sistema de 200 MHz serían menores de 1.48 pulgadas. Este tamaño representa un problema para el fabricante de tarjetas principales a la hora de colocar los zócalos (bases para microprocesador) para un sistema de memoria de 200 MHz. Un sistema de bus de 200 MHz bien sería el límite para la tecnología. Si el bus está puesto a punto, podría haber una manera de subir la frecuencia; sólo el tiempo determinará si es posible. Hasta hoy, todo lo que se puede hacer es jugar con las palabras en la publicidad, como 800 Mbytes por segundo para clasificar la velocidad de un bus. (Como se transfieren 64 bits [8 bytes] a la vez, en realidad 800 Mbytes por segundo son 100 MHz.)

¿Sería posible sobrepasar el sistema de memoria de 200 MHz? Sí, si desarrollamos una nueva tecnología para interconectar el microprocesador, el conjunto de chips y la memoria. En la actualidad, la memoria funciona en ráfagas de cuatro números de 64 bits cada vez que se lee la memoria principal. Esta ráfaga de 32 bits se lee y se coloca en la caché. La memoria principal requiere tres estados de espera a 100 MHz para acceder al primer número de 64 bits y después cero estados de espera para cada uno de los tres números de 64 bits restantes, para un total de siete ciclos de reloj del bus de 100 MHz. Lo anterior significa que estamos leyendo datos a $70\text{ ns}/32 = 2.1875\text{ ns}$ por byte, lo cual representa una velocidad de bus de 457 Mbytes por segundo. Esto es más lento que el reloj en un microprocesador de 1 GHz, pero como la mayoría de los programas son cíclicos y las instrucciones se almacenan en una caché interna, con frecuencia nos acercamos a la frecuencia de operación del microprocesador.

Diagrama de terminales

La figura 19-4 muestra el diagrama de terminales de la versión de zócalo (socket) 370 del microprocesador Pentium III. Este circuito integrado se empaqueta en un zócalo tipo matriz de rejillas (PGA) de 370 terminales. Fue diseñado para funcionar con uno de los conjuntos de chips disponibles de Intel. Además de la versión completa del Pentium III, también está disponible el Celeron, que utiliza una velocidad de bus de memoria de 66 MHz. El Pentium III Xeon, también fabricado por Intel, permite un tamaño mayor en la caché para las aplicaciones de servidor.

19-4

EL PENTIUM 4

La versión más reciente del microprocesador con arquitectura de Pentium Pro es el microprocesador Pentium 4 de Intel. Hasta ahora, los microprocesadores Pentium II, Pentium III y Pentium 4 son versiones de la arquitectura del Pentium Pro. El Pentium 4 salió al mercado en noviembre de 2002, con una velocidad de 1.3 GHz. En la actualidad está disponible en velocidades de hasta 3.4 GHz. Hay dos paquetes disponibles para este microprocesador integrado: PGA de 423 terminales y FC-PGA2 de 478 terminales. Ambas versiones del Pentium 4 utilizan la tecnología de 0.18 micrones para su fabricación. Las versiones más recientes emplean la tecnología de 0.13 micrones o la de 90 nm (0.09 micrones). Intel desarrolla en la actualidad una tecnología de 65 nm para sus productos. Al igual que con las primeras

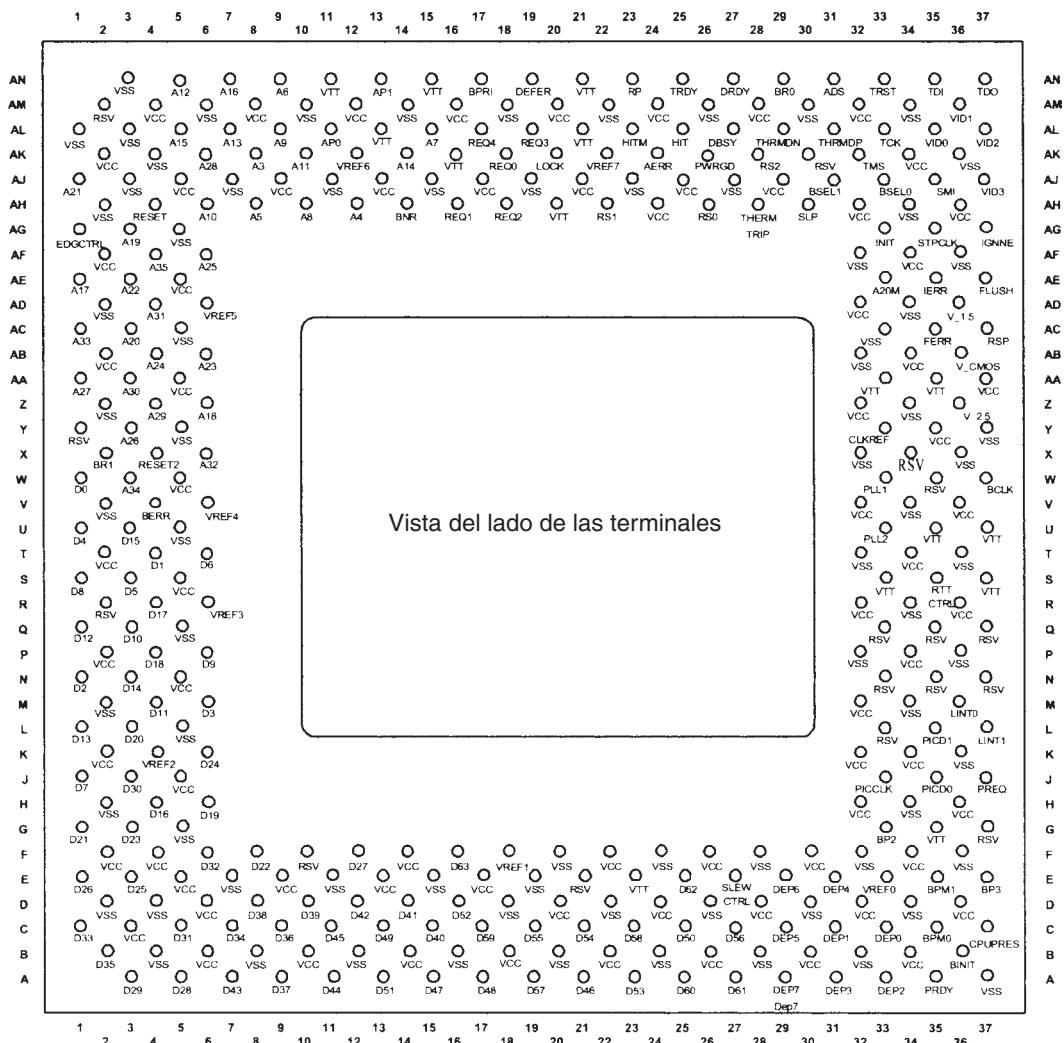


FIGURA 19-4 Diagrama de terminales de la versión del microprocesador Pentium III de zócalo (socket) 370.
(Cortesía de Intel Corporation.)

versiones del Pentium III, el Pentium 4 utiliza una velocidad de bus de memoria de 100 MHz, pero como tiene un impulso cuádruple (quad pumping), la velocidad del bus puede llegar cerca de los 400 MHz. Las versiones más recientes manejan el bus de 133 MHz, que se clasifica como de 533 MHz, debido al impulso cuádruple (quad pumping), o de 200 MHz, que se clasifican como 800 MHz. La figura 19-5 muestra el diagrama de terminales del paquete PGA de 423 terminales del microprocesador Pentium 4.

Interfaz de memoria

Por lo general, la interfaz de memoria para el Pentium 4 utiliza el conjunto de chips Intel 850, 865 u 875. El conjunto de chips 850 proporciona un bus de memoria de canalización doble al microprocesador, en donde cada canalización se conecta a una sección de 32 bits de la memoria. Las dos canalizaciones operan en conjunto para formar la ruta de datos de 64 bits que va al microprocesador. Debido a esta disposición de doble canalización, la memoria debe poblarse con pares de dispositivos de memoria RDRAM, que operan a 600 MHz u 800 MHz. De acuerdo con Intel, tal disposición proporciona un incremento del 300% en la velocidad, en comparación con un sistema de memoria poblado por memoria PC-100.

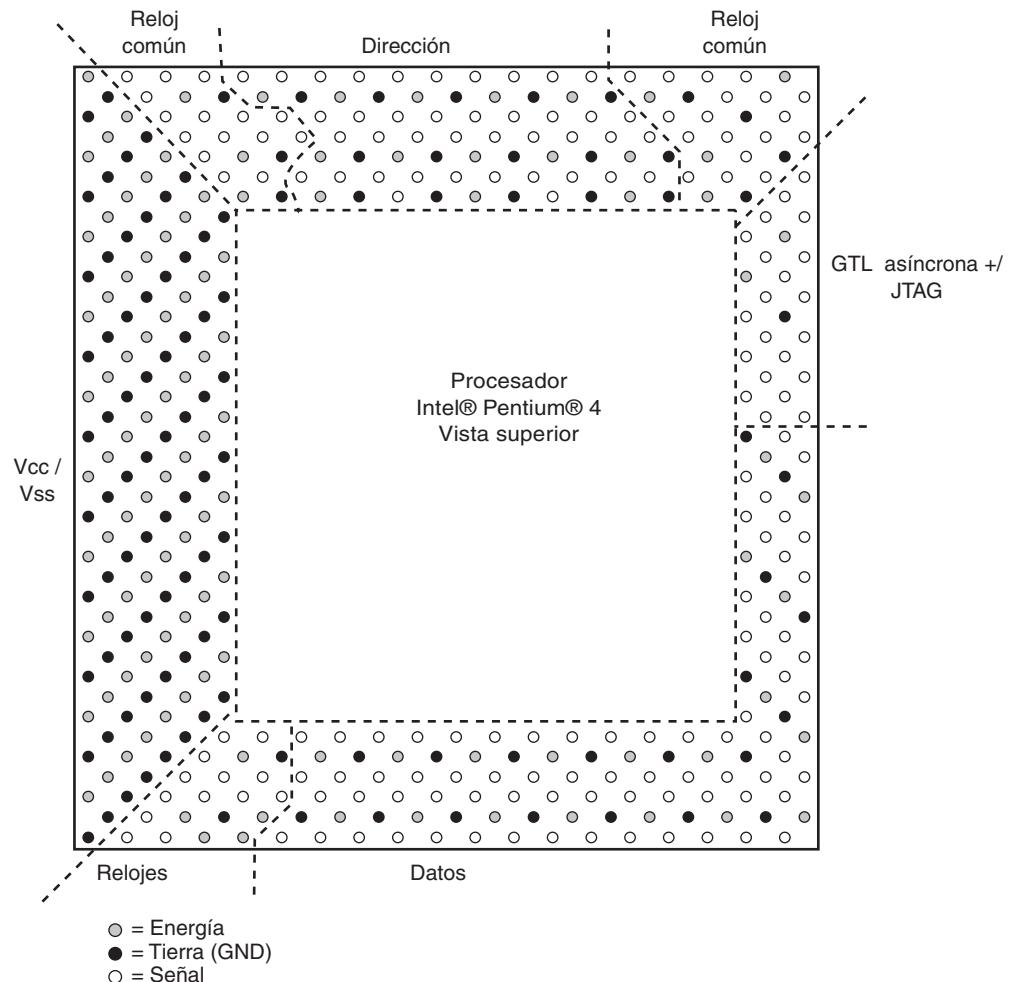


FIGURA 19-5 Diagrama de terminales del paquete PGA 423 del Pentium 4. (Cortesía de Intel Corporation.)

Intel ha dejado de utilizar RDRAM a favor de la memoria DDR (doble velocidad de datos) en los conjuntos de chips 865 y 875. Parece ser que la afirmación de un incremento del 300% en la velocidad de la RDRAM resultó ser falsa. Además de incluir soporte para memoria DDR, también se ha agregado soporte de memoria para la interfaz de disco ATA serial.

Conjunto de registros

El conjunto de registros del Pentium 4 es casi idéntico al de todas las demás versiones del Pentium, con excepción de que los registros MMX son entidades separadas de los registros de punto flotante. Además, se agregaron ocho registros XMM de 128 bits para usarse con las instrucciones SIMD (una instrucción, varios datos) que vimos en el capítulo 14, y con los números de punto flotante con empaquetamiento doble de 128 bits.

Los registros XMM llegan a considerarse como registros MMX de doble anchura que pueden almacenar un par de números de punto flotante de 64 bits con doble precisión o cuatro números de punto flotante con precisión simple. De igual forma, logran almacenar números de 16 bytes, así como los registros MMX almacenan números de 8 bytes. Los registros XMM son registros MMX de doble anchura.

Si descarga el nuevo parche para MASM 6.15 de Microsoft, los programas podrán ensamblarse con el uso de las instrucciones MMX y XMM. El programa ML.EXE se incluye en Microsoft Visual Studio .NET 2003. Para ensamblar programas que incluyan instrucciones MMX, use el modificador .MMX. Para programas que incluyan instrucciones SIMD, maneje el modificador .XMM. El ejemplo 19-1 muestra

un programa muy simple que utiliza las instrucciones MMX para sumar dos números de ocho bytes. Observe cómo se utiliza el modificador .MMX para seleccionar el conjunto de instrucciones MMX. Las instrucciones MOVQ transfieren números entre la memoria y los registros MMX. Estos registros se enumeran de MM₀ a MM₇. También puede usar las instrucciones MMX y SIMD en Microsoft Visual C++ por medio del ensamblador en línea, si descarga el parche más reciente de Microsoft para Visual Studio versión 6.0 o si utiliza una versión más reciente de Visual Studio.

EJEMPLO 19-1

```
.MMX
.DATA
    DATOS1 DQ    1FFH
    DATOS2 DQ    101H
    DATOS3 DQ    ?
.CODE
    MOVQ    MM0,DATOS1
    MOVQ    MM1,DATOS2
    PADDB  MM0,MM1
    MOVQ    DATOS3,MM0
```

De manera similar, el software XMM se puede utilizar en un programa mediante el modificador .XMM. La mayoría de los programas modernos utilizan los registros XMM y el conjunto de instrucciones XMM para realizar operaciones multimedia y demás tipos de operaciones de alta velocidad. El ejemplo 19-2 muestra un programa corto que ilustra el uso de unas cuantas instrucciones XMM. Este programa multiplica dos conjuntos de números de punto flotante de precisión simple y almacena los cuatro productos en las cuatro dobles palabras en RESP. Para habilitar el acceso a las palabras octales (números de 128 bytes), utilizamos la directiva OWORD PTR. Observe también que se maneja el modelo FLAT con el perfil C. Las instrucciones SIMD sólo funcionan en modo protegido, por lo que el programa emplea el formato de modelo plano (FLAT). Lo anterior significa que se colocan los modificadores .686 y .XMM antes de la instrucción del modelo.

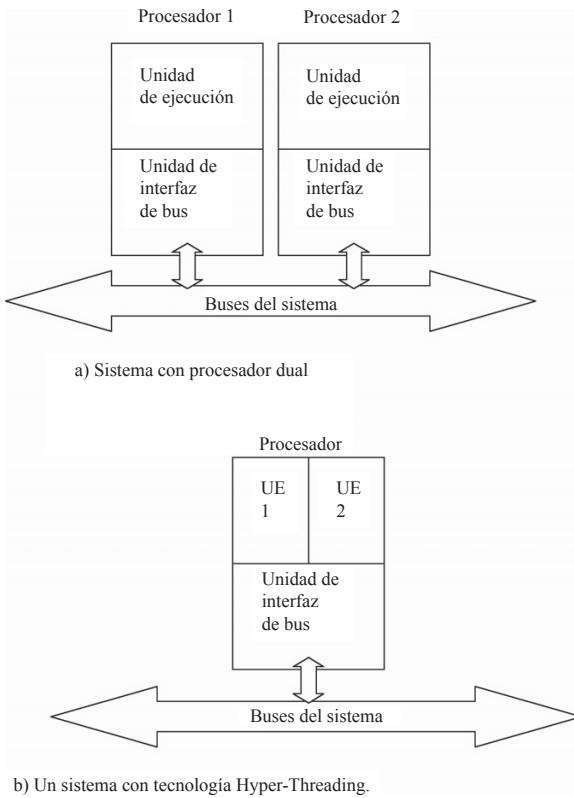
EJEMPLO 19-2

```
.686
.XMM
.MODEL FLAT,C
.DATA
    DATOS1 DD    1.0      ;define cuatro números de punto flotante para
                           ;DATOS1
                           DD    2.0
                           DD    3.0
                           DD    4.0
    DATOS2 DD    6.3      ;define cuatro números de punto flotante para
                           ;DATOS2
                           DD    4.6
                           DD    4.5
                           DD    -2.3
    RESP   DD    4 DUP(?) ;el código adicional va aquí
.CODE
    MOVAPS XMM0,OWORD PTR DATOS1
    MOVAPS XMM1,OWORD PTR DATOS2
    MULPS XMM0,XMM1
    MOVAPS OWORD PTR RESP,XMM0
END
```

Tecnología de hiper-procesamiento (Hyper-Threading)

La innovación más reciente y nueva para el Pentium se conoce como tecnología Hyper-Threading. Este considerable avance combina dos microprocesadores en un solo paquete. Para comprender esta nueva tecnología consulte la figura 19-6, que muestra un sistema tradicional con doble procesador y un sistema con tecnología Hyper-Threading.

FIGURA 19-6 Un sistema con procesador dual y un sistema con procesador y tecnología Hyper-Threading.



El procesador con tecnología Hyper-Threading contiene dos unidades de ejecución, en la que cada una de ellas trae un conjunto completo de los registros capaces de ejecutar software en forma independiente o concurrente. Estos dos contextos de equipos separados comparten una unidad de interfaz de bus común. Durante la operación del equipo, cada procesador es capaz de ejecutar un proceso (thread) en forma independiente, con lo cual se incrementa la velocidad de una aplicación escrita para utilizar múltiples procesos. La unidad de interfaz contiene las cachés de nivel 2 y 3, junto con la interfaz para la estructura de memoria y E/S del equipo. Cuando uno de los microprocesadores necesita acceder a la memoria o a la E/S, debe compartir la unidad de interfaz de bus.

Esta unidad de interfaz de bus se utiliza para acceder a la memoria, pero como el acceso a ella se realiza en ráfagas que llenan las cachés, por lo general está inactivo. Debido a ello, a un segundo procesador le es posible usar este tiempo de inactividad para acceder a la memoria mientras el primer procesador está ocupado ejecutando instrucciones. ¿Se duplica la velocidad del sistema? Sí y no. Algunos procesos pueden ejecutarse en forma independiente unos de los otros siempre y cuando no accedan a la misma área de memoria. Si cada proceso accede a la misma área de memoria, en realidad el equipo operaría a menor velocidad con la tecnología Hyper-Threading. Esto no ocurre muy a menudo; en la mayoría de los casos se incrementa el rendimiento del sistema, por lo que la tecnología Hyper-Threading logra casi el mismo rendimiento que un sistema de procesador dual.

Con el tiempo, la mayoría de los equipos utilizarán la tecnología Hyper-Threading, lo cual significa que se pondrá más atención al desarrollo de software con procesamiento múltiple. En un sistema con procesadores duales o procesadores con Hyper-Threading, cada proceso se ejecuta en un procesador distinto, lo cual incrementa el rendimiento. En el futuro tal vez la arquitectura llegue a incluir aún más procesadores para manejar procesos adicionales.

CPUID

Al igual que en las versiones anteriores del Pentium, la instrucción CPUID accede a la información que indica el tipo de microprocesador, así como las características que soporta. En la siempre evolutiva serie

TABLA 19-7 Instrucciones CPUID del Pentium 4.

<i>Valor de entrada en EAX</i>	<i>Registros de salida</i>	<i>Observaciones</i>
0	EAX = máximo valor de entrada EBX = “uneG” ECX = “lene” EDX = “letn”	Se devuelve “GenuineIntel” en formato Little Endian.
1	EAX = información de la versión EBX = información de las características ECX = información de las características extendidas EDX = información de las características	Información de las características.
2	EAX, EBX, ECX y EDX	Información de caché y TLB.
3	ECX y EDX	Número de serie sólo en el Pentium III.
4	EAX, EBX, ECX y EDX	Parámetros de caché determinísticos.
5	EAX, EBX, ECX y EDX	Información de monitor/Mwait.
8000000H	EAX	Información de funciones extendidas.
80000001H	EAX	Reservado.
80000002H, 80000003H y 80000004H	EAX, EBX, ECX y EDX	Cadena de marca del procesador.
80000006H	ECX	Información de la caché.

de los microprocesadores es importante acceder a esta información, de manera que pueda escribirse software eficiente para operar en muchas versiones distintas del microprocesador.

La tabla 19-7 lista las características más recientes disponibles para las instrucciones CPUID. Para acceder a estas características, EAX se carga con el número de entrada listado en la tabla y después se ejecuta la instrucción CPUID. Por lo general, esta instrucción devuelve información en los registros EAX, EBX, ECX y EDX, en el modo real o en el protegido. Como puede verse de la tabla, se han agregado características adicionales a las instrucciones CPUID, en comparación con las versiones anteriores.

En el capítulo 18 se desarrolló software para leer y mostrar los datos disponibles después de invocar la instrucción CPUID con EAX = 1. Aquí veremos cómo leer la cadena de marca del procesador y prepararla para mostrarla en pantalla mediante una función en Visual C++. Si está soportada, la cadena de marca contiene la frecuencia a la cual el microprocesador está certificado para operar, junto con las palabras clave Intel genuino. La función CadenaMarca (vea el ejemplo 19-3) devuelve un valor CString que contiene la información almacenada en los miembros 80000002H-80000004H de CPUID. Este software requiere un sistema Pentium 4 para que la función CadenaMarca muestre la información apropiada. La función Convierte lee el contenido de EAX, EBX, ECX y EDX del registro especificado como parámetro y lo convierte en un valor CString de retorno. El sistema del autor muestra que la cadena de marca es

“Intel(R) Pentium(R) 4 CPU 3.06GHz”

EJEMPLO 19-3

```
CString CCPUIDDlg::CadenaMarca (void)
{
    CString temp;
    int temp1;
    _asm
    {
        mov     eax,80000000h
```

```

        cpuid
        mov    temp1,eax
    }
    if ( temp1 > 0x80000004 )           //si está presente la cadena de marca
    {
        temp += Convierte(0x80000002);   //registro de lectura 80000002H
        temp += Convierte(0x80000003);   //registro de lectura 80000003H
        temp += Convierte(0x80000004);   //registro de lectura 80000004H
    }
    return temp;
}

CString CCPUIDDlg::Convierte(int  EAXvalor)
{
    CString temp = "                 ";      //deben ser 16 espacios
    int temp1, temp2, temp3, temp4;
    _asm
    {
        mov    eax,EAXvalor
        cpuid
        mov    temp1,eax
        mov    temp2,ebx
        mov    temp3,ecx
        mov    temp4,edx
    }
    for ( int a = 0; a <4; a++ )
    {
        temp.SetAt (a, temp1);
        temp.SetAt (a + 4, temp2);
        temp.SetAt (a + 8, temp3);
        temp.SetAt (a + 12, temp4);
        temp1 >>= 8;
        temp2 >>= 8;
        temp3 >>= 8;
        temp4 >>= 8;
    }
    return temp;
}

```

La demás información disponible acerca del sistema se devuelve en EAX, EBX, ECX y EDX después de ejecutar CPUID, si se carga un 1 en EAX. El registro EAX contiene la información de la versión, como el modelo, la familia y la información de escalonamiento, como se muestra en la figura 19-7. El registro EBX trae información sobre la caché, como el tamaño de la línea de caché que se vacía mediante la instrucción CFLUSH en los bits 15-8 y el ID asignado a APIC local en el renicio, en los bits 31-24. Los bits 23-16 indican cuántos procesadores internos están disponibles para Hyper-

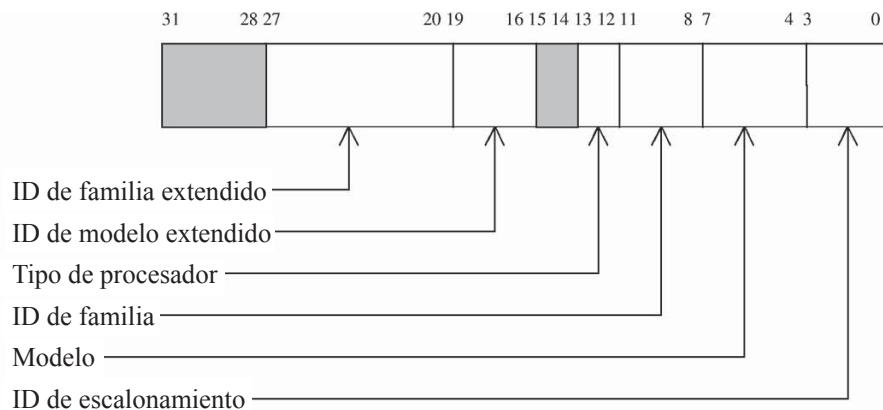


FIGURA 19-7 EAX después de una instrucción CPUID en la que se muestra la información de la versión.

Threading (dos para el microprocesador Pentium 4 actual). El ejemplo 19-4 muestra una función que identifica el número de procesadores en un CPU con Hyper-Threading y lo devuelve como una cadena de caracteres. Si en un futuro se agregan más procesadores al microprocesador, entonces el software en el ejemplo 19-4 tendrá que modificarse.

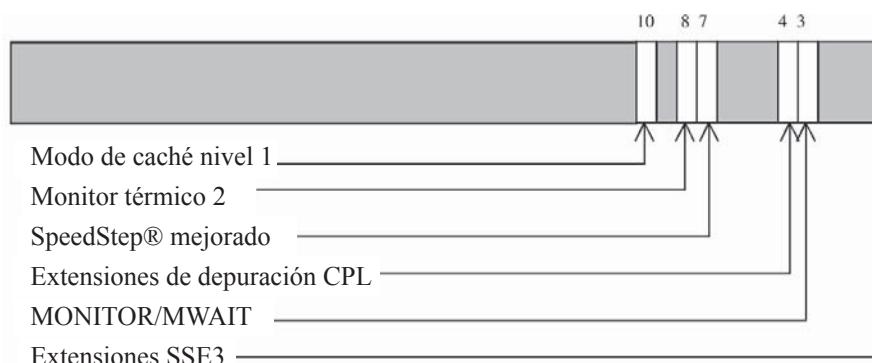
EJEMPLO 19-4

```
CString CCPUIDDlg::ObtieneConteoProcesadores (void)
{
    CString temp = "Este CPU tiene ";
    char templ;
    _asm
    {
        mov     eax,1
        cpuid
        mov     temp1,31h
        bt      edx,28          ;comprueba si hay Hyper-Threading
        jnc    ObtenPro1        ;si no hay Hyper-Threading, temp1 = 1
        bswap   ebx
        add    bh,30h
        mov     temp1,bh
    ObtenPro1:
        return temp + templ + " procesadores.";
    }
}
```

La información de las características para el microprocesador se devuelve en ECX y EDX, como se indica en las figuras 19-8 y 19-9. Cada bit es un 1 lógico si la característica está presente. Por ejemplo, si se necesita Hyper-Threading en el bit de una aplicación, se evalúa la posición 28 en EDX para ver si se soporta el Hyper-Threading. Esto aparece en el ejemplo 19-4, junto con la lectura del número de procesadores encontrados en un microprocesador con Hyper-Threading. La instrucción BT evalúa el bit indicado y lo coloca en la bandera de acarreo. Si el bit bajo prueba es 1, entonces el acarreo resultante es 1 y si el bit bajo prueba es un 0, entonces el acarreo resultante es cero.

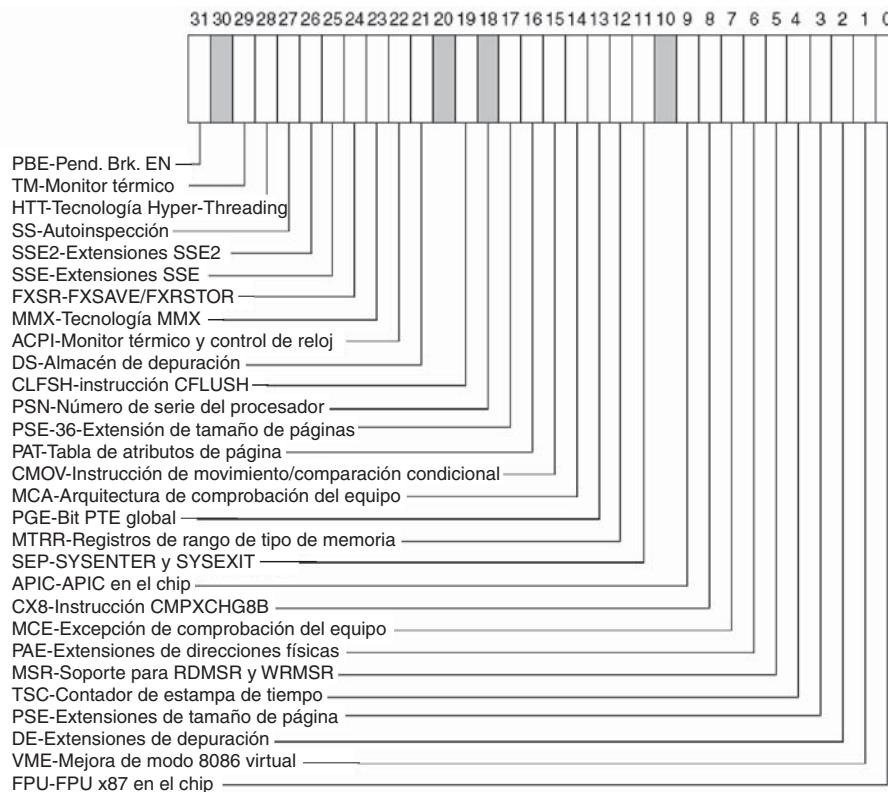
Registros específicos del modelo

Al igual que con las versiones anteriores del Pentium, el Pentium 4 contiene registros específicos del modelo, los cuales se leen mediante la instrucción RDMSR y se escribe en ellos mediante la instrucción WRMSR. El Pentium 4 tiene 1743 registros específicos del modelo, enumerados de 0H a 6CFH. Intel no proporciona información sobre todos esos registros. No los identifica como reservados ni si se utilizan para alguna característica o función no documentada.



Nota: 1 en un bit indica que se soporta la extensión.

FIGURA 19-8 ECX después de una instrucción CPUID en la que se muestran las extensiones de la versión.



Nota: 1 en un bit indica que se soporta la extensión.

FIGURA 19-9 EDX después de una instrucción CPUID en la que se muestran las extensiones de la versión.

Las instrucciones de lectura y escritura para los registros específicos del modelo funcionan de la misma forma. El registro ECX se carga con el número de registro al que se va a acceder, en tanto que los datos se transfieren a través del par de registros EDX:EAX como un número de 64 bits, en donde EDX contiene los 32 bits más significativos y EAX los menos significativos. Se debe acceder a estos registros ya sea en el modo real (DOS) o en el anillo 0 del modo protegido. Por lo general, el sistema operativo es quien accede a estos registros, por lo que no se tiene acceso a ellos en la programación normal con Visual C++.

Registros de monitoreo del rendimiento

Otra característica en el Pentium 4 es un conjunto de registros de monitoreo del rendimiento (PMR), que, al igual que los registros específicos del modelo, sólo pueden usarse en modo real o en el anillo 0 del modo protegido. El único registro al que es posible acceder mediante software de usuario es el contador de estampa de tiempo, el cual es un registro de monitoreo del rendimiento. Para acceder al resto de los PMRs se utiliza la instrucción RDPMR. Dicha instrucción es similar a RDMSR en cuanto a que maneja ECX para especificar el número de registro y el resultado aparece en EDX:EAX. No hay instrucción de escritura para los PMRs.

Tecnología de extensión de 64 bits

Al momento de escribir este libro, Intel anunciaba su tecnología de extensión de 64 bits para la familia de la arquitectura de 32 bits de Intel, pero no ha anunciado todavía cuándo va a salir al mercado un microprocesador que la soporte. El conjunto de instrucciones y la arquitectura son compatibles en forma descendente con el 8086, lo que significa que las instrucciones y el conjunto de registros siguen siendo compatibles. Lo que cambia es que el conjunto de registros se extendió a 64 bits de anchura, en lugar

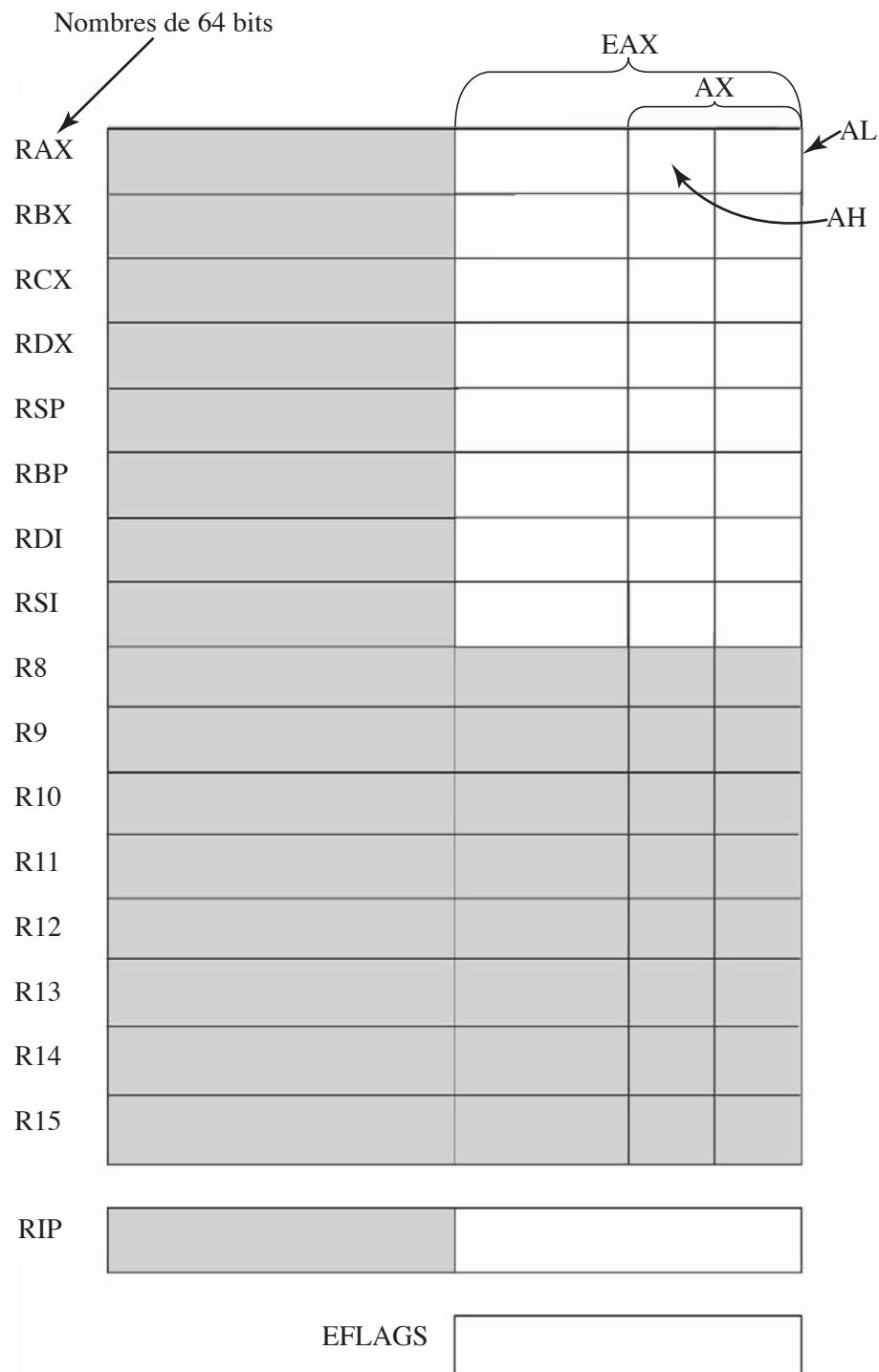


FIGURA 19-10 El conjunto de registros enteros en el modo de 64 bits del Pentium 4. Las áreas sombreadas son nuevas para el Pentium 4 cuando se opera en el modo de 64 bits.

de los registros de 32 bits actuales. En la figura 19-10 podrá consultar el modelo de programación del Pentium 4 en modo de 64 bits.

Observe que ahora el conjunto de registros contiene dieciséis registros de propósito general de 64 bits: RAX, RBX, RCX, RDX, RSP, RBP, RDI, RSI, R₈-R₁₅. El apuntador de instrucciones también se extendió a una anchura de 64 bits, lo cual ayuda al microprocesador acceder a la memoria mediante el uso de una dirección de 64 bits. Lo anterior permite que el microprocesador dirija tanta memoria como corresponda al número de terminales de su implementación específica.

Los registros pueden direccionarse como registros de 64 bits, de 32 bits, de 16 bits o de 8 bits. Ejemplos son R₈ (64 bits), R8D (32 bits), R8W (16 bits) y R8L (8 bits). No hay forma de direccionar el byte superior (como en BH) para un registro enumerado; sólo es posible direccionar su byte inferior. El direccionamiento heredado tal como MOV AH,AL funciona en forma correcta, aunque no se permite direccionar un registro de byte superior y un registro de byte inferior enumerado heredados. En otras palabras, MOV AH,R9L no se permite, pero MOV AL,R9L sí. Si la instrucción MOV AH,R9L se incluye en un programa no se producirá ningún error; en vez de eso, se cambiará a MOV BPL,R9L. AH, BH, CH y DH modificándose los 8 bits de menor orden (la L indica menor orden) de BPL, SPL, DIL y SIL, respectivamente. Aparte de eso, los registros heredados llegan a mezclarse con los nuevos registros enumerados R₈-R₁₅, como en MOV R₁₁, RAX, MOV R11D,ECX o MOV BX,R14W.

Otra adición a la arquitectura es un conjunto de registros SSE adicionales enumerados como XMM₈-XMM₁₅. Para acceder a estos registros se utilizan las instrucciones SSE, SSE₂ o SSE₃. Por lo demás, la unidad SSE no ha sufrido más modificaciones. Los registros de control y de depuración se expandieron a 64 bits de anchura. También se agregó un nuevo registro específico del modelo para controlar las características extendidas en la dirección C0000080H. La figura 19-11 muestra el registro de control de características extendidas.

- | | |
|------------|--|
| SCE | El bit habilita llamada (CALL) del sistema se activa para habilitar las instrucciones SYSCALL y SYSRET en el modo de 64 bits. |
| LME | El bit habilita modo se activa para permitir que el microprocesador utilice el modo extendido de 64 bits. |
| LMA | El bit modo activo muestra que el microprocesador está operando en el modo extendido de 64 bits. |

Los registros de la tabla de descriptores en modo protegido se amplían en el modo extendido de 64 bits, de manera que cada uno de los registros de la tabla de descriptores (GDTR, LDTR, IDTR) y el registro de tareas (TR) almacenen una dirección base de 64 bits, en lugar de una de 32 bits. El mayor cambio es que se ignoran la dirección base y los límites de los descriptores de segmento. El sistema utiliza una dirección base de 0000000000000000H para el segmento de código, en tanto que se ignoran los segmentos DS, ES y SS.

La paginación también se modificó para incluir una unidad de paginación que soporte la traducción de una dirección lineal de 64 bits en una dirección física de 52 bits. Intel afirma que en la primera versión de este Pentium de 64 bits la dirección lineal será de 48 bits y la dirección física de 40 bits. Esto significa que habrá una dirección de 40 bits para soportar 1 Tbyte (terabyte) de memoria física traducida desde un espacio de direcciones lineales de 256 Tbytes. La dirección de 52 bits accede a 4 Pbytes (petabytes) de memoria y una dirección lineal de 64 bits accede a 16 Ebytes (exabytes) de memoria. La traducción se logra mediante el uso de tablas adicionales en la unidad de paginación. En lugar de dos tablas (un directorio de páginas y una tabla de páginas), la unidad de paginación extendida de 64 bits utiliza cuatro niveles de tablas de páginas.

63	12	11	9	8	7	1	0
	LMA		LME			SCE	

FIGURA 19-11 El contenido del registro de características extendidas específico del modelo.

19-5**RESUMEN**

1. El Pentium II difiere de los primeros microprocesadores debido a que, en lugar de ofrecerse como un circuito integrado, está disponible en un cartucho insertable o una tarjeta de circuitos impresos.
2. La caché de nivel 2 para el Pentium II está montada dentro del cartucho, excepto en el Celeron, que no cuenta con caché de nivel 2. La velocidad de la caché es la mitad de la velocidad de reloj del Pentium II, excepto en el Xeon, en donde tiene la misma velocidad que el Pentium II. Todas las versiones del Pentium II contienen una caché interna de nivel 1 que almacena 32 Kbytes de datos.
3. El Pentium II es el primer microprocesador Intel que se controla desde un controlador de bus externo. A diferencia de las primeras versiones del microprocesador, que emitían señales de lectura y de escritura, un controlador de bus externo ordena al Pentium II que lea o escriba información.
4. El Pentium II opera a frecuencias de reloj que van desde los 233 MHz hasta los 450 MHz, con velocidades de bus de 66 MHz o 100 MHz. La caché de nivel 2 puede ser de 512 K, 1 M o 2 Mbytes. El Pentium II contiene un bus de datos de 64 bits y un bus de direcciones de 36 bits, lo que le permite acceder hasta 64 Gbytes de memoria.
5. Las nuevas instrucciones que se agregaron al Pentium II son SYSENTER, SYSEXIT, FXSAVE y FXRSTOR.
6. Los comandos SYSENTER y SYSEXIT se optimizaron para acceder al sistema operativo en el nivel de privilegio 0 desde un acceso de nivel de privilegio 3. Estas instrucciones operan a una velocidad mucho más alta que una comutación de tareas, o incluso una combinación de llamada y retorno.
7. Las instrucciones FXSAVE y FXRSTOR se optimizaron para almacenar en forma apropiada el estado de la unidad de tecnología MMX y del coprocesador de punto flotante.
8. El microprocesador Pentium III es una extensión de la arquitectura del Pentium Pro, con la adición del conjunto de instrucciones SIMD que utilizan los registros XMM.
9. El microprocesador Pentium 4 es una extensión de la arquitectura del Pentium Pro que incluye mejoras que le permiten operar a frecuencias de reloj más altas de lo que era posible, gracias a las tecnologías de fabricación de 0.13 micrones y la más reciente de 90 nm.
10. El microprocesador Pentium 4 requiere una fuente de energía y un gabinete ATX modificados para funcionar en forma apropiada en un sistema.
11. La versión 6.15 del programa MASM y la versión 6 de Visual Studio cuentan ahora con soporte para las nuevas instrucciones MMX y SIMD, mediante el uso del modificador .686 con los modificadores .MMX y .XMM.
12. Los microprocesadores Pentium II, Pentium III y Pentium 4 son todos variaciones del microprocesador Pentium Pro.
13. Los futuros microprocesadores Pentium 4 utilizarán la extensión de 64 bits a la arquitectura de 32 bits. Al momento de escribir este libro no hay dispositivo o información concreta al respecto.

19-6**PREGUNTAS Y PROBLEMAS**

1. ¿Cuál es el tamaño de la caché de nivel 1 en el microprocesador Pentium II?
2. ¿Qué tamaños hay disponibles para la caché de nivel 2 en el microprocesador Pentium II? (Liste todas las versiones.)
3. ¿Cuál es la diferencia entre la caché de nivel 2 en el sistema basado en Pentium y la del sistema basado en Pentium II?
4. ¿Cuál es la diferencia entre la caché de nivel 2 en el sistema basado en el Pentium Pro y la del Pentium III?
5. La velocidad de la caché de nivel 2 del Pentium II Xeon es _____ veces más rápida que la caché en el Pentium II (excluyendo al Celeron).

6. ¿Cuánta memoria puede direccionar el Pentium II?
7. ¿Está el Pentium II disponible en formato de circuito integrado?
8. ¿Cuántas conexiones de terminales hay en el cartucho del Pentium II?
9. ¿Cuál es el propósito de las señales de control PICD?
10. ¿Qué ocurrió con las terminales de lectura y de escritura en el Pentium II?
11. ¿A qué velocidades de bus opera el Pentium II?
12. ¿Qué tan veloz es la SDRAM que se conecta al sistema Pentium II para una versión con velocidad de bus de 100 MHz?
13. ¿Qué tan amplia es la memoria del Pentium II si se emplea ECC?
14. ¿Qué nuevos registros específicos del modelo (MSR) se han agregado al microprocesador Pentium II?
15. ¿Qué nueva información de identificación de CPUD se ha agregado al microprocesador Pentium II?
16. ¿Cómo se direcciona un registro específico del modelo y qué instrucción se utiliza para leerlo?
17. Escriba software que almacene 12H en el registro específico del modelo 175H.
18. Escriba un procedimiento corto que determine si el microprocesador contiene las instrucciones SYSENTER y SYSEXIT. Su procedimiento deberá devolver un 1 lógico en el acarreo, si las instrucciones están presentes, y un 0 lógico, si no están presentes.
19. ¿Cómo se transfiere la dirección de retorno al sistema cuando se utiliza la instrucción SYSENTER?
20. ¿Cómo se recupera la dirección de retorno cuando se utiliza la instrucción SYSEXIT para regresar a la aplicación?
21. ¿En qué nivel de privilegio transfiere la instrucción SYSENTER el control al software?
22. ¿En qué nivel de privilegio transfiere la instrucción SYSEXIT el control al software?
23. ¿Cuál es la diferencia entre las instrucciones FSAVE y FXSAVE?
24. El Pentium III es una extensión de la arquitectura _____.
25. ¿Qué nuevas instrucciones hay en el microprocesador Pentium III, que no aparezcan en el microprocesador Pentium Pro?
26. ¿Qué modificaciones requiere el microprocesador Pentium 4 en la fuente de energía?
27. Escriba un programa corto que lea y muestre el número de serie del microprocesador Pentium III en la pantalla.
28. Desarrolle una función en C++ que devuelva el valor booleano verdadero si el Pentium 4 soporta la tecnología Hyper-Threading, y falso si no lo soporta.
29. Desarrolle una función corta en C++ que devuelva el valor booleano verdadero si el Pentium 4 soporta las extensiones SSE, SSE₂ y SSE₃.
30. Compare, con sus palabras, la tecnología Hyper-Threading y el procesamiento dual. Postule acerca de la posibilidad de incluir más de 2 procesadores adicionales.

APÉNDICE A

El ensamblador, Visual C++ y DOS

Este apéndice se proporciona para que el ensamblador pueda utilizarse en el desarrollo de programas en el entorno DOS y también en el entorno Visual C++. El entorno DOS ha desaparecido casi por completo (a menos que todavía se utilice Windows 98), pero continúa vivo a través del programa de emulación llamado CMD.EXE en la carpeta de accesos directos de Microsoft Windows. Algunas personas derramarían una lágrima debido a la desaparición del DOS, pero toman en cuenta que el entorno DOS era un enorme dolor de cabeza para todos los que pasamos años programando en él. Sólo tenía un sistema de memoria de 1 Mbyte y los controladores eran un problema, en especial en los años recientes. Microsoft en realidad nunca proporcionó un modo protegido decente del DOS. Este sistema operativo mostraba la información textual muy bien, pero los gráficos eran otra historia, porque la arquitectura de la memoria de video y la falta de controladores.

Windows resolvió muchos de los problemas que infestaron al DOS e introdujo la era de la GUI, que es una gran mejora en comparación con las aplicaciones del DOS basadas en texto. Windows es mucho más fácil de usar y controlar para un humano. El autor recuerda los viejos días cuando tenía que escribir archivos de procesamiento por lotes para que su esposa pudiera utilizar su computadora. Ahora ella es una verdadera profesional gracias a Windows. Windows es un estupendo sistema; sin comentarios.

EL ENSAMBLADOR

Aunque el programa ensamblador no se utiliza tan a menudo como un medio de programación independiente, sigue teniendo cierta aplicación en el desarrollo de módulos que se enlazan con programas en Visual C++ (vea el capítulo 7). El programa en sí se incluye con Visual C++ en el directorio C:\Archivos de programa\Microsoft Visual Studio .NET 2003\VC7\bin como ML.EXE. También en el mismo directorio se encuentran el programa LIB.EXE (biblioteca) para crear colecciones de bibliotecas y el programa LINK, que se maneja para enlazar módulos objeto.

El ejemplo A-1 muestra cómo ensamblar un programa escrito en lenguaje ensamblador. Se utiliza un archivo llamado WOW.TXT (no necesita la extensión .ASM aun cuando se maneja con frecuencia para los módulos en lenguaje ensamblador). El archivo WOW.TXT se compila con comprobación de errores (/c = c minúscula) y se genera un archivo de listado (/F1) llamado WOW.LST. Si se necesitan otros modificadores sólo hay que escribir ML/? en el símbolo del sistema para que aparezca una lista de los modificadores. También podría incluirse el modificador /coff (formato de archivos objeto de c) para generar un archivo objeto que logre enlazarse con un programa de Visual C++, como en ML/c/coff WOW.TXT.

EJEMPLO A-1

```
ML /c /F1WOW.LST WOW.TXT
```

TABLA A-1 Modelos del ensamblador que se utilizan comúnmente.

<i>Modelo</i>	<i>Descripción</i>
.TINY	Todos los datos y el código deben ajustarse en un solo segmento de memoria de 64 Kbytes. Los programas diminutos (tiny) se ensamblan como archivos .COM del DOS y deben usar un origen en 0100H para el código.
.SMALL	Un modelo de dos segmentos con un solo segmento de código y un solo segmento de datos. Los programas pequeños (small) generan archivos .EXE del DOS y tienen un origen de 0000H.
.FLAT	El modelo plano (flat) utiliza un solo segmento de hasta 4 Gbytes de longitud. Los programas planos son programas que sólo funcionan en Windows, con un origen de 00000000H.

Si utiliza el programa LINK desde Visual C++, no podrá generar un archivo ejecutable compatible con el DOS, debido a que es un enlazador de 32 bits. El enlazador de 16 bits para el DOS no se encuentra en el paquete de Visual Studio. Si hay que desarrollar software para el DOS, necesitará obtener el kit de desarrollo de controladores de Windows (Windows DDK) de Microsoft. El DDK contiene el enlazador de 16 bits que se requiere para desarrollar aplicaciones del DOS. El enlazador se encuentra en el directorio C:\WINDDK\2600.1106\bin\win_me\bin16 del DDK. Además del programa enlazador, aparece una versión de 16 bits del lenguaje C++ para DOS como CL.EXE. Estos archivos se proporcionan para las aplicaciones heredadas.

El ejemplo A-2 muestra cómo enlazar un programa generado por el ensamblador. Aquí suponemos que usted está usando el programa enlazador de 16 bits en modo real del DOS. Por lo general, el enlazador de 32 bits se utiliza desde Visual C++ para las aplicaciones Windows. Aquí, el objeto generado por el ejemplo A-1 se enlaza para producir un programa ejecutable llamado WOW.EXE o, si se está usando el modelo diminuto, WOW.COM.

EJEMPLO A-2

LINK WOW.OBJ

Modelos de memoria del ensamblador

Aunque el modelo plano es el que se utiliza más a menudo en Visual C++, hay otros modelos de memoria que se manejan con aplicaciones DOS y el desarrollo de programas integrados. La tabla A-1 lista los modelos que se emplean con más frecuencia para esas aplicaciones. El origen se establece mediante la directiva .STARTUP en un programa del DOS y de manera automática en un programa plano.

La tabla A-2 lista la información predeterminada para cada uno de los modelos que se listan en la tabla A-1. Si requiere información adicional sobre los modelos, visite el sitio Web de Microsoft y busque los modelos del ensamblador.

LLAMADAS A FUNCIONES SELECTAS DEL DOS

No se incluyen todas las llamadas a las funciones del DOS, debido a que es muy poco probable que se vayan a utilizar todas. La versión más reciente del DOS tiene llamadas a funciones desde la función 00H hasta la función 6CH. En este libro sólo listaremos las llamadas a funciones que se manejen para aplicaciones simples. Muchas de las llamadas a funciones eran de la versión 1.0 del DOS y han estado obsoletas durante muchos años; otras se utilizan para acceder al sistema de disco, que se emplea en Visual C++.

TABLA A-2 Valores predeterminados para los modelos de lenguaje ensamblador más comunes.

Modelo	Directivas	Nombre	Alineación	Combinado	Clase	Grupo
.TINY	.CODE	_TEXT	Palabra	PUBLIC	'CODE'	DGROUP
	.FARDATA	FAR_DATA	Párrafo	Private	'FAR_DATA'	
	.FARDATA?	FAR_BSS	Párrafo	Private	'FAR_BSS'	
	.DATA	_DATA	Palabra	PUBLIC	'DATA'	DGROUP
	.CONST	CONST	Palabra	PUBLIC	'CONST'	DGROUP
	.DATA?	_BSS	Palabra	PUBLIC	'BSS'	DGROUP
	.CODE	_TEXT	Palabra	PUBLIC	'CODE'	
	.FARDATA	FAR_DATA	Párrafo	Private	'FAR_DATA'	
	.FARDATA?	FAR_BSS	Párrafo	Private	'FAR_BSS'	
	.DATA	_DATA	Palabra	PUBLIC	'DATA'	DGROUP
.SMALL	.CONST	CONST	Palabra	PUBLIC	'CONST'	DGROUP
	DATA?	_BSS	Palabra	PUBLIC	'BSS'	DGROUP
	.STACK	STACK	Párrafo	STACK	'STACK'	DGROUP
	.CODE	_TEXT	DPalabra	PUBLIC	'CODE'	
	.FARDATA	_DATA	DPalabra	PUBLIC	'DATA'	
.FLAT	.FARDATA?	_BSS	DPalabra	PUBLIC	'FBSS'	
	.DATA	_DATA	DPalabra	PUBLIC	'DATA'	DGROUP
	.CONST	CONST	DPalabra	PUBLIC	'CONST'	DGROUP
	DATA?	_BSS	DPalabra	PUBLIC	'BSS'	DGROUP
	.STACK	STACK	DPalabra	STACK	'STACK'	DGROUP

Para utilizar una llamada a una función del DOS en un programa DOS, coloque el nombre de la función en AH y otros datos que sean necesarios en otros registros, como se indica en la tabla A-3. El ejemplo A-3 muestra un ejemplo de la función del DOS número 01H. Esta función lee el teclado del DOS y devuelve un carácter ASCII en AL. Una vez que se cargas todo, ejecuta la instrucción INT 21H para realizar la tarea.

EJEMPLO A-3

```
MOV      AH,01H          ;carga número de función del DOS
INT      21H              ;accede al DOS
;regresa con AL = código de tecla ASCII
```

USO DE VISUAL C++

Muchos de los nuevos ejemplos en el libro utilizan Visual C++ .NET 2003. Hay muy pocos programas escritos en lenguaje ensamblador. Si se maneja este lenguaje, por lo general aparece en un programa en C++ para realizar una tarea especial o para aumentar el rendimiento de una sección de un programa.

No todos están familiarizados con el entorno C++, por lo que hemos agregado esta sección como una guía para configurar programas que utilicen lenguaje ensamblador dentro de Visual C++. El tipo de aplicación más sencillo para esto es una aplicación basada en cuadro de diálogo, que utiliza las Microsoft Foundation Classes (MFC).

TABLA A-3 Llamadas a funciones del DOS.

00H	TERMINA UN PROGRAMA
Entrada	AH = 00H CS = dirección de prefijo de segmento del programa.
Salida	Se entra al DOS.
01H	LEE EL TECLADO
Entrada	AH = 01H
Salida	AL = carácter ASCII.
Observaciones	Si AL = 00H, la llamada a la función debe invocarse de nuevo para leer un carácter ASCII extendido. En la tabla 8-1 del capítulo 8 verá un listado de los códigos de teclado ASCII extendidos. Esta llamada a la función envía como eco, en forma automática, cualquier cosa que se escriba a la pantalla de video.
02H	ESCRIBE EN DISPOSITIVO DE SALIDA ESTÁNDAR
Entrada	AH = 02H DL = carácter ASCII que se va a mostrar.
Observaciones	La llamada a esta función por lo general muestra datos en la pantalla de video.
03H	LEE CARÁCTER DE COM1
Entrada	AH = 03H
Salida	AL = carácter ASCII leído del puerto de comunicaciones.
Observaciones	La llamada a esta función lee datos del puerto de comunicaciones en serie.
04H	ESCRIBE EN COM1
Entrada	AH = 04H DL = carácter que se va a enviar desde COM1.
Observaciones	Esta función transmite datos a través del puerto de comunicación serial. La asignación de puerto COM puede modificarse para utilizar otros puertos COM con las funciones 03H y 04H, mediante el uso del comando MODE del DOS para reasignar COM1 hacia otro puerto COM.

05H	ESCRIBE EN LPT1
Entrada	AH = 05H DL = carácter ASCII que se va a imprimir.
Observaciones	Imprime DL en la impresora de línea conectada a LPT1. Tenga en cuenta que el puerto de impresora puede modificarse con el comando MODE del DOS.
06H	LECTURA/ESCRITURA DIRECTA DE CONSOLA
Entrada	AH = 06H DL = 0FFH o DL = carácter ASCII.
Salida	AL = carácter ASCII.
Observaciones	<p>Si DL = 0FFH a la entrada, entonces esta función lee la consola. Si DL = carácter ASCII, entonces esta función muestra el carácter ASCII en la pantalla de video de la consola (CON).</p> <p>Si se lee un carácter desde el teclado de la consola, la bandera de cero (ZF) indica si se escribió un carácter o no. Una condición de cero indica que no se oprimió ninguna tecla, en tanto que una condición distinta de cero indica que AL contiene el código ASCII de la tecla o un 00H. Si AL = 00H, la función debe invocarse de nuevo para leer un carácter ASCII extendido desde el teclado. Tenga en cuenta que la clave no envía eco a la pantalla de video.</p>
07H	ENTRADA DIRECTA DE CONSOLA SIN ECO
Entrada	AH = 07H
Salida	AL = carácter ASCII
Observaciones	Funciona de igual forma que el número 06H con DL = 0FFH, sólo que no regresará hasta que se oprima la tecla.
08H	LEE ENTRADA ESTÁNDAR SIN ECO
Entrada	AH = 08H
Salida	AL = carácter ASCII.
Observaciones	Trabaja igual que la función 07H, sólo que lee el dispositivo de entrada estándar. El dispositivo de entrada estándar puede asignarse como el teclado o el puerto COM. Esta función también responde a una interrupción del control, mientras que las funciones 06H y 07H no. Una interrupción del control deja que se ejecute la instrucción INT 23H. Ésta trabaja igual que la función 07H, de manera predeterminada.

09H	MUESTRA UNA CADENA DE CARACTERES
Entrada	AH = 09H DS:DX = dirección de la cadena de caracteres.
Observaciones	La cadena de caracteres debe terminar con un \$ ASCII (24H). La cadena de caracteres puede ser de cualquier longitud y contener caracteres de control tales como el retorno de carro (0DH) y el avance de línea (0AH).
0AH	ENTRADA DE TECLADO CON BÚFER
Entrada	AH = 0AH DS:DX = dirección del búfer de entrada del teclado.
Observaciones	El primer byte del búfer contiene el tamaño del mismo (hasta 255). El segundo byte se llena con el número de caracteres escritos al momento del retorno. Los bytes del tercero hasta el final del búfer contienen la cadena de caracteres escrita, seguida de un retorno de carro (0DH). Esta función sigue leyendo el teclado (muestra los datos a medida que se escriben) hasta que se escriba el número especificado de caracteres o hasta que se oprima Intro.
0BH	COMPRUEBA EL ESTADO DEL DISPOSITIVO DE ENTRADA ESTÁNDAR
Entrada	AH = 0BH
Salida	AL = estado del dispositivo de entrada.
Observaciones	Esta función comprueba el dispositivo de entrada estándar para determinar si hay datos disponibles. Si AL = 00, no hay datos disponibles. Si AL = 0FFH, entonces hay datos disponibles que deben introducirse mediante el uso de la función número 08H.
0CH	BORRA BÚFER DE TECLADO E INVOCA FUNCIÓN DE TECLADO
Entrada	AH = 0CH AL = 01H, 06H, 07H o 0AH.
Salida	Consulte la salida para las funciones 01H, 06H, 07H o 0AH.
Observaciones	El búfer del teclado almacena las pulsaciones de tecla mientras los programas trabajan otras tareas. Esta función vacía o borra el búfer y luego invoca a la función de teclado ubicada en el registro AL.

Cree una aplicación de cuadro de diálogo

Inicie Visual Studio, deberá aparecer la pantalla que se muestra en la figura A-1. Haga clic en New Project para empezar un nuevo proyecto en C++. Deberá aparecer la pantalla en la figura A-2. Haga clic en el signo + enseguida de Visual C++ Projects, después haga clic en MFC y por último en MFC Application. En este punto escriba el nombre del proyecto y la ruta en donde se indican. En este ejemplo elegimos el nombre de proyecto MyNewProject. Una vez que haya realizado todo esto, haga clic en OK.

En este momento deberá ver la pantalla que se muestra en el ejemplo A-3. Esta pantalla le permite seleccionar opciones para el programa a través del asistente para aplicaciones. La única característica que necesita modificarse en este formulario es la selección de la aplicación basada en cuadro de diálogo. Una vez revisado todo, haga clic en Finish para generar la aplicación de cuadro de diálogo.

Ahora resumiremos el proceso para crear una aplicación basada en cuadro de diálogo.

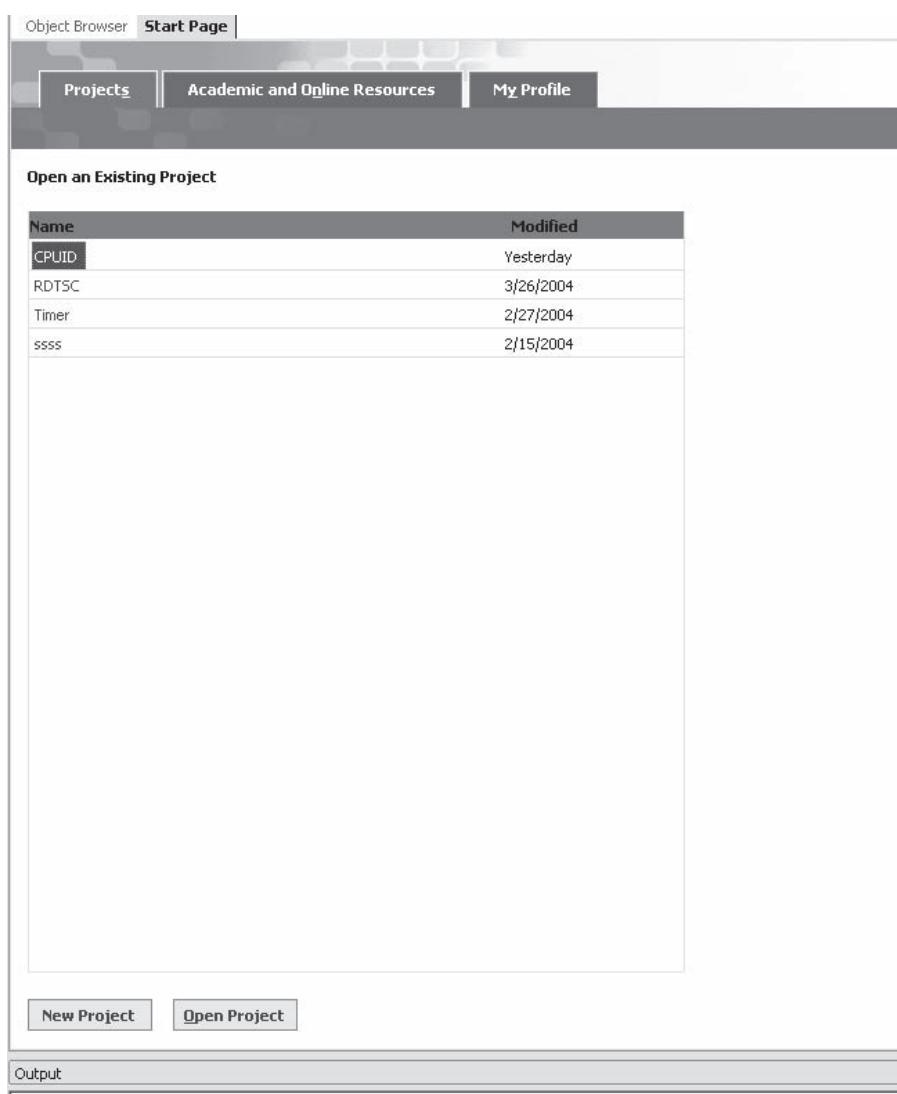


FIGURA A-1 Pantalla de inicio de Visual Studio .NET 2003.

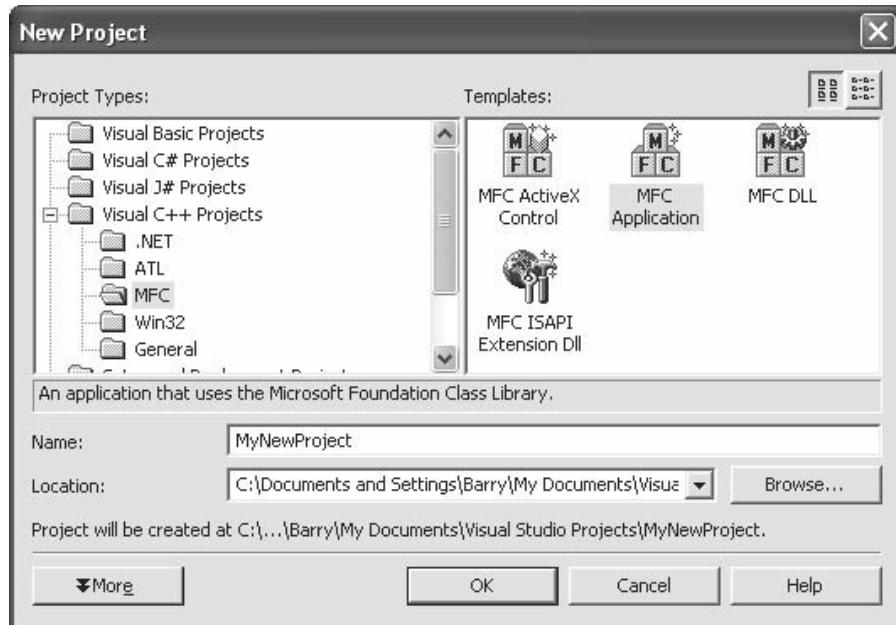


FIGURA A-2 La nueva pantalla de proyectos.

1. Inicie Visual Studio .NET 2003.
2. Haga clic en New Project.
3. Haga clic en MFC Application y escriba un nombre de proyecto, después haga clic en OK.
4. Seleccione Application Type y elija Dialog-Based.
5. Active la casilla aplicación Dialog-Based y luego haga clic en Finish.

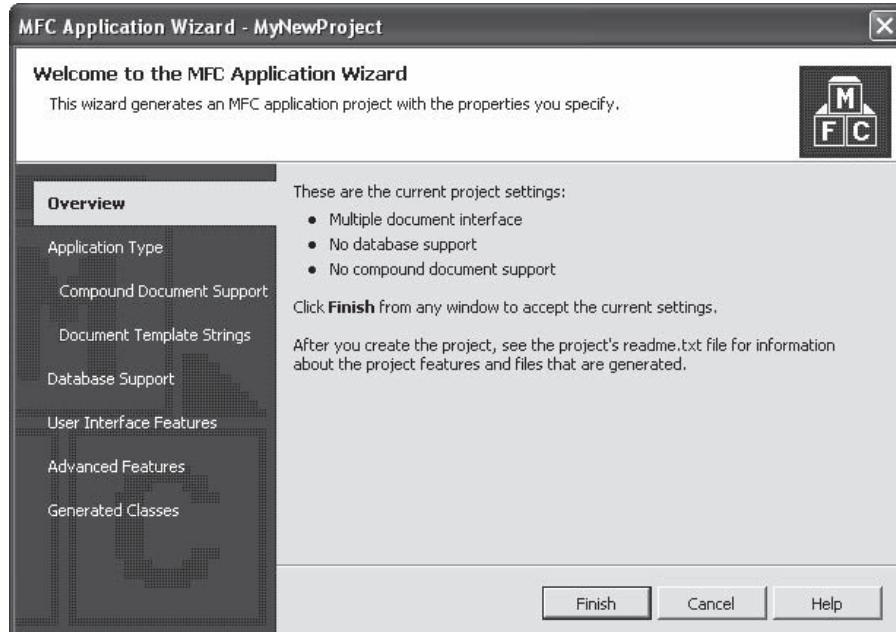


FIGURA A-3 El Asistente para aplicaciones MFC.

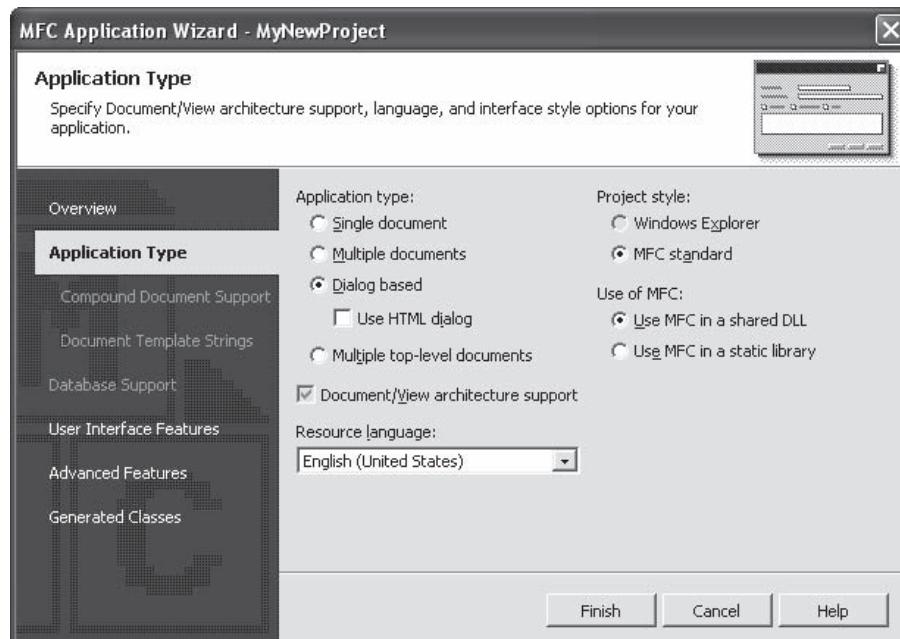


FIGURA A-4 Selección del tipo de aplicación.

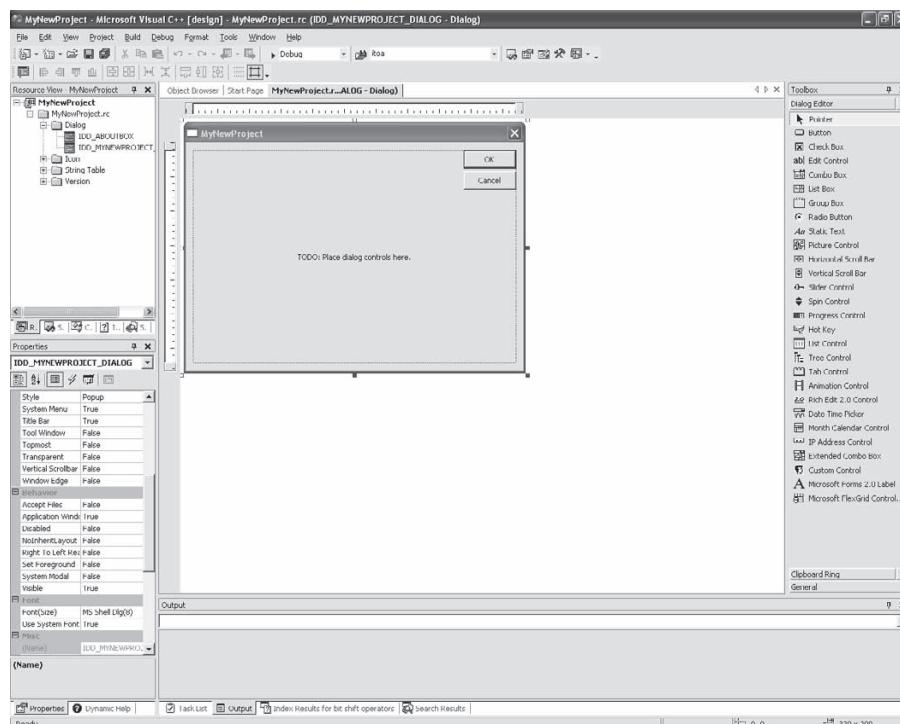
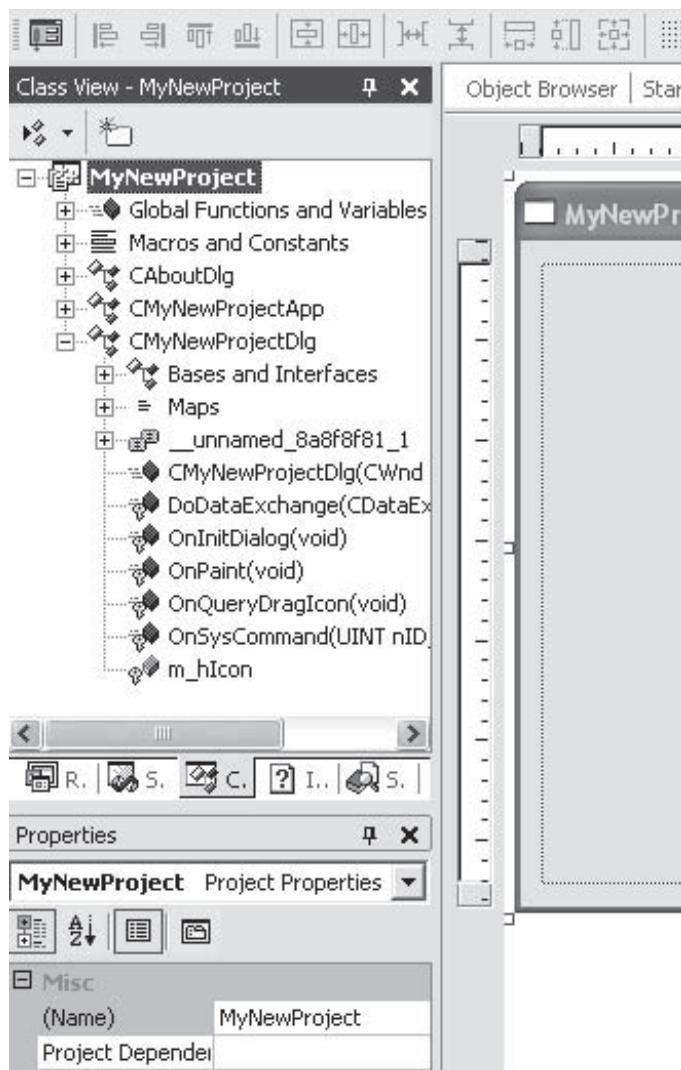


FIGURA A-5 Captura de pantalla de MyNewProject en el editor de recursos.

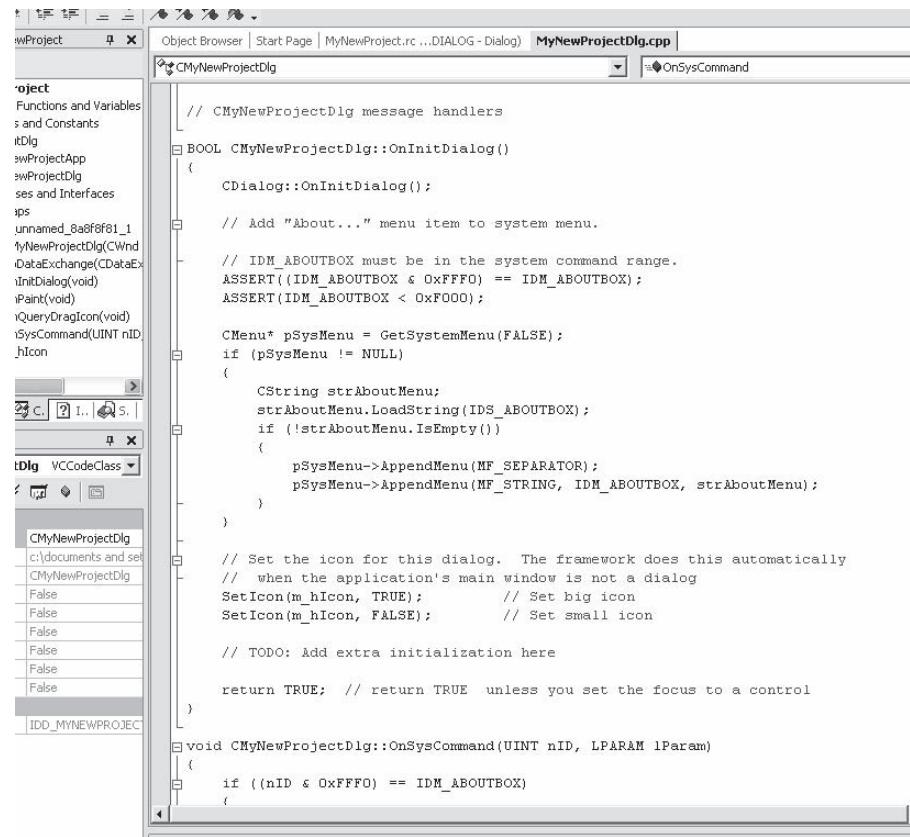
FIGURA A-6 La vista de clases de MyNewProject.



Después de completar estos cinco pasos, aparecerá la pantalla que se muestra en la figura A-5. Ésta es la aplicación basada en cuadro de diálogo que viene en el editor de recursos. En este punto está listo para empezar a colocar objetos en el formulario del cuadro de diálogo. La pantalla que usted vea y la que aparece en la figura podrían variar un poco, según lo determinado por el perfil en la página principal de Visual Studio. Para modificarlo, haga clic en la ficha My Profile de la figura A-1.

La ventana en la esquina superior izquierda de la figura A-5 tiene una ficha debajo de ella. Estas fichas se utilizan para seleccionar distintas vistas del programa. Observe que la ficha R está resaltada, ya que la vista es Resource View. Si hace clic en la ficha C aparecerá la Vista de clases, como se muestra en la figura A-6. En esta figura se hizo clic en los signos positivos al lado de MyNewProject y CMynewProjectDlg para expandir la clase dlg (diálogo). Aquí es donde se encuentra el software para el cuadro de diálogo, según la instalación que realiza Visual Studio para esta aplicación. Muchos de los programas en el libro se manejan con la función OnInitDialog para configurar el cuadro de diálogo.

La figura A-7 muestra el software en la función OnInitDialog. Para ver esta función, haga doble clic en OnInitDialog en la vista de clases. El marco de trabajo MFC llamará a la función OnInitDialog antes de mostrar el cuadro de diálogo en la pantalla. Todo el software que se agregue para la inicialización deberá colocarse cerca de la parte inferior de la figura A-7, donde aparece un comentario TODO:. Una vez que llegue a este punto, estará listo para introducir y ejecutar cualquiera de los programas en lenguaje ensamblador que vienen en el libro.



```

// CMyNewProjectDlg message handlers

BOOL CMyNewProjectDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.
    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFFF) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xFOOO);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);          // Set small icon

    // TODO: Add extra initialization here

    return TRUE; // return TRUE unless you set the focus to a control
}

void CMyNewProjectDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFFF) == IDM_ABOUTBOX)
    {
}

```

FIGURA A-7 Contenido de la función OnInitDialog.

APÉNDICE B

Resumen del conjunto de instrucciones

Este resumen del conjunto de instrucciones contiene un listado alfabético completo de todo el conjunto de instrucciones de los microprocesadores 8086 al Pentium 4. Las instrucciones del coprocesador y MMX se listan en el capítulo 14, por lo cual no las repetiremos en este apéndice. Las instrucciones SIMD aparecen al final de este apéndice, después del resumen del conjunto principal de instrucciones.

Cada una de las entradas de instrucciones lista el código de operación mnemónico más una breve descripción del propósito de la instrucción. Además, se lista la codificación en lenguaje máquina binario de cada instrucción y de cualquier otra información requerida para formar la instrucción, como el desplazamiento o los datos inmediatos. A la derecha de cada una de las versiones en lenguaje máquina binario de la instrucción se listan los bits de bandera y cualquier cambio que pudiera ocurrir para la instrucción. Las banderas se describen de la siguiente manera: un espacio en blanco indica que no hay efecto o cambio; una ? indica un cambio con un resultado impredecible, un * indica un cambio con un resultado predecible; un 1 indica que la bandera se activa y un 0 indica que la bandera se borra. Si los bits de bandera ODITSZAPC no se muestran con una instrucción, significa que esa instrucción no modifica ninguna de esas banderas.

Antes del listado de instrucciones presentaremos cierta información sobre la configuración de los bits en las versiones en lenguaje máquina de las instrucciones. La tabla B-1 lista los bits modificadores, que se codifican como 00 en el listado de instrucciones.

La tabla B-2 lista los modos de direccionamiento de memoria disponibles mediante el uso de una codificación de registro de campo mmm. Esta tabla se aplica a todas las versiones del microprocesador, siempre y cuando el modo de operación sea de 16 bits.

La tabla B-3 lista las selecciones de registro que proporciona el campo rrr en una instrucción. Esta tabla incluye las selecciones de registro para los registros de 8, 16 y 32 bits.

La tabla B-4 lista la asignación del bit de registro de segmento (rrr) que se incluye en las instrucciones MOV, PUSH y POP.

TABLA B-1 Los bits modificadores, que se codifican como 00 en el listado de instrucciones.

00	Función
00	Si mmm = 110 un desplazamiento va después del código de operación; en caso contrario, no se utiliza desplazamiento.
01	Un desplazamiento de 8 bits con signo va después del código de operación.
10	Un desplazamiento de 16 o 32 bits con signo va después del código de operación.
11	mmm especifica un registro, en lugar de un modo de direccionamiento.

TABLA B-2 La descripción del campo de registro/memoria (mmm) de 16 bits.

<i>mmm</i>	<i>16 bits</i>
000	DS:[BX+SI]
001	DS:[BX+DI]
010	SS:[BP+SI]
011	SS:[BP+DI]
100	DS:[SI]
101	DS:[DI]
110	SS:[BP]
111	DS:[BX]

TABLA B-3 El campo de registros (rrr).

<i>rrr</i>	<i>W = 0</i>	<i>W = 1 (16 bits)</i>	<i>W = 1 (32 bits)</i>
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

TABLA B-4 Asignaciones del campo de registro (rrr) para los registros de segmento.

<i>rrr</i>	<i>Registro de segmento</i>
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS

Cuando se utilizan los microprocesadores del 80386 al Pentium 4, cambian algunas de las definiciones que se proporcionan en las tablas B-1 a B-3. En las tablas B-5 y B-6 consulte estos cambios según su aplicación en los microprocesadores 80386 al Pentium 4.

TABLA B-5 Registros índice especificados por rrr cuando los microprocesadores del 80386 al Pentium 4 se operan en modo de 32 bits.

<i>rrr</i>	<i>Registro índice</i>
000	DS:[EAX]
001	DS:[ECX]
010	DS:[EDX]
011	DS:[EBX]
100	(vea la tabla B-6)
101	SS:[EBP]
110	DS:[ESI]
111	DS:[EDI]

TABLA B-6 Posibles combinaciones de oo, mmm y rrr para los microprocesadores del 80386 al Pentium 4 que utilizan direccionamiento de 32 bits.

<i>oo</i>	<i>mmm</i>	<i>rrr</i> (<i>basado en byte de índice escalado</i>)	<i>Modo de direccionamiento</i>
00	000	—	DS:[EAX]
00	001	—	DS:[ECX]
00	010	—	DS:[EDX]
00	011	—	DS:[EBX]
00	100	000	DS:[EAX+índice escalado]
00	100	001	DS:[ECX+índice escalado]
00	100	010	DS:[EDX+índice escalado]
00	100	011	DS:[EBX+índice escalado]
00	100	100	SS:[ESP+índice escalado]
00	100	101	DS:[desp32+índice escalado]
00	100	110	DS:[ESI+índice escalado]
00	100	111	DS:[EDI+índice escalado]
00	101	—	DS:desp32
00	110	—	DS:[ESI]
00	111	—	DS:[EDI]
01	000	—	DS:[EAX+desp8]
01	001	—	DS:[ECX+desp8]
01	010	—	DS:[EDX+desp8]
01	011	—	DS:[EBX+desp8]
01	100	000	DS:[EAX+índice escalado+desp8]
01	100	001	DS:[ECX+índice escalado+desp8]
01	100	010	DS:[EDX+índice escalado+desp8]
01	100	011	DS:[EBX+índice escalado+desp8]
01	100	100	SS:[ESP+índice escalado+desp8]
01	100	101	SS:[EBP+índice escalado+desp8]
01	100	110	DS:[ESI+índice escalado+desp8]
01	100	111	DS:[EDI+índice escalado+desp8]
01	101	—	SS:[EBP+desp8]
01	110	—	DS:[ESI+desp8]
01	111	—	DS:[EDI+desp8]
10	000	—	DS:[EAX+desp32]
10	001	—	DS:[ECX+desp32]
10	010	—	DS:[EDX+desp32]
10	011	—	DS:[EBX+desp32]
10	100	000	DS:[EAX+índice escalado+desp32]
10	100	001	DS:[ECX+índice escalado+desp32]
10	100	010	DS:[EDX+índice escalado+desp32]
10	100	011	DS:[EBX+índice escalado+desp32]
10	100	100	SS:[ESP+índice escalado+desp32]
10	100	101	SS:[EBP+índice escalado+desp32]
10	100	110	DS:[ESI+índice escalado+desp32]
10	100	111	DS:[EDI+índice escalado+desp32]
10	101	—	SS:[EBP+desp32]
10	110	—	DS:[ESI+desp32]
10	111	—	DS:[EDI+desp32]

Nota: desp8 = desplazamiento de 8 bits y desp32 = desplazamiento de 32 bits.

Para utilizar los modos de direccionamiento de índice escalado que se listan en la tabla B-6, se codifican 00 y mmm en el segundo byte del código de operación. Por lo general, el byte de índice escalado es el tercer byte y contiene tres campos. Los dos bits de más a la izquierda determinan el factor de escala (00 = $\times 1$, 01 = $\times 2$, 10 = $\times 4$ o 11 = $\times 8$). Los siguientes tres bits hacia la derecha traen el número de registro de índice escalado (éste se obtiene de la tabla B-5). Los tres bits de más a la derecha son del campo mmm que se lista en la tabla B-6. Por ejemplo, la instrucción MOV AL[EBX+2*ECX] tiene un byte de índice escalado de 01 001 011, en donde 01 = X_2 , 001 = ECX y 011 = EBX.

Algunas instrucciones utilizan un prefijo para cambiar el segmento predeterminado o para redefinir el modo de instrucción. La tabla B-7 lista los prefijos de sustitución de segmento y modo de instrucción que se adjuntan al principio de una instrucción si se manejan para formarla. Por ejemplo, la instrucción MOV AL,ES:[BX] emplea el segmento extra debido al prefijo de sustitución ES:.

En los microprocesadores 8086 y 8088, el cálculo de la dirección efectiva requiere ciclos de reloj adicionales que se agregan a los tiempos en el resumen del conjunto de instrucciones. Estos tiempos adicionales se listan en la tabla B-8. En los microprocesadores 80286 al Pentium 4 no se agregan esos tiempos. Tenga en cuenta que el resumen del conjunto de instrucciones no incluye los tiempos de reloj para los microprocesadores del Pentium Pro al Pentium 4. Intel no ha publicado estos tiempos y ha decidido que la instrucción RDTSC puede usarse para hacer que el microprocesador cuente el número de ciclos de reloj requeridos para una aplicación dada. Aun cuando los tiempos no aparecen para estos nuevos microprocesadores, son muy similares a los del Pentium, los cuales llegan a utilizarse como una guía.

TABLA B-7 Prefijos de sustitución.

Byte de prefijo	Propósito
26H	Sustitución de segmento ES:
2EH	Sustitución de segmento CS:
36H	Sustitución de segmento SS:
3EH	Sustitución de segmento DS:
64H	Sustitución de segmento FS:
65H	Sustitución de segmento GS:
66H	Sustitución del modo de instrucción de operando de memoria.
67H	Sustitución del modo de instrucción de operando de registro.

TABLA B-8 Cálculos de dirección efectiva para los microprocesadores 8086 y 8088.

Tipo	Ciclos de reloj	Instrucción de ejemplo
Base o índice	5	MOV CL,[DI]
Desplazamiento	3	MOV AL,DATOS1
Base más índice	7	MOV AL,[BP+SI]
Desplazamiento más base o índice	9	MOV DH,[DI+20H]
Desplazamiento más índice más desplazamiento	11	MOV CL,[BX+DI+2]
Sustitución de segmento	ea + 2	MOV AL,ED:[DI]

RESUMEN DEL CONJUNTO DE INSTRUCCIONES

AAA	Ajuste ASCII de AL antes de suma									
00110111		O	D	I	T	S	Z	A	P	C
Ejemplo		?				?	?	*	?	*
						Microprocesador				
										Ciclos de reloj
AAA		8086				8				
		8088				8				
		80286				3				
		80386				4				
		80486				3				
		Pentium-Pentium 4				3				
AAD	Ajuste ASCII de AX antes de división									
11010101 00001010		O	D	I	T	S	Z	A	P	C
Ejemplo		?				*	*	?	*	*
						Microprocesador				
										Ciclos de reloj
AAD		8086				60				
		8088				60				
		80286				14				
		80386				19				
		80486				14				
		Pentium-Pentium 4				10				
AAM	Ajuste ASCII de AX antes de multiplicación									
11010100 00001010		O	D	I	T	S	Z	A	P	C
Ejemplo		?				*	*	?	*	*
						Microprocesador				
										Ciclos de reloj
AAM		8086				83				
		8088				83				
		80286				16				
		80386				17				
		80486				15				
		Pentium-Pentium 4				18				

AAS		Ajuste ASCII de AL después de resta								
00111111		O	D	I	T	S	Z	A	P	C
Ejemplo		?		?	*	?	?	*	*	
						Microprocesador				Ciclos de reloj
AAS		8086				8				
		8088				8				
		80286				3				
		80386				4				
		80486				3				
		Pentium-Pentium 4				3				
ADC		Suma con acarreo								
000100dw oorrrmmm desp		O	D	I	T	S	Z	A	P	C
Formato	Ejemplos	*		*	*	*	*	*	*	*
						Microprocesador				Ciclos de reloj
ADC reg,reg	ADC AX,BX ADC AL,BL ADC EAX,EBX ADC CX,SI ADC ESI,EDI	8086				3				
		8088				3				
		80286				3				
		80386				3				
		80486				1				
		Pentium-Pentium 4				1 o 3				
ADC mem,reg	ADC DATOSY,AL ADC LISTA,SI ADC DATOS2[DI],CL ADC [EAX],BL ADC [EBX+2*ECX],EDX	8086				16 + ea				
		8088				24 + ea				
		80286				7				
		80386				7				
		80486				3				
		Pentium-Pentium 4				1 o 3				

ADC reg,mem	ADC BL,DATOS1 ADC SI,LISTA1 ADC CL,DATOS2[SI] ADC CX,[ESI] ADC ESI,[2*ECX]	8086	9 + ea
		8088	13 + ea
		80286	7
		80386	6
		80486	2
		Pentium-Pentium 4	1 o 2
100000sw oo010mmm desp datos			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
ADC reg,imm	ADC CX,3 ADC DI,1AH ADC DL,34H ADC EAX,12345 ADC CX,1234H	8086	4
		8088	4
		80286	3
		80386	2
		80486	1
		Pentium-Pentium 4	1 o 3
ADC mem,imm	ADC DATOS4,33 ADC LISTA,'A' ADC DATOS3[DI],2 ADC BYTE PTR[EBX],3 ADC WORD PTR[DI],669H	8086	17 + ea
		8088	23 + ea
		80286	7
		80386	7
		80486	3
		Pentium-Pentium 4	1 o 3
ADC acu,imm	ADC AX,3 ADC AL,1AH ADC AH,34H ADC EAX,2 ADC AL,'Z'	8086	4
		8088	4
		80286	3
		80386	2
		80486	1
		Pentium-Pentium 4	1

ADD		Suma								
		O	D	I	T	S	Z	A	P	C
Formato	Ejemplos	Microprocesador					Ciclos de reloj			
ADD reg,reg	000000dw oorrrmmm desp	*	*	*	*	*	*	*	*	*
	ADD AX,BX	8086					3			
	ADD AL,BL	8088					3			
	ADD EAX,EBX	80286					2			
	ADD CX,SI	80386					2			
	ADD ESI,EDI	80486					1			
ADD mem,reg	Pentium-Pentium 4						1 o 3			
	ADD DATOSY,AL	8086					16 + ea			
	ADD LISTA,SI	8088					24 + ea			
	ADD DATOS6[DI],CL	80286					7			
	ADD [EAX],CL	80386					7			
	ADD [EDX+4*ECX],EBX	80486					3			
	Pentium-Pentium 4						1 o 3			
ADD reg,mem	100000sw oo000mmm desp datos	8086					9 + ea			
	ADD SI,LISTA3	8088					13 + ea			
	ADD CL,DATOS2[DI]	80286					7			
	ADD CX,[EDI]	80386					6			
	ADD ESI,[ECX+200H]	80486					2			
	Pentium-Pentium 4						1 o 2			
Formato		Microprocesador					Ciclos de reloj			
ADD reg,imm	ADD CX,3	8086					4			
	ADD DI,1AH	8088					4			
	ADD DL,34H	80286					3			
	ADD EDX,1345H	80386					2			
	ADD CX,1834H	80486					1			
	Pentium-Pentium 4						1 o 3			

ADC mem,imm	ADD DATOS4,33 ADD LISTA,'A' ADD DATOS3[DI],2 ADD BYTE PTR[EBX],3 ADD WORD PTR[DI],669H	8086	17 + ea	
		8088	23 + ea	
		80286	7	
		80386	7	
		80486	3	
		Pentium-Pentium 4	1 o 3	
ADD acu,imm	ADD AX,3 ADD AL,1AH ADD AH,34H ADD EAX,2 ADD AL,'Z'	8086	4	
		8088	4	
		80286	3	
		80386	2	
		80486	1	
		Pentium-Pentium 4	1	
AND AND Lógico				
001000dw oorrrmmm desp		O D I T	S Z A P C	
Formato		0	* * ? * 0	
Ejemplos		Microprocesador	Ciclos de reloj	
AND reg,reg	AND CX,BX AND DL,BL AND ECX,EBX AND BP,SI AND EDX,EDI	8086	3	
		8088	3	
		80286	2	
		80386	2	
		80486	1	
		Pentium-Pentium 4	1 o 3	
AND mem,reg	ADD BIT,AL AND LISTA,DI AND DATOSZ[BX],CL AND [EAX],BL AND [ESI+4*ECX],EDX	8086	16 + ea	
		8088	24 + ea	
		80286	7	
		80386	7	
		80486	3	
		Pentium-Pentium 4	1 o 3	

AND reg,mem	AND BL,DATOSW AND SI,LISTA AND CL,DATOSQ[SI] AND CX,[EAX] AND ESI,[ECX+43H]	8086	9 + ea	
		8088	13 + ea	
		80286	7	
		80386	6	
		80486	2	
		Pentium-Pentium 4	1 o 2	
		100000sw oo100mmm desp datos		
Formato		Ejemplos	Microprocesador	
			Ciclos de reloj	
AND reg,imm	AND BP,1 AND DI,10H AND DL,34H AND EBP,1345H AND SP,1834H	8086	4	
		8088	4	
		80286	3	
		80386	2	
		80486	1	
		Pentium-Pentium 4	1 o 3	
		AND DATOS4,33 AND LISTA,'A' AND DATOS3[DI],2 AND BYTE PTR[EBX],3 AND DWORD PTR[DI],66H		
AND mem,imm	AND DATOS4,33 AND LISTA,'A' AND DATOS3[DI],2 AND BYTE PTR[EBX],3 AND DWORD PTR[DI],66H	8086	17 + ea	
		8088	23 + ea	
		80286	7	
		80386	7	
		80486	3	
		Pentium-Pentium 4	1 o 3	
		AND AX,3 AND AL,1AH AND AH,34H AND EAX,2 AND AL,r'		
AND acu,imm	AND AX,3 AND AL,1AH AND AH,34H AND EAX,2 AND AL,r'	8086	4	
		8088	4	
		80286	3	
		80386	2	
		80486	1	
		Pentium-Pentium 4	1	

ARPL Ajusta nivel de privilegio solicitado			
01100011 oorrrmmm desp		O D I T	S Z A P C *
Formato	Ejemplos	Microprocesador	Ciclos de reloj
ARPL reg,reg	ARPL AX,BX	8086	—
	ARPL BX,SI	8088	—
	ARPL AX,DX	80286	10
	ARPL BX,AX	80386	20
	ARPL SI,DI	80486	9
		Pentium-Pentium 4	7
ARPL mem,reg	ARPL DATOSY,AX	8086	—
	ARPL LISTA,DI	8088	—
	ARPL DATOS3[DI],CX	80286	11
	ARPL [EBX],AX	80386	21
	ARPL [EDX+4*ECX],BP	80486	9
		Pentium-Pentium 4	7
BOUND Compara arreglo con el límite			
01100010 oorrrmmm desp			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
BOUND reg,mem	BOUND AX,BETS	8086	—
	BOUND BP,LISTAG	8088	—
	BOUND CX,DATOSX	80286	13
	BOUND BX,[DI]	80386	10
	BOUND SI,[BX+2]	80486	7
		Pentium-Pentium 4	8

BSF		Exploración de bits hacia delante					
		00001111 10111100 oorrrmmm desp			O D I T	S Z A P C	
Formato	Ejemplos				?	? * ? ? ?	
BSF reg,reg	BSF AX,BX	8086	—				
	BSF BX,SI	8088	—				
	BSF EAX,EDX	80286	—				
	BSF EBX,EAX	80386	10 + 3n				
	BSF SI,DI	80486	6-42				
		Pentium-Pentium 4	6-42				
BSF reg,mem	BSF AX,DATOSY	8086	—				
	BSF SI,LISTA	8088	—				
	BSF CX,DATOS3[DI]	80286	—				
	BSF EAX,[EBX]	80386	10 + 3n				
	BSF EBP,[EDX+4*ECX]	80486	7-43				
		Pentium-Pentium 4	6-43				
BSR		Exploración de bits inversa					
		00001111 10111101 oorrrmmm desp			O D I T	S Z A P C	
Formato	Ejemplos				?	? * ? ? ?	
BSR reg,reg	BSR AX,BX	8086	—				
	BSR BX,SI	8088	—				
	BSR EAX,EDX	80286	—				
	BSR EBX,EAX	80386	10 + 3n				
	BSR SI,DI	80486	6-103				
		Pentium-Pentium 4	7-71				

BSR reg,mem	BSR AX,DATOSY BSR SI,LISTA BSR CX,DATOS3[DI] BSR EAX,[EBX] BSR EBP,[EDX+4*ECX]	8086	—	
		8088	—	
		80286	—	
		80386	10 + 3n	
		80486	7-104	
		Pentium-Pentium 4	7-72	
BSWAP Intercambio de bytes				
00001111 11001rrr				
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
BSWAP reg32	BSWAP EAX BSWAP EBX BSWAP EDX BSWAP ECX BSWAP ESI	8086	—	
		8088	—	
		80286	—	
		80386	—	
		80486	1	
		Pentium-Pentium 4	1	
BT Prueba de bits				
00001111 10111010 oo100mmm desp datos				
O D I T S Z A P C * *				
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
BT reg,imm8	BT AX,2 BT CX,4 BT BP,10H BT CX,8 BT BX,2	8086	—	
		8088	—	
		80286	—	
		80386	3	
		80486	3	
		Pentium-Pentium 4	4	

BT mem,imm8	BT DATOS1,2	8086	–
	BT LISTA,2	8088	–
	BT DATOS2[DI],3	80286	–
	BT [EAX],1	80386	6
	BT RANA,6	80486	3
		Pentium-Pentium 4	4
	00001111 10100011 desp		
Formato	Ejemplos	Microprocesador	Ciclos de reloj
BT reg,reg	BT AX,CX	8086	–
	BT CX,DX	8088	–
	BT BP,AX	80286	–
	BT SI,CX	80386	3
	BT EAX,EBX	80486	3
		Pentium-Pentium 4	4 o 9
BT mem,reg	BT DATOS4,AX	8086	–
	BT LISTA,BX	8088	–
	BT DATOS3[DI],CX	80286	–
	BT [EBX],DX	80386	12
	BT [DI],DI	80486	8
		Pentium-Pentium 4	4 o 9

BTC		Prueba de bits y complemento			
		00001111 10111010 oo111mmm desp datos		O D I T	S Z A P C
Formato	Ejemplos			Microprocesador	Ciclos de reloj
BTC reg,imm8	BTC AX,2	8086	—		
	BTC CX,4	8088	—		
	BTC BP,10H	80286	—		
	BTC CX,8	80386	6		
	BTC BX,2	80486	6		
		Pentium-Pentium 4	7 u 8		
BTC mem,imm8	BTC DATOS1,2	8086	—		
	BTC LISTA,2	8088	—		
	BTC DATOS2[DI],3	80286	—		
	BTC [EAX],1	80386	7 u 8		
	BTC RANA,6	80486	8		
		Pentium-Pentium 4	8		
00001111 10111011 desp					
Formato	Ejemplos			Microprocesador	Ciclos de reloj
BTC reg,reg	BTC AX,CX	8086	—		
	BTC CX,DX	8088	—		
	BTC BP,AX	80286	—		
	BTC SI,CX	80386	6		
	BTC EAX,EBX	80486	6		
		Pentium-Pentium 4	7 o 13		
BTC mem,reg	BTC DATOS4,AX	8086	—		
	BTC LISTA,BX	8088	—		
	BTC DATOS3[DI],CX	80286	—		
	BTC [EBX],DX	80386	13		
	BTC [DI],DI	80486	13		
		Pentium-Pentium 4	7 o 13		

BTR		Prueba de bits y reinicio			
		00001111 10111010 oo110mmm desp datos		O D I T	S Z A P C
Formato	Ejemplos			Microprocesador	Ciclos de reloj
BTR reg,imm8	BTR AX,2	8086	—		*
	BTR CX,4	8088	—		
	BTR BP,10H	80286	—		
	BTR CX,8	80386	6		
	BTR BX,2	80486	6		
		Pentium-Pentium 4	7 u 8		
BTR mem,imm8	BTR DATOS1,2	8086	—		
	BTR LISTA,2	8088	—		
	BTR DATOS2[DI],3	80286	—		
	BTR [EAX],1	80386	8		
	BTR RANA,6	80486	8		
		Pentium-Pentium 4	7 u 8		
00001111 10110011 desp					
Formato	Ejemplos			Microprocesador	Ciclos de reloj
BTR reg,reg	BTR AX,CX	8086	—		
	BTR CX,DX	8088	—		
	BTR BP,AX	80286	—		
	BTR SI,CX	80386	6		
	BTR EAX,EBX	80486	6		
		Pentium-Pentium 4	7 o 13		
BTR mem,reg	BTR DATOS4,AX	8086	—		
	BTR LISTA,BX	8088	—		
	BTR DATOS3[DI],CX	80286	—		
	BTR [EBX],DX	80386	13		
	BTR [DI],DI	80486	13		
		Pentium-Pentium 4	7 o 13		

BTS		Prueba de bits y activación			
00001111 10111010 oo101mmmm desp datos		O	D	I	T
Formato	Ejemplos	Microprocesador			Ciclos de reloj
BTS reg,imm8	BTS AX,2	8086	–	–	–
	BTS CX,4	8088	–	–	–
	BTS BP,10H	80286	–	–	–
	BTS CX,8	80386	6	–	–
	BTS BX,2	80486	6	–	–
		Pentium-Pentium 4	7 u 8	–	–
BTS mem,imm8	BTS DATOS1,2	8086	–	–	–
	BTS LISTA,2	8088	–	–	–
	BTS DATOS2[DI],3	80286	–	–	–
	BTS [EAX],1	80386	8	–	–
	BTS RANA,6	80486	8	–	–
		Pentium-Pentium 4	7 u 8	–	–
00001111 10101011 desp					
Formato	Ejemplos	Microprocesador			Ciclos de reloj
BTS reg,reg	BTS AX,CX	8086	–	–	–
	BTS CX,DX	8088	–	–	–
	BTS BP,AX	80286	–	–	–
	BTS SI,CX	80386	6	–	–
	BTS EAX,EBX	80486	6	–	–
		Pentium-Pentium 4	7 o 13	–	–
BTS mem,reg	BTS DATOS4,AX	8086	–	–	–
	BTS LISTA,BX	8088	–	–	–
	BTS DATOS3[DI],CX	80286	–	–	–
	BTS [EBX],DX	80386	13	–	–
	BTS [DI],DI	80486	13	–	–
		Pentium-Pentium 4	7 o 13	–	–

CALL Llamada a procedimiento (subrutina)			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
11101000 desp Formato	CALL etiqueta (cercana) CALL POR_DIVERSION CALL CASA CALL ET CALL ESPERA CALL ALGUIEN	8086 8088 80286 80386 80486 Pentium-Pentium 4	19 23 7 3 3 1
10011010 desp Formato	CALL FAR PTR FECHAS CALL QUE CALL DONDE CALL FARSA CALL QUIEN	8086 8088 80286 80386 80486 Pentium-Pentium 4	28 36 13 17 18 4
11111111 oo010mmm Formato	CALL AX CALL BX CALL CX CALL DI CALL SI	8086 8088 80286 80386 80486 Pentium-Pentium 4	16 20 7 7 5 2

CALL mem (cercana)	CALL DIRECCION CALL NEAR PTR[DI] CALL DATOS1 CALL RANA CALL AMI_AHORA	8086	21 + ea
		8088	29 + ea
		80286	11
		80386	10
		80486	5
		Pentium-Pentium 4	2
11111111 00011mmm			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
CALL mem (lejana)	CALL FAR_LISTA[SI] CALL DESDE_AQUI CALL HACIA_ALLA CALL SIXX CALL OCT	8086	16
		8088	20
		80286	7
		80386	7
		80486	5
		Pentium-Pentium 4	2
CBW Convierte byte en palabra (AL \Rightarrow AX)			
10011000			
Ejemplo		Microprocesador	Ciclos de reloj
CBW		8086	2
		8088	2
		80286	2
		80386	3
		80486	3
		Pentium-Pentium 4	3

CDQ	Convierte doble palabra en palabra cuádruple (EAX \Rightarrow EDX:EAX)			
11010100 00001010 Ejemplo				Microprocesador Ciclos de reloj
CDQ	8086	–		
	8088	–		
	80286	–		
	80386	2		
	80486	2		
	Pentium-Pentium 4	2		
CLC	Borra bandera de acarreo			
11111000 Ejemplo	O D I T 0	S Z A P C		
CLC	8086	2		
	8088	2		
	80286	2		
	80386	2		
	80486	2		
	Pentium-Pentium 4	2		
CLD	Borra bandera de dirección			
11111100 Ejemplo	O D I T 0	S Z A P C		
CLD	8086	2		
	8088	2		
	80286	2		
	80386	2		
	80486	2		
	Pentium-Pentium 4	2		

CLI	Borra bandera de interrupción				
11111010		O	D	I	T
Ejemplo		0			
		Microp			Ciclos de reloj
CLI		8086			2
		8088			2
		80286			3
		80386			3
		80486			5
		Pentium-Pentium 4			7
CLTS	Borra bandera de tarea conmutada (CR0)				
00001111 00000110		O	D	I	T
Ejemplo		0			
		Microp			Ciclos de reloj
CLTS		8086			–
		8088			–
		80286			2
		80386			5
		80486			7
		Pentium-Pentium 4			10
CMC	Complementa bandera de acarreo				
10011000		O	D	I	T
Ejemplo		0			
		Microp			Ciclos de reloj
CMC		8086			2
		8088			2
		80286			2
		80386			2
		80486			2
		Pentium-Pentium 4			2

CMOVcondicion Movimiento condicional				
Formato		Ejemplos	Microprocesador	Ciclos de reloj
CMOVcc reg,mem		CMOVNZ AX,RANA CMOVC EAX,[EDI] CMOVNC BX,DATOS1 CMOVP EBX,ESPERA CMOVNE DI,[SI]	8086	-
			8088	-
			80286	-
			80386	-
			80486	-
			Pentium-Pentium 4	-
Condición				
Códigos	Mnemónico	Bandera	Descripción	
0000	CMOVO	O = 1	Mueve si hay desbordamiento	
0001	CMOVNO	O = 0	Mueve si no hay desbordamiento	
0010	CMOVB	C = 1	Mueve si está por debajo	
0011	CMOVAE	C = 0	Mueve si está por encima o si es igual	
0100	CMOVE	Z = 1	Mueve si es igual/cero	
0101	CMOVNE	Z = 0	Mueve si no es igual/cero	
0110	CMOVBE	C = 1 + Z = 1	Mueve si está por debajo o si es igual	
0111	CMOVA	C = 0 • Z = 0	Mueve si está por encima	
1000	CMOVS	S = 1	Mueve si hay signo	
1001	CMOVNS	S = 0	Mueve si no hay signo	
1010	CMOVP	P = 1	Mueve si hay paridad	
1011	CMOVNP	P = 0	Mueve si no hay paridad	
1100	CMOVL	S • O	Mueve si es menor que	
1101	CMOVGE	S = 0	Mueve si es mayor o igual que	
1110	CMOVLE	Z = 1 + S • O	Mueve si es menor o igual que	
1111	CMOVG	Z = 0 + S = O	Mueve si es mayor que	
CMP Comparación				
001110dw oorrrmmm desp			O D I T	S Z A P C
			*	* * * * *
Formato		Ejemplos	Microprocesador	Ciclos de reloj
CMP reg,reg		CMP AX,BX CMP AL,BL CMP EAX,EBX CMP CX,SI CMP ESI,EDI	8086	3
			8088	3
			80286	2
			80386	2
			80486	1
			Pentium-Pentium 4	1 o 2

CMP mem,reg	CMP DATOS,Y,AL CMP LISTA,SI CMP DATOS6[DI],CL CMP [EAX],CL CMP [EDX+4*ECX],EBX	8086	9 + ea
		8088	13 + ea
		80286	7
		80386	5
		80486	2
		Pentium-Pentium 4	1 o 2
CMP reg,mem	CMP BL,DATOS2 CMP SI,LISTA3 CMP CL,DATOS2[DI] CMP CX,[EDI] CMP ESI,[ECX+200H]	8086	9 + ea
		8088	13 + ea
		80286	6
		80386	6
		80486	2
		Pentium-Pentium 4	1 o 2
100000sw oo111mmm desp datos			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
CMP reg, imm	CMP CX,3 CMP DI,1AH CMP DL,34H CMP EDX,1345H CMP CX,1834H	8086	4
		8088	4
		80286	3
		80386	2
		80486	1
		Pentium-Pentium 4	1 o 2
CMP mem, imm	CMP DATOS,3 CMP BYTE PTR[EDI],1AH CMP PAPE,34H CMP LISTA,'A' CMP RANITA,1834H	8086	10 + ea
		8088	14 + ea
		80286	6
		80386	5
		80486	2
		Pentium-Pentium 4	1 o 2

0001111w datos			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
CMP cue,imm	CMP AX,3	8086	4
	CMP AL,1AH	8088	4
	CMP AH,34H	80286	3
	CMP EAX,1345H	80386	2
	CMP AL,'Y'	80486	1
		Pentium-Pentium 4	1
CMPS Comparación de cadenas			
1010011w		O D I T	S Z A P C
		*	* * * * *
Formato	Ejemplos	Microprocesador	Ciclos de reloj
CMPSB CMPSW CMPSD	CMPSB	8086	32
	CMPSW	8088	30
	CMPSD	80286	8
	CMPSB DATOS1,DATOS2	80386	10
	REPE CMPSB	80486	8
	REPNE CMPSW	Pentium-Pentium 4	5
CMPXCHG Comparación e intercambio			
00001111 1011000w 11rrr		O D I T	S Z A P C
		*	* * * * *
Formato	Ejemplos	Microprocesador	Ciclos de reloj
CMPXCHG reg,reg	CMPXCHG EAX,EBX	8086	—
	CMPXCHG ECX,EDX	8088	—
		80286	—
		80386	—
		80486	6
		Pentium-Pentium 4	6

0001111w datos				
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
CMPXCHG mem,reg	CMPXCHG DATOSD,EAX CMPXCHG DATOS2,EDI	8086	—	
		8088	—	
		80286	—	
		80386	—	
		80486	7	
		Pentium-Pentium 4	6	
CMPXCHG8B Comparación e intercambio de 8 bytes				
00001111 11000111 oorrrmmm		O D I T	S Z A P C *	
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
CMPXCHG8B mem64	CMPXCHG8B DATOS3	8086	—	
		8088	—	
		80286	—	
		80386	—	
		80486	—	
		Pentium-Pentium 4	10	
CPUID Código de identificación de CPU				
00001111 10100010		Microprocesador	Ciclos de reloj	
Ejemplo				
CPUID		8086	—	
		8088	—	
		80286	—	
		80386	—	
		80486	—	
		Pentium-Pentium 4	14	

CWD	Convierte palabra en doble palabra (AX \Rightarrow DX:AX)				
10011000 Ejemplo		Microprocesador	Ciclos de reloj		
CWD	8086	5			
	8088	5			
	80286	2			
	80386	2			
	80486	3			
	Pentium-Pentium 4	2			
CWDE	Convierte palabra en doble palabra extendida (AX \Rightarrow EAX)				
10011000 Ejemplo		Microprocesador	Ciclos de reloj		
CWDE	8086	–			
	8088	–			
	80286	–			
	80386	3			
	80486	3			
	Pentium-Pentium 4	3			
DAA	Ajuste decimal de AL después de la suma				
00100111 Ejemplo	O D I T S Z A P C ? * * * * *	Microprocesador	Ciclos de reloj		
DAA	8086	4			
	8088	4			
	80286	3			
	80386	4			
	80486	2			
	Pentium-Pentium 4	3			

DAS		Ajuste decimal de AL después de la resta												
		O	D	I	T	S	Z	A	P	C				
00101111					?		*	*	*	*				
Ejemplo						Microprocesador				Ciclos de reloj				
DAS		8086				4								
		8088				4								
		80286				3								
		80386				4								
		80486				2								
		Pentium-Pentium 4				3								
DEC		Decremento												
1111111w oo001mmm desp						O	D	I	T	S Z A P C				
						*				*	*	*	*	*
Formato	Ejemplos					Microprocesador				Ciclos de reloj				
DEC reg8	DEC BL DEC BH DEC CL DEC DH DEC AH	8086				3								
		8088				3								
		80286				2								
		80386				2								
		80486				1								
		Pentium-Pentium 4				1 o 3								
DEC mem	DEC DATOSY DEC LISTA DEC DATOS6[DI] DEC BYTE PTR [BX] DEC WORD PTR [EBX]	8086				15 + ea								
		8088				23 + ea								
		80286				7								
		80386				6								
		80486				3								
		Pentium-Pentium 4				1 o 3								

Formato	Ejemplos	Microprocesador	Ciclos de reloj	
DEC reg16 DEC reg32	DEC CX	8086	3	
	DEC DI	8088	3	
	DEC EDX	80286	2	
	DEC ECX	80386	2	
	DEC BP	80486	1	
		Pentium-Pentium 4	1	
DIV División				
1111011w oo110mmm desp		O D I T	S Z A P C	
Formato	Ejemplos	?	? ? ? ? ?	
DIV reg		Microprocesador	Ciclos de reloj	
		8086	162	
		8088	162	
		80286	22	
		80386	38	
		80486	40	
		Pentium-Pentium 4	17-41	
DIV mem	DIV DATOSY DIV LISTA DIV DATOS6[DI] DIV BYTE PTR [BX] DIV WORD PTR [EBX]	8086	168	
		8088	176	
		80286	25	
		80386	41	
		80486	40	
		Pentium-Pentium 4	17-41	

ENTER Crea una estructura tipo pila			
11001000 datos			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
ENTER imm,0	ENTER 4,0 ENTER 8,0 ENTER 100,0 ENTER 200,0 ENTER 1024,0	8086	–
		8088	–
		80286	11
		80386	10
		80486	14
		Pentium-Pentium 4	11
ENTER imm,1	ENTER 4,1 ENTER 10,1	8086	–
		8088	–
		80286	12
		80386	15
		80486	17
		Pentium-Pentium 4	15
ENTER imm,imm	ENTER 3,6 ENTER 100,3	8086	–
		8088	–
		80286	12
		80386	15
		80486	17
		Pentium-Pentium 4	15 + 2n
ESC Escape (obsoleta; vea el coprocesador)			

HLT		Alto		
11110100				
Ejemplo		Microprocesador	Ciclos de reloj	
HLT		8086	2	
		8088	2	
		80286	2	
		80386	5	
		80486	4	
		Pentium-Pentium 4	Varía	
IDIV		División de enteros (con signo)		
1111011w oo111mmm desp		O D I T	S Z A P C	
		? ? ? ? ?		
Formato		Ejemplos	Microprocesador	
IDIV reg		IDIV BL	8086	
		IDIV BH	184	
		IDIV ECX	8088	
		IDIV DH	184	
		IDIV CX	80286	
			25	
IDIV mem		80386	43	
		80486	43	
		Pentium-Pentium 4	22-46	
		IDIV DATOSY	8086	
		IDIV LISTA	190	
		IDIV DATOS6[DI]	8088	
		IDIV BYTE PTR[BX]	194	
		IDIV WORD PTR[EBX]	80286	
			28	
		80386	46	
		80486	44	
		Pentium-Pentium 4	22-46	

IMUL Multiplicación de enteros (con signo)			
1111011w oo101mmm desp		O D I T	S Z A P C
Formato	Ejemplos	Microprocesador Ciclos de reloj	
IMUL reg	IMUL BL IMUL CX IMUL ECX IMUL DH IMUL AL	8086	154
		8088	154
		80286	21
		80386	38
		80486	42
		Pentium-Pentium 4	10-11
IMUL mem	IMUL DATOSY IMUL LISTA IMUL DATOS6[DI] IMUL BYTE PTR[BX] IMUL WORD PTR[EBX]	8086	160
		8088	164
		80286	24
		80386	41
		80486	42
		Pentium-Pentium 4	10-11
011010s1 oorrrmmm desp datos			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
IMUL reg,imm	IMUL CX,16 IMUL DI,100 IMUL EDX,20	8086	—
		8088	—
		80286	21
		80386	38
		80486	42
		Pentium-Pentium 4	10
IMUL reg,reg,imm	IMUL DX,AX,2 IMUL CX,DX,3 IMUL BX,AX,33	8086	—
		8088	—
		80286	21
		80386	38
		80486	42
		Pentium-Pentium 4	10

IMUL reg,mem,imm	IMUL CX,DATOSY,99	8086	-	
		8088	-	
		80286	24	
		80386	38	
		80486	42	
		Pentium-Pentium 4	10	
00001111 10101111 corrrmmm desp				
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
IMUL reg,reg	IMUL CX,DX IMUL DI,BX IMUL EDX,EBX	8086	-	
		8088	-	
		80286	-	
		80386	38	
		80486	42	
		Pentium-Pentium 4	10	
IMUL reg,mem	IMUL DX,DATOSY IMUL CX,LISTA IMUL ECX,DATOS6[DI]	8086	-	
		8088	-	
		80286	-	
		80386	41	
		80486	42	
		Pentium-Pentium 4	10	
IN Introduce datos desde el puerto				
1110010w #puerto				
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
IN acum,pt	IN AL,12H IN AX,12H IN AL,0FFH IN AX,0A0H IN EAX,10H	8086	10	
		8088	14	
		80286	5	
		80386	12	
		80486	14	
		Pentium-Pentium 4	7	

Formato	Ejemplos	Microprocesador	Ciclos de reloj	
IN acum,DX	IN AL,DX IN AX,DX IN EAX,DX	8086	8	
		8088	12	
		80286	5	
		80386	13	
		80486	14	
		Pentium-Pentium 4	7	
INC Incremento				
1111111w 00000mmm desp		O D I T * * * * *	S Z A P C	
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
INC reg8	INC BL INC BH INC AL INC AH INC DH	8086	3	
		8088	3	
		80286	2	
		80386	2	
		80486	1	
		Pentium-Pentium 4	1 o 3	
INC mem	INC DATOS3 INC LISTA INC CONTEO INC BYTE PTR[DI] INC WORD PTR[ECX]	8086	15 + ea	
		8088	23 + ea	
		80286	7	
		80386	6	
		80486	3	
		Pentium-Pentium 4	1 o 3	
INC reg16 INC reg32	INC CX INC DX INC BP INC ECX INC ESP	8086	3	
		8088	3	
		80286	2	
		80386	2	
		80486	1	
		Pentium-Pentium 4	1	

INS Introduce una cadena de texto desde el puerto			
0110110w Formato	Ejemplos	Microprocesador	Ciclos de reloj
INSB	INSB	8086	–
INSW	INSW	8088	–
INSD	INSD	80286	5
	INS DATOS2	80386	15
	REP INSB	80486	17
		Pentium-Pentium 4	9
INT Interrupción			
11001101 tipo Formato	Ejemplos	Microprocesador	Ciclos de reloj
INT tipo	INT 12H	8086	51
	INT 15H	8088	71
	INT 21H	80286	23
	INT 2FH	80386	37
	INT 10H	80486	30
		Pentium-Pentium 4	16-82
INT 3 Interrupción 3			
11001100 Ejemplo		Microprocesador	Ciclos de reloj
INT 3		8086	52
		8088	72
		80286	23
		80386	33
		80486	26
		Pentium-Pentium 4	13-56

INTO	Interrupción por desbordamiento				
11001110 Ejemplo		Microprocesador	Ciclos de reloj		
INTO		8086	53		
		8088	73		
		80286	24		
		80386	35		
		80486	28		
		Pentium-Pentium 4	13-56		
INVD	Invalida caché de datos				
00001111 00001000 Ejemplo		Microprocesador	Ciclos de reloj		
INTVD		8086	–		
		8088	–		
		80286	–		
		80386	–		
		80486	4		
		Pentium-Pentium 4	15		
IRET/IRET D	Retorno de interrupción				
11001101 datos	O D I T S Z A P C				
	* * * *	* * * *			
Formato	Ejemplos	Microprocesador	Ciclos de reloj		
IRET IRETD	IRET IRETD IRET 100	8086	32		
		8088	44		
		80286	17		
		80386	22		
		80486	15		
		Pentium-Pentium 4	8-27		

Jcondición Salto condicional					
0111cccc desp		Ejemplos	Microprocesador	Ciclos de reloj	
Formato	Jcnd etiqueta (desp 8 bits)				
Jcnd etiqueta (desp 8 bits)	JA ARRIBA JB DEBAJO JG MAYOR JE IGUAL JZ CERO		8086	16/4	
			8088	16/4	
			80286	7/3	
			80386	7/3	
			80486	3/1	
			Pentium-Pentium 4	1	
00001111 1000cccc desp					
Formato	Jcnd etiqueta (desp 16 bits)	Ejemplos	Microprocesador	Ciclos de reloj	
Jcnd etiqueta (desp 16 bits)		JNE NO_IGUAL JLE MENOR_QUE	8086	—	
			8088	—	
			80286	—	
			80386	7/3	
			80486	3/1	
			Pentium-Pentium 4	1	
Condición					
Códigos	Nemónico	Bandera	Descripción		
0000	JO	O = 1	Salta si hay desbordamiento		
0001	JNO	O = 0	Salta si no hay desbordamiento		
0010	JB/NAE	C = 1	Salta si está por debajo		
0011	JAE/JNB	C = 0	Salta si está por encima o si es igual		
0100	JE/JZ	Z = 1	Salta si es igual/cero		
0101	JNE/JNZ	Z = 0	Salta si no es igual/cero		
0110	JBE/JNA	C = 1 + Z = 1	Salta si está por debajo o si es igual		
0111	JA/JNBE	C = 0 • Z = 0	Salta si está por encima		
1000	JS	S = 1	Salta si hay signo		
1001	JNS	S = 0	Salta si no hay signo		
1010	JP/JPE	P = 1	Salta si hay paridad		
1011	JNP/JPO	P = 0	Salta si no hay paridad		
1100	JL/JNGE	S • O	Salta si es menor que		
1101	JGE/JNL	S = 0	Salta si es mayor o igual que		
1110	JLE/JNG	Z = 1 + S • O	Salta si es menor o igual que		
1111	JG/JNLE	Z = 0 + S = O	Salta si es mayor que		

JCXZ/JECXZ Salta si CX (ECX) es igual a cero			
11100011 Formato	Ejemplos	Microprocesador	Ciclos de reloj
JCXZ etiqueta JECXZ etiqueta	JCXZ ENCIMA	8086	18/6
	JCXZ DEBAJO	8088	18/6
	JECXZ MAYOR	80286	8/4
	JECXZ IGUAL	80386	9/5
	JCXZ SIGUIENTE	80486	8/5
		Pentium-Pentium 4	6/5
JMP Salto			
11101011 desp Formato	Ejemplos	Microprocesador	Ciclos de reloj
JMP etiqueta (corto)	JMP CORTO ARRIBA	8086	15
	JMP CORTO ABAJO	8088	15
	JMP CORTO ENCIMA	80286	7
	JMP CORTO CIRCUITO	80386	7
	JMP CORTO LISTON	80486	3
		Pentium-Pentium 4	1
11101001 desp Formato	Ejemplos	Microprocesador	Ciclos de reloj
JMP etiqueta (cercano)	JMP VERS	8086	15
	JMP RANA	8088	15
	JMP DEBAJO	80286	7
	JMP NEAR PTR SOBRE	80386	7
		80486	3
		Pentium-Pentium 4	1

11101010 desp Formato		Ejemplos	Microprocesador	Ciclos de reloj
JMP etiqueta (lejano)		JMP NO_MAS	8086	15
		JMP DEBAJO	8088	15
		JMP DE_NUEVO	80286	11
		JMP FAR PTR AH1	80386	12
			80486	17
			Pentium-Pentium 4	3
11111111 oo100mmm Formato		Ejemplos	Microprocesador	Ciclos de reloj
JMP reg (cercano)		JMP AX	8086	11
		JMP EAX	8088	11
		JMP CX	80286	7
		JMP DX	80386	7
			80486	3
			Pentium-Pentium 4	2
JMP mem (cercano)		JMP VERS	8086	18 + ea
		JMP RANA	8088	18 + ea
		JMP CD:DEBAJO	80286	11
		JMP DATOS1[DI+2]	80386	10
			80486	5
			Pentium-Pentium 4	4
11111111 oo101mmm Formato		Ejemplos	Microprocesador	Ciclos de reloj
JMP mem (lejano)		JMP MUY_DESVIADO	8086	24 + ea
		JMP TABLA	8088	24 + ea
		JMP ARRIBA	80286	15
		JMP FUERA_DE_AQUI	80386	12
			80486	13
			Pentium-Pentium 4	4

LAHF Carga AH de las banderas				
10011111 Ejemplo		Microprocesador	Ciclos de reloj	
LAHF		8086	4	
		8088	4	
		80286	2	
		80386	2	
		80486	3	
		Pentium-Pentium 4	2	
LAR Carga byte de derechos de acceso				
00001111 000000010 oorrrmmm desp		O D I T	S Z A P C	
Formato		Ejemplos	Microprocesador Ciclos de reloj	
LAR reg,reg		LAR AX,BX LAR CX,DX LAR ECX,EDX	8086 8088 80286 80386 80486 Pentium-Pentium 4	
			— — 14 15 11 8	
LAR reg,mem		LAR CX,DATOS1 LAR AX,LISTA3 LAR ECX,RANITA	8086 8088 80286 80386 80486 Pentium-Pentium 4	
			— — 16 16 11 8	

LDS Carga apuntador lejano a DS y registro				
11000101 oorrrmmm Formato		Ejemplos	Microprocesador	Ciclos de reloj
LDS reg,mem		LDS DI,DATOS3	8086	16 + ea
		LDS SI,LISTA2	8088	24 + ea
		LDS BX,ARREGLO_PTR	80286	7
		LDS CX,APNTDR	80386	7
			80486	6
			Pentium-Pentium 4	4
LEA Carga dirección efectiva				
10001101 oorrrmmm desp Formato		Ejemplos	Microprocesador	Ciclos de reloj
LEA reg,mem		LEA DI,DATOS3	8086	2 + ea
		LEA SI,LISTA2	8088	2 + ea
		LEA BX,ARREGLO_PTR	80286	3
		LEA CX,APNTDR	80386	2
			80486	2
			Pentium-Pentium 4	1
LEAVE Deja procedimiento de alto nivel				
11001001 Ejemplo		Microprocesador	Ciclos de reloj	
LEAVE		8086	—	
		8088	—	
		80286	5	
		80386	4	
		80486	5	
		Pentium-Pentium 4	3	

LES Carga apuntador lejano a ES y registro				
11000100 oorrrmmm Formato		Ejemplos	Microprocesador	Ciclos de reloj
LES reg,mem		LES DI,DATOS3	8086	16 + ea
		LES SI,LISTA2	8088	24 + ea
		LES BX,ARREGLO_PTR	80286	7
		LES CX,APNTDR	80386	7
			80486	6
			Pentium-Pentium 4	4
LFS Carga apuntador lejano a FS y registro				
00001111 10110100 oorrrmmm desp Formato		Ejemplos	Microprocesador	Ciclos de reloj
LFS reg,mem		LFS DI,DATOS3	8086	–
		LFS SI,LISTA2	8088	–
		LFS BX,ARREGLO_PTR	80286	–
			80386	7
			80486	6
			Pentium-Pentium 4	4
LGDT Carga la tabla de descriptores globales				
00001111 00000001 oo010mmm desp Formato		Ejemplos	Microprocesador	Ciclos de reloj
LGDT mem64		LGDT DESCRIP	8086	–
		LGDT TABLAD	8088	–
			80286	11
			80386	11
			80486	11
			Pentium-Pentium 4	6

LGS Carga apuntador lejano a GS y registro			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
LGS reg,mem	LGS DI,DATOS3	8086	–
	LGS SI,LISTA2	8088	–
	LGS BX,ARREGLO_PTR	80286	–
	LGS CX,APNTDR	80386	7
		80486	6
		Pentium-Pentium 4	4
LIDT Carga la tabla de descriptores de interrupción			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
LIDT mem64	LIDT DATOS3	8086	–
	LIDT LISTA2	8088	–
		80286	12
		80386	11
		80486	11
		Pentium-Pentium 4	6
LLDT Carga la tabla de descriptores locales			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
LLDT reg	LLDT BX	8086	–
	LLDT DX	8088	–
	LLDT CX	80286	17
		80386	20
		80486	11
		Pentium-Pentium 4	9

LLDT mem	LLDT DATOS1 LLDT LISTA3 LLDT RANITA	8086	—	
		8088	—	
		80286	19	
		80386	24	
		80486	11	
		Pentium-Pentium 4	9	
LMSW Carga la palabra de estado del equipo (sólo 80286)				
00001111 0000001 oo110mmm desp				
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
LMSW reg	LMSW BX LMSW DX LMSW CX	8086	—	
		8088	—	
		80286	3	
		80386	10	
		80486	2	
		Pentium-Pentium 4	8	
LMSW mem	LMSW DATOS1 LMSW LISTA3 LMSW RANITA	8086	—	
		8088	—	
		80286	6	
		80386	13	
		80486	3	
		Pentium-Pentium 4	8	

LOCK Bloquea el bus			
11110000 Formato	Ejemplos	Microprocesador	Ciclos de reloj
LOCK:inst	LOCK:XCHG AX,BX LOCK:ADD AL,3	8086	2
		8088	3
		80286	0
		80386	0
		80486	1
		Pentium-Pentium 4	1
LODS Carga el operando tipo cadena de caracteres			
1010110w Formato	Ejemplos	Microprocesador	Ciclos de reloj
LODSB LODSW LODSD	LODSB LODSW LODSD LODSD DATOS3	8086	12
		8088	15
		80286	5
		80386	5
		80486	5
		Pentium-Pentium 4	2
LOOP/LOOPD Itera hasta que CX = 0 o ECX = 0			
11100010 desp Formato	Ejemplos	Microprocesador	Ciclos de reloj
LOOP etiqueta LOOPD etiqueta	LOOP SIGUIENTE LOOP ATRAS LOOPD CICLOS	8086	17/5
		8088	17/5
		80286	8/4
		80386	11
		80486	7/6
		Pentium-Pentium 4	5/6

LOOPE/LOOPED Itera mientras sea igual				
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
11100001 desp LOOPE etiqueta LOOPED etiqueta LOOPZ etiqueta LOOPZD etiqueta	LOOPE DE_NUEVO LOOPED HASTA LOOPZ ZORRO LOOPZD WOW	8086	18/6	
		8088	18/6	
		80286	8/4	
		80386	11	
		80486	9/6	
		Pentium-Pentium 4	7/8	
LOOPNE/LOOPNED Itera mientras no sea igual				
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
11100000 desp LOOPNE etiqueta LOOPNED etiqueta LOOPNZ etiqueta LOOPNZD etiqueta	LOOPNE HACIA_DELANTE LOOPNED UPS LOOPNZ PRUEBA_DE_NUEVO LOOPZD WOO	8086	19/5	
		8088	19/5	
		80286	8/4	
		80386	11	
		80486	9/6	
		Pentium-Pentium 4	7/8	
LSL Carga el límite del segmento				
00001111 00000011 oorrrmmm desp		O D I T	S Z A P C	
			*	
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
LSL reg,reg	LSL AX,BX LSL CX,BX LSL EDX,EAX	8086	—	
		8088	—	
		80286	14	
		80386	25	
		80486	10	
		Pentium-Pentium 4	8	

LSL reg,mem	LSL AX,LIMITE LSL EAX,NUM	8086	—	
		8088	—	
		80286	16	
		80386	26	
		80486	10	
		Pentium-Pentium 4	8	
LSS Carga apuntador lejano a SS y registro				
00001111 10110010 oorrrmmm desp				
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
LSS reg,mem	LSS DI,DATOS1 LSS SP,PARTESUP_PILA LSS CX,ARREGLO	8086	—	
		8088	—	
		80286	—	
		80386	7	
		80486	6	
		Pentium-Pentium 4	4	
LTR Carga registro de tareas				
00001111 00000000 oo001mmm desp				
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
LTR reg	LTR AX LTR CX LTR DX	8086	—	
		8088	—	
		80286	17	
		80386	23	
		80486	20	
		Pentium-Pentium 4	10	

LTR mem16	LTR TAREA LTR NUM	8086	-	
		8088	-	
		80286	19	
		80386	27	
		80486	20	
		Pentium-Pentium 4	10	
MOV Mueve datos				
100010dw oorrrmmm desp				
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
MOV reg,reg	MOV CL,CH MOV BH,CL MOV CX,DX MOV EAX,EBP MOV ESP,ESI	8086	2	
		8088	2	
		80286	2	
		80386	2	
		80486	1	
		Pentium-Pentium 4	1	
MOV mem,reg	MOV DATOS7,DL MOV NUM,CX MOV TEMP,EBX MOV [ECX],BL MOV [DI],DH	8086	9 + ea	
		8088	13 + ea	
		80286	3	
		80386	2	
		80486	1	
		Pentium-Pentium 4	1	
MOV reg,mem	MOV DL,DATOS8 MOV DX,NUM MOV EBX,TEMP+3 MOV CH,TEMP[EDI] MOV CL,DATOS2	8086	10 + ea	
		8088	12 + ea	
		80286	5	
		80386	4	
		80486	1	
		Pentium-Pentium 4	1	

1100011w oo000mmmm datos desp			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
MOV mem,imm	MOV DATOSF,23H	8086	10 + ea
	MOV LISTA,12H	8088	14 + ea
	MOV BYTE PTR[DI],2	80286	3
	MOV NUM,234H	80386	2
	MOV DWORD PTR[ECX],1	80486	1
		Pentium-Pentium 4	1
1011wrrr datos			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
MOV reg,imm	MOV BX,22H	8086	4
	MOV CX,12H	8088	4
	MOV CL,2	80286	3
	MOV ECX,123456H	80386	2
	MOV DI,100	80486	1
		Pentium-Pentium 4	1
101000dw desp			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
MOV mem,acu	MOV DATOSF,AL	8086	10
	MOV LISTA,AX	8088	14
	MOV NUM,EAX	80286	3
		80386	2
		80486	1
		Pentium-Pentium 4	1
MOV acu,mem	MOV AL,DATOSE	8086	10
	MOV AX,LISTA	8088	14
	MOV EAX,LUTE	80286	5
		80386	4
		80486	1
		Pentium-Pentium 4	1

100011d0 oosssmmm desp		Formato	Ejemplos	Microprocesador	Ciclos de reloj
MOV seg,reg			MOV SS,AX	8086	2
			MOV DS,DX	8088	2
			MOV ES,CX	80286	2
			MOV FS,BX	80386	2
			MOV GS,AX	80486	1
				Pentium-Pentium 4	1
MOV seg,mem			MOV SS,PARTESUP_PILA	8086	8 + ea
			MOV DS,DATOSS	8088	12 + ea
			MOV ES,TEMP1	80286	2
				80386	2
				80486	1
				Pentium-Pentium 4	2 o 3
MOV reg,seg			MOV BX,DS	8086	2
			MOV CX,FS	8088	2
			MOV CX,ES	80286	2
				80386	2
				80486	1
				Pentium-Pentium 4	1
MOV mem,seg			MOV DATOS2,CS	8086	9 + ea
			MOV TEMP,DS	8088	13 + ea
			MOV NUM1,SS	80286	3
			MOV TEMP2,GS	80386	2
				80486	1
				Pentium-Pentium 4	1

00001111 001000d0 11rrrmmm			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
MOV reg,cr	MOV EBX,CR0 MOV ECX,CR2 MOV EBX,CR3	8086	—
		8088	—
		80286	—
		80386	6
		80486	4
		Pentium-Pentium 4	4
MOV cr,reg	MOV CR0,EAX MOV CR1,EBX MOV CR3,EDX	8086	—
		8088	—
		80286	—
		80386	10
		80486	4
		Pentium-Pentium 4	12-46
00001111 001000d1 11rrrmmm			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
MOV reg,dr	MOV EBX,DR6 MOV ECX,DR7 MOV EBX,DR1	8086	—
		8088	—
		80286	—
		80386	22
		80486	10
		Pentium-Pentium 4	11
MOV dr,reg	MOV DR0,EAX MOV DR1,EBX MOV DR3,EDX	8086	—
		8088	—
		80286	—
		80386	22
		80486	11
		Pentium-Pentium 4	11

00001111 001001d0 11rrrrmm		Ejemplos	Microprocesador	Ciclos de reloj	
Formato					
MOV reg,tr		MOV EBX,TR6 MOV ECX,TR7	8086	—	
			8088	—	
			80286	—	
			80386	12	
			80486	4	
			Pentium-Pentium 4	11	
MOV tr,reg		MOV TR6,EAX MOV TR7,EBX	8086	—	
			8088	—	
			80286	—	
			80386	12	
			80486	6	
			Pentium-Pentium 4	11	
MOVS		Mueve datos tipo cadena de texto			
1010010w		Ejemplos	Microprocesador	Ciclos de reloj	
Formato					
MOVSB MOVSW MOVSD		MOVSB MOVSW MOVSD MOVS DATOS1,DATOS2	8086	18	
			8088	26	
			80286	5	
			80386	7	
			80486	7	
			Pentium-Pentium 4	4	

MOVSX Movimiento con extensión de signo			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
MOVSX reg,reg	MOVSX BX,AL MOVSX EAX,DX	8086	–
		8088	–
		80286	–
		80386	3
		80486	3
		Pentium-Pentium 4	3
MOVSX reg,mem	MOVSX AX,DATOS34 MOVSX EAX,NUMB	8086	–
		8088	–
		80286	–
		80386	6
		80486	3
		Pentium-Pentium 4	3
MOVZX Movimiento con extensión de cero			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
MOVZX reg,reg	MOVZX BX,AL MOVZX EAX,DX	8086	–
		8088	–
		80286	–
		80386	3
		80486	3
		Pentium-Pentium 4	3
MOVZX reg,mem	MOVZX AX,DATOS34 MOVZX EAX,NUMB	8086	–
		8088	–
		80286	–
		80386	6
		80486	3
		Pentium-Pentium 4	3

MUL Multiplicación					
1111011w oo100mmm desp		O	D	I	T
Formato	Ejemplos	*	?	?	?
MUL reg	MUL BL MUL CX MUL EDX	8086			118
		8088			143
		80286			21
		80386			38
		80486			42
		Pentium-Pentium 4		10 o 11	
MUL mem	MUL DATOS9 MUL WORD PTR[ESI]	8086			139
		8088			143
		80286			24
		80386			41
		80486			42
		Pentium-Pentium 4		11	
NEG Negación					
1111011w oo011mmm desp		O	D	I	T
Formato	Ejemplos	*	?	?	?
NEG reg	NEG BL NEG CX NEG EDI	8086			3
		8088			3
		80286			2
		80386			2
		80486			1
		Pentium-Pentium 4		1 o 3	

NEG mem	NEG DATOS9 NEG WORD PTR[ESI]	8086	16 + ea	
		8088	24 + ea	
		80286	7	
		80386	6	
		80486	3	
		Pentium-Pentium 4	1 o 3	
NOP Ninguna operación				
10010000 Ejemplo		Microprocesador	Ciclos de reloj	
NOP		8086	3	
		8088	3	
		80286	3	
		80386	3	
		80486	3	
		Pentium-Pentium 4	1	
NOT Complemento a uno				
1111011w oo0010mmm desp				
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
NOT reg	NOT BL NOT CX NOT EDI	8086	3	
		8088	3	
		80286	2	
		80386	2	
		80486	1	
		Pentium-Pentium 4	1 o 3	
NOT mem	NOT DATOS9 NOT WORD PTR[ESI]	8086	16 + ea	
		8088	24 + ea	
		80286	7	
		80386	6	
		80486	3	
		Pentium-Pentium 4	1 o 3	

OR	OR inclusivo		
000010dw oorrrmmmm desp		O D I T	S Z A P C
Formato	Ejemplos	Microprocesador	Ciclos de reloj
OR reg,reg	OR AX,BX	8086	3
	OR AL,BL	8088	3
	OR EAX,EBX	80286	2
	OR CX,SI	80386	2
	OR ESI,EDI	80486	1
		Pentium-Pentium 4	1 o 2
OR mem,reg	OR DATOSY,AL	8086	16 + ea
	OR LISTA,SI	8088	24 + ea
	OR DATOS2[DI],CL	80286	7
	OR [EAX],BL	80386	7
	OR [EBX+2*ECX],EDX	80486	3
		Pentium-Pentium 4	1 o 3
OR reg,mem	OR BL,DATOS1	8086	9 + ea
	OR SI,LISTA1	8088	13 + ea
	OR CL,DATOS2[SI]	80286	7
	OR CX[ESI]	80386	6
	OR ESI,[2*ECX]	80486	2
		Pentium-Pentium 4	1 o 3
100000sw oo001mmm desp datos			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
OR reg,imm	OR CX,3	8086	4
	OR DI,1AH	8088	4
	OR DL,34H	80286	3
	OR EDX,1345H	80386	2
	OR CX,1834H	80486	1
		Pentium-Pentium 4	1 o 3

OR mem,imm	OR DATOS,3	8086	17 + ea
	OR BYTE PTR[EDI],1AH	8088	25 + ea
	OR PAPI,34H	80286	7
	OR LISTA,'A'	80386	7
	OR RANITA,1834H	80486	3
		Pentium-Pentium 4	1 o 3
	0000110w datos		
Formato	Ejemplos	Microprocesador	Ciclos de reloj
OR acu,imm	OR AX,3	8086	4
	OR AL,1AH	8088	4
	OR AH,34H	80286	3
	OR EAX,1345H	80386	2
	OR AL,'Y'	80486	1
		Pentium-Pentium 4	1
	OUT Envía datos al puerto		
1110011w #puerto			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
OUT pt,acu	OUT 12H,AL	8086	10
	OUT 12H,AX	8088	14
	OUT 0FFH,AL	80286	3
	OUT 0A0H,AX	80386	10
	OUT 10H,EAX	80486	10
		Pentium-Pentium 4	12-26
1110111w			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
OUT DX,acu	OUT DX,AL	8086	8
	OUT DX,AX	8088	12
	OUT DX,EAX	80286	3
		80386	11
		80486	10
		Pentium-Pentium 4	12-26

OUTS	Envía cadena de texto al puerto		
0110111w Formato	Ejemplos	Microprocesador	Ciclos de reloj
OUTSB	OUTSB	8086	—
OUTSW	OUTSW	8088	—
OUTSD	OUTSD	80286	5
	OUTS DATOS2	80386	14
	REP OUTSB	80486	10
		Pentium-Pentium 4	13-27
POP	Saca datos de la pila		
01011rrr Formato	Ejemplos	Microprocesador	Ciclos de reloj
POP reg	POP CX POP AX POP EDI	8086	8
		8088	12
		80286	5
		80386	4
		80486	1
		Pentium-Pentium 4	1
10001111 oo0000mmm desp Formato	Ejemplos	Microprocesador	Ciclos de reloj
POP mem	POP DATOS1 POP LISTAS POP NUMEROS	8086	17 + ea
		8088	25 + ea
		80286	5
		80386	5
		80486	4
		Pentium-Pentium 4	3

00sss111 Formato		Ejemplos	Microprocesador	Ciclos de reloj	
POP seg		POP DS	8086	8	
		POP ES	8088	12	
		POP SS	80286	5	
			80386	7	
			80486	3	
			Pentium-Pentium 4	3	
00001111 10sss001 Formato		Ejemplos	Microprocesador	Ciclos de reloj	
POP seg		POP FS	8086	—	
		POP GS	8088	—	
			80286	—	
			80386	7	
			80486	3	
			Pentium-Pentium 4	3	
POPA/POPAD		Saca todos los registros de la pila			
01100001 Ejemplo		Microprocesador			
POPA POPAD			8086	—	
			8088	—	
			80286	19	
			80386	24	
			80486	9	
			Pentium-Pentium 4	5	

POPF/POPFD		Saca las banderas de la pila	
10010000		O D I T S Z A P C * * * * * * * * *	
Ejemplo		Microprocesador	Ciclos de reloj
POPF	8086	8	
POPFD	8088	12	
	80286	5	
	80386	5	
	80486	6	
	Pentium-Pentium 4	4 o 6	
PUSH	Mete datos en la pila		
01010rrr			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
PUSH reg	PUSH CX PUSH AX PUSH EDI	8086 8088 80286 80386 80486 Pentium-Pentium 4	11 15 3 2 1 1
11111111 oo110mmm desp			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
PUSH mem	PUSH DATOS1 PUSH LISTAS PUSH NUMEROS	8086 8088 80286 80386 80486 Pentium-Pentium 4	16 + ea 24 + ea 5 5 4 1 o 2

00ss110 Formato		Ejemplos	Microprocesador	Ciclos de reloj
PUSH seg		PUSH ES	8086	10
		PUSH CX	8088	14
		PUSH DS	80286	3
			80386	2
			80486	3
			Pentium-Pentium 4	1
00001111 10sss000 Formato		Ejemplos	Microprocesador	Ciclos de reloj
PUSH seg		PUSH FS	8086	—
		PUSH GS	8088	—
			80286	—
			80386	2
			80486	3
			Pentium-Pentium 4	1
011010s0 datos Formato		Ejemplos	Microprocesador	Ciclos de reloj
PUSH imm		PUSH 2000H	8086	—
		PUSH 53220	8088	—
		PUSHW 10H	80286	3
		PUSH';	80386	2
		PUSHD 100000H	80486	1
			Pentium-Pentium 4	1

PUSHA/PUSHAD Mete todos los registros en la pila					
01100000 Ejemplo		Microprocesador	Ciclos de reloj		
PUSHA	8086	—			
PUSHAD	8088	—			
	80286	17			
	80386	18			
	80486	11			
	Pentium-Pentium 4	5			
PUSHF/PUSHFD Mete las banderas en la pila					
10011100 Ejemplo		Microprocesador	Ciclos de reloj		
PUSHF	8086	10			
PUSHFD	8088	14			
	80286	3			
	80386	4			
	80486	3			
	Pentium-Pentium 4	3 o 4			
RCL/RCR/ROL/ROR Desplazamiento cíclico					
1101000w ooTTTmmm desp	O D I T	S Z A P C			
	*	*			
TTT = 000 = ROL, TTT = 001 = ROR, TTT = 010 = RCL y TTT = 011 = RCR					
Formato	Ejemplos	Microprocesador	Ciclos de reloj		
ROL reg,1	ROL CL,1	8086	2		
ROR reg,1	ROL DX,1	8088	2		
	ROR CH,1	80286	2		
	ROR SI,1	80386	3		
		80486	3		
		Pentium-Pentium 4	1 o 3		

RCL reg,1 RCR reg,1	RCL CL,1	8086	2
	RCL SI,1	8088	2
	RCR AH,1	80286	2
	RCR EBX,1	80386	9
		80486	3
		Pentium-Pentium 4	1 o 3
ROL mem,1 ROR mem,1	ROL DATOSY,1	8086	15 + ea
	ROL LISTA,1	8088	23 + ea
	ROR DATOS2[DI],1	80286	7
	ROR BYTE PTR[EAX],1	80386	7
		80486	4
		Pentium-Pentium 4	1 o 3
RCL mem,1 RCR mem,1	RCL DATOS1,1	8086	15 + ea
	RCL LISTA,1	8088	23 + ea
	RCR DATOS2[SI],1	80286	7
	RCR WORD PTR[ESI],1	80386	10
		80486	4
		Pentium-Pentium 4	1 o 3
1101001w ooTTTmmmm desp			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
ROL reg,CL ROR reg,CL	ROL CH,CL	8086	8 + 4n
	ROL DX,CL	8088	8 + 4n
	ROR AL,CL	80286	5 + n
	ROR ESI,CL	80386	3
		80486	3
		Pentium-Pentium 4	4

RCL reg,CL RCR reg,CL	RCL CH,CL RCL SI,CL RCR AH,CL RCR RBX,CL	8086	8 + 4n
		8088	8 + 4n
		80286	5 + n
		80386	9
		80486	3
		Pentium-Pentium 4	7-27
ROL mem,CL ROR mem,CL	ROL DATOSY,CL ROL LISTA,CL ROR DATOS2[DI],CL ROR BYTE PTR[EAX],CL	8086	20 + 4n
		8088	28 + 4n
		80286	8 + n
		80386	7
		80486	4
		Pentium-Pentium 4	4
RCL mem,CL RCR mem,CL	RCL DATOS1,CL RCL LISTA,CL RCR DATOS2[SI],CL RCR WORD PTR[ESI],CL	8086	20 + 4n
		8088	28 + 4n
		80286	8 + n
		80386	10
		80486	9
		Pentium-Pentium 4	9-26
1100000w ooTTTmmm desp datos			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
ROL reg,imm ROR reg,imm	ROL CH,4 ROL DX,5 ROR AL,2 ROR ESI,14	8086	—
		8088	—
		80286	5 + n
		80386	3
		80486	2
		Pentium-Pentium 4	1 o 3

RCL reg,imm RCR reg,imm	RCL CL,2	8086	–
	RCL SI,12	8088	–
	RCR AH,5	80286	5 + n
	RCR EBX,18	80386	9
		80486	8
		Pentium-Pentium 4	8-27
ROL mem,imm ROR mem,imm	ROL DATOSY,4	8086	–
	ROL LISTA,3	8088	–
	ROR DATOS2[DI],7	80286	8 + n
	ROR BYTE PTR[EAX],11	80386	7
		80486	4
		Pentium-Pentium 4	1 o 3
RCL mem,imm RCR mem,imm	RCL DATOS1,5	8086	–
	RCL LISTA,3	8088	–
	RCR DATOS2[SI],9	80286	8 + n
	RCR WORD PTR[ESI],8	80386	10
		80486	9
		Pentium-Pentium 4	8-27
RDMSR Lee registro específico del modelo			
00001111 00110010 Ejemplo		Microprocesador	Ciclos de reloj
RDMSR		8086	–
		8088	–
		80286	–
		80386	–
		80486	–
		Pentium-Pentium 4	20-24

REP Prefijo de repetición			
11110011 1010010w			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
REP MOVS	REP MOVSB REP MOVSW REP MOVSD REP MOVS DATOS1,DATOS2	8086	9 + 17n
		8088	9 + 25n
		80286	5 + 4n
		80386	8 + 4n
		80486	12 + 3n
		Pentium-Pentium 4	13 + n
11110011 1010101w			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
REP STOS	REP STOSB REP STOSW REP STOSD REP STOS ARREGLO	8086	9 + 10n
		8088	9 + 14n
		80286	4 + 3n
		80386	5 + 5n
		80486	7 + 4n
		Pentium-Pentium 4	9 + n
11110011 0110110w			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
REP INS	REP INSS REP INSW REP INSD REP INS ARREGLO	8086	–
		8088	–
		80286	5 + 4n
		80386	12 + 5n
		80486	17 + 5n
		Pentium-Pentium 4	25 + 3n

11110011 0110111w			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
REP OUTS	REP OUTSB	8086	—
	REP OUTSW	8088	—
	REP OUTSD	80286	5 + 4n
	REP OUTS ARREGLO	80386	12 + 5n
		80486	17 + 5n
		Pentium-Pentium 4	25 + 4n
REPE/REPNE		Repetición condicional	
11110011 1010011w			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
REPE CMPS	REPE CMPSB	8086	9 + 22n
	REPE CMPSW	8088	9 + 30n
	REPE CMPSD	80286	5 + 9n
	REPE CMPS DATOS1,DATOS2	80386	5 + 9n
		80486	7 + 7n
		Pentium-Pentium 4	9 + 4n
11110011 1010111w			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
REPE SCAS	REPE SCASB	8086	9 + 15n
	REPE SCASW	8088	9 + 19n
	REPE SCASD	80286	5 + 8n
	REPE SCAS ARREGLO	80386	5 + 8n
		80486	7 + 5n
		Pentium-Pentium 4	9 + 4n

11110010 1010011w		Formato	Ejemplos	Microprocesador	Ciclos de reloj
REPNE CMPS			REPNE CMPSB	8086	9 + 22n
			REPNE CMPSW	8088	9 + 30n
			REPNE CMPSD	80286	5 + 9n
			REPNE CMPS ARREGLO,LISTA	80386	5 + 9n
				80486	7 + 7n
				Pentium-Pentium 4	8 + 4n
11110010 101011w		Formato	Ejemplos	Microprocesador	Ciclos de reloj
REPNE SCAS			REPNE SCASB	8086	9 + 15n
			REPNE SCASW	8088	9 + 19n
			REPNE SCASD	80286	5 + 8n
			REPNE SCAS ARREGLO	80386	5 + 8n
				80486	7 + 5n
				Pentium-Pentium 4	9 + 4n
RET	Retorno de un procedimiento				
11000011	Ejemplo			Microprocesador	Ciclos de reloj
RET (cercano)				8086	16
				8088	20
				80286	11
				80386	10
				80486	5
				Pentium-Pentium 4	2

11000010 datos			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
RET imm (cercano)	RET 4 RET 100H	8086	20
		8088	24
		80286	11
		80386	10
		80486	5
		Pentium-Pentium 4	3
11001011			
Ejemplo		Microprocesador	Ciclos de reloj
RET (lejano)		8086	26
		8088	34
		80286	15
		80386	18
		80486	13
		Pentium-Pentium 4	4-23
11001010 datos			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
RET imm (lejano)	RET 4 RET 100H	8086	25
		8088	33
		80286	11
		80386	10
		80486	5
		Pentium-Pentium 4	4-23

RSM	Reanuda desde el modo de administración del sistema								
00001111 10101010	O	D	I	T	S	Z	A	P	C
Ejemplo	*	*	*	*	*	*	*	*	*
RSM			Microporcesador	Ciclos de reloj					
RSM	8086		—						
	8088		—						
	80286		—						
	80386		—						
	80486		—						
	Pentium-Pentium 4		83						
SAHF	Almacena AH en las banderas								
10011110	O	D	I	T	S	Z	A	P	C
Ejemplo	*	*	*	*	*	*	*	*	*
SAHF			Microporcesador	Ciclos de reloj					
SAHF	8086		4						
	8088		4						
	80286		2						
	80386		3						
	80486		2						
	Pentium-Pentium 4		2						
SAL/SAR/SHL/SHR	Desplazamiento								
1101000w ooTTTmmm desp	O	D	I	T	S	Z	A	P	C
TTT = 100 = SHL/SAL, TTT = 101 = SHR y TTT = 111 = SAR	*	*	*	?	*	*	*	*	*
Formato	Ejemplos		Microporcesador	Ciclos de reloj					
SAL reg,1	SAL CL,1	8086	2						
SHL reg,1	SHL DX,1	8088	2						
SHR reg,1	SAR CH,1	80286	2						
SAR reg,1	SHR SI,1	80386	3						
		80486	3						
		Pentium-Pentium 4	1 o 3						

SAL mem,1 SHL mem,1 SHR mem,1 SAR mem,1	SAL DATOS1,1 SHL BYTE PTR[DI],1 SAR NUME,1 SHR WORD PTR[EDI],1	8086 8088 80286 80386 80486	15 + ea 23 + ea 7 7 4
1101001w ooTTTmmm desp			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
SAL reg,CL SHL reg,CL SAR reg,CL SHR reg,CL	SAL CH,CL SHL DX,CL SAR AL,CL SHR ESI,CL	8086 8088 80286 80386 80486	8 + 4n 8 + 4n 5 + n 3 3
		Pentium-Pentium 4	4
SAL mem,CL SHL mem,CL SAR mem,CL SHR mem,CL	SAL DATOSU,CL SHL BYTE PTR[ESI],CL SAR NUME,CL SHR TEMP,CL	8086 8088 80286 80386 80486	20 + 4n 28 + 4n 8 + n 7 4
		Pentium-Pentium 4	4
1100000w ooTTTmmm desp datos			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
SAL reg,imm SHL reg,imm SAR reg,imm SHR reg,imm	SAL CH,4 SHL DX,10 SAR AL,2 SHR ESI,23	8086 8088 80286 80386 80486	— — 5 + n 3 2
		Pentium-Pentium 4	1 o 3

SAL mem,imm SHL mem,imm SAR mem,imm SHR mem,imm	SAL DATOSU,3 SHL BYTE PTR[ESI],15 SAR NUME,3 SHR TEMP,5	8086 8088 80286 80386 80486 Pentium-Pentium 4	– – 8 + n 7 4 1 o 3
SBB Resta con acarreo			
000110dw oorrrmmm desp		O D I T	S Z A P C
		*	* * * *
Formato	Ejemplos	Microprocesador	Ciclos de reloj
SBB reg,reg	SBB CL,DL SBB AX,DX SBB CH,CL SBB EAX,EBX SBB ESI,EDI	8086 8088 80286 80386 80486 Pentium-Pentium 4	3 3 2 2 1 1 o 2
SBB mem,reg	SBB DATOSJ,CL SBB BYTES,CX SBB NUMEROS,ECX SWBB [EAX],CX	8086 8088 80286 80386 80486 Pentium-Pentium 4	16 + ea 24 + ea 7 6 3 1 o 3
SBB reg,mem	SBB CL,DATOSJ SBB CX,BYTES SBB ECX,NUMEROS SWBB DX,[EBX+EDI]	8086 8088 80286 80386 80486 Pentium-Pentium 4	9 + ea 13 + ea 7 7 2 1 o 2

100000sw oo011mmm desp datos			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
SBB reg,imm	SBB CX,3	8086	4
	SBB DI,1AH	8088	4
	SBB DL,34H	80286	3
	SBB EDX,1345H	80386	2
	SBB CX,1834H	80486	1
		Pentium-Pentium 4	1 o 3
SBB mem,imm	SBB DATOS,3	8086	17 + ea
	SBB BYTE PTR[EDI],1AH	8088	25 + ea
	SBB PAPI,34H	80286	7
	SBB LISTA,'A'	80386	7
	SBB RANITA,1834H	80486	3
		Pentium-Pentium 4	1 o 3
0001110w datos			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
SBB acu,imm	SBB AX,3	8086	4
	SBB AL,1AH	8088	4
	SBB AH,34H	80286	3
	SBB EAX,1345H	80386	2
	SBB AL,'Y'	80486	1
		Pentium-Pentium 4	1
SCAS	Exploración de cadena de texto		
1010111w			
		O D I T	S Z A P C
		*	* * * * *
Formato	Ejemplos	Microprocesador	Ciclos de reloj
SCASB SCASW SCASD	SCASB	8086	15
	SCASW	8088	19
	SCASD	80286	7
	SCAS DATOSF	80386	7
	REP SCASB	80486	6
		Pentium-Pentium 4	4

SETcondición		Asignación condicional		
Formato	Ejemplos	Microprocesador		Ciclos de reloj
SETcnd reg8	SETA BL SETB CH SETG DL SETE BH SETZ AL	8086	—	
		8088	—	
		80286	—	
		80386	4	
		80486	3	
		Pentium-Pentium 4	1 o 2	
SETcnd mem8	SETE DATOSK SETAE MUCHO_MENOR	8086	—	
		8088	—	
		80286	—	
		80386	5	
		80486	3	
		Pentium-Pentium 4	1 o 2	
Condición				
Códigos	Nemónico	Bandera	Descripción	
0000	SETO	O = 1	Asigna si hay desbordamiento	
0001	SETNO	O = 0	Asigna si no hay desbordamiento	
0010	SETB/SETAE	C = 1	Asigna si está por debajo	
0011	SETAE/SETNB	C = 0	Asigna si está por encima o si es igual	
0100	SETE/SETZ	Z = 1	Asigna si es igual/cero	
0101	SETNE/SETNZ	Z = 0	Asigna si no es igual/cero	
0110	SETBE/SETNA	C = 1 + Z = 1	Asigna si está por debajo o si es igual	
0111	SETA/SETNBE	C = 0 • Z = 0	Asigna si está por encima	
1000	SETS	S = 1	Asigna si hay signo	
1001	SETNS	S = 0	Asigna si no hay signo	
1010	SETP/SETPE	P = 1	Asigna si hay paridad	
1011	SETNP/SETPO	P = 0	Asigna si no hay paridad	
1100	SETL/SETNGE	S • O	Asigna si es menor que	
1101	SETGE/SETNL	S = 0	Asigna si es mayor o igual que	
1110	SETLE/SETNG	Z = 1 + S • O	Asigna si es menor o igual que	
1111	SETG/SETNLE	Z = 0 + S = O	Asigna si es mayor que	

SGDT/SIDT/SLDT Almacena registros de tabla de descriptores			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
SGDT mem	SGDT MEMORIA SGDT GLOBAL	8086	–
		8088	–
		80286	11
		80386	9
		80486	10
		Pentium-Pentium 4	4
00001111 00000001 oo000mmmm desp			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
SIDT mem	SIDT DATOS SIDT INTERRUPCION	8086	–
		8088	–
		80286	12
		80386	9
		80486	10
		Pentium-Pentium 4	4
00001111 00000000 oo000mmmm desp			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
SLDT reg	SLDT CX SLDT DX	8086	–
		8088	–
		80286	2
		80386	2
		80486	2
		Pentium-Pentium 4	2
SLDT mem	SLDT NUMEROS SLDT LOCALES	8086	–
		8088	–
		80286	3
		80386	2
		80486	3
		Pentium-Pentium 4	2

SHLD/SHRD Desplazamiento de doble precisión					
00001111 10100100 oorrrmmm desp datos			O D I T	S Z A P C	
Formato	Ejemplos		Microprocesador	Ciclos de reloj	
SHLD reg,reg,imm	SHLD AX,CX,10 SHLD DX,BX,8 SHLD CX,DX,2		8086	—	
			8088	—	
			80286	—	
			80386	3	
			80486	2	
			Pentium-Pentium 4	4	
SHLD mem,reg,imm	SHLD DATOSQ,CX,8		8086	—	
			8088	—	
			80286	—	
			80386	7	
			80486	3	
			Pentium-Pentium 4	4	
00001111 10101100 oorrrmmm desp datos					
Formato	Ejemplos		Microprocesador	Ciclos de reloj	
SHRD reg,reg,imm	SHRD CX,DX,2		8086	—	
			8088	—	
			80286	—	
			80386	3	
			80486	2	
			Pentium-Pentium 4	4	
SHRD mem,reg,imm	SHRD DATOSZ,DX,4		8086	—	
			8088	—	
			80286	—	
			80386	7	
			80486	2	
			Pentium-Pentium 4	4	

00001111 10100101 oorrrmmm desp			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
SHLD reg,reg,CL	SHLD BX,DX,CL	8086	—
		8088	—
		80286	—
		80386	3
		80486	3
		Pentium-Pentium 4	4 o 5
SHLD mem,reg,CL	SHLD DATOSZ,DX,CL	8086	—
		8088	—
		80286	—
		80386	7
		80486	3
		Pentium-Pentium 4	4 o 5
00001111 10101101 oorrrmmm desp			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
SHRD reg,reg,CL	SHRD AX,DX,CL	8086	—
		8088	—
		80286	—
		80386	3
		80486	3
		Pentium-Pentium 4	4 o 5
SHRD mem,reg,CL	SHRD DATOSZ,DX,CL	8086	—
		8088	—
		80286	—
		80386	7
		80486	3
		Pentium-Pentium 4	4 o 5

SMSW Almacena palabra de estado del equipo (80286)				
00001111 00000001 oo100mmm desp		Microprocesador	Ciclos de reloj	
Formato	Ejemplos			
SMSW reg	SMSW AX SMSW DX SMSW BP	8086	–	
		8088	–	
		80286	2	
		80386	10	
		80486	2	
		Pentium-Pentium 4	4	
SMSW mem	SMSW DATOSQ	8086	–	
		8088	–	
		80286	3	
		80386	3	
		80486	3	
		Pentium-Pentium 4	4	
STC Activa bandera de acarreo				
11111001		O D I T 1	S Z A P C	
Ejemplo		Microprocesador	Ciclos de reloj	
STC		8086	2	
		8088	2	
		80286	2	
		80386	2	
		80486	2	
		Pentium-Pentium 4	2	

STD	Activa bandera de dirección			
11111101		O D I T	S Z A P C	
Ejemplo		1	Microprocesador	Ciclos de reloj
STD		8086	2	
		8088	2	
		80286	2	
		80386	2	
		80486	2	
		Pentium-Pentium 4	2	
STI	Activa bandera de interrupción			
11111001		O D I T	S Z A P C	
Ejemplo		1	Microprocesador	Ciclos de reloj
STI		8086	2	
		8088	2	
		80286	2	
		80386	3	
		80486	5	
		Pentium-Pentium 4	7	
STOS	Almacena datos tipo cadena de texto			
1010101w				
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
STOSB	STOSB	8086	11	
STOSW	STOSW	8088	15	
STOSD	STOSD STOS LISTA_DATOS REP STOSB	80286 80386 80486 Pentium-Pentium 4	3 40 5 3	

STR Almacena registro de tareas			
00001111 00000000 oo001mmm desp			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
STR reg	STR AX	8086	—
	STR DX	8088	—
	STR BP	80286	2
		80386	2
		80486	2
		Pentium-Pentium 4	2
STR mem	STR DATOS3	8086	—
		8088	—
		80286	2
		80386	2
		80486	2
		Pentium-Pentium 4	2
SUB Resta			
000101dw oorrrmmm desp		O D I T	S Z A P C
		*	* * * * *
Formato	Ejemplos	Microprocesador	Ciclos de reloj
SUB reg,reg	SUB CL,DL	8086	3
	SUB AX,DX	8088	3
	SUB CH,CL	80286	2
	SUB EAX,EBX	80386	2
	SUB ESI,EDI	80486	1
		Pentium-Pentium 4	1 o 2

SUB mem,reg	SUB DATOSJ,CL	8086	16 + ea
	SUB BYTES,CX	8088	24 + ea
	SUB NUMEROS,ECX	80286	7
	SUB [EAX],ECX	80386	6
		80486	3
		Pentium-Pentium 4	1 o 3
SUB reg,mem	SUB CL,DATOSL	8086	9 + ea
	SUB CX,BYTES	8088	13 + ea
	SUB ECX,NUMEROS	80286	7
	SUB DX,[EBX+EDI]	80386	7
		80486	2
		Pentium-Pentium 4	1 o 2
100000sw oo101mmmm desp datos			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
SUB reg,imm	SUB CX,3	8086	4
	SUB DI,1AH	8088	4
	SUB DL,34H	80286	3
	SUB EDX,1345H	80386	2
	SUB CX,1834H	80486	1
		Pentium-Pentium 4	1 o 3
SUB mem,imm	SUB DATOS,3	8086	17 + ea
	SUB BYTE PTR[EDI],1AH	8088	25 + ea
	SUB PAPI,34H	80286	7
	SUB LISTA,'A'	80386	7
	SUB RANITA,1834H	80486	3
		Pentium-Pentium 4	1 o 3

Formato	Ejemplos	Microprocesador	Ciclos de reloj	
SUB acu,imm	SUB AL,3 SUB AX,1AH SUB EAX,34H	8086	4	
		8088	4	
		80286	3	
		80386	2	
		80486	1	
		Pentium-Pentium 4	1	
TEST Prueba operandos (comparación lógica)				
1000001w oorrrmmmm desp		O D I T	S Z A P C	
Formato	Ejemplos	0	* * ? * 0	
		Microprocesador	Ciclos de reloj	
TEST reg,reg	TEST CL,DL TEST BX,DX TEST DH,CL TEST EBP,EBX TEST EAX,EDI	8086	5	
		8088	5	
		80286	2	
		80386	2	
		80486	1	
		Pentium-Pentium 4	1 o 2	
TEST mem,reg reg, mem	TEST DATOSJ,CL TEST BYTES,CX TEST NUMEROS,ECX TEST [EAX],CX TEST CL,POPS	8086	9 + ea	
		8088	13 + ea	
		80286	6	
		80386	5	
		80486	2	
		Pentium-Pentium 4	1 o 2	

1111011sw oo000mmm desp datos			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
TEST reg,imm	TEST BX,3	8086	4
	TEST DI,1AH	8088	4
	TEST DH,44H	80286	3
	TEST EDX,1AB345H	80386	2
	TEST SI,1834H	80486	1
		Pentium-Pentium 4	1 o 2
TEST mem,imm	TEST DATOS,3	8086	11 + ea
	TEST BYTE PTR[EDI],1AH	8088	11 + ea
	TEST PAPI,34H	80286	6
	TEST LISTA,'A'	80386	5
	TEST RANITA,1834H	80486	2
		Pentium-Pentium 4	1 o 2
1010100w datos			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
TEST acu,imm	TEST AL,3	8086	4
	TEST AX,1AH	8088	4
	TEST EAX,34H	80286	3
		80386	2
		80486	1
		Pentium-Pentium 4	1

VERR/VERW Verificación de lectura/escritura				
00001111 00000000 oo100mmm desp		O D I T	S Z A P C	*
Formato	Ejemplos	Microprocesador		Ciclos de reloj
VERR reg	VERR CX VERR DX VERR DI	8086	—	
		8088	—	
		80286	14	
		80386	10	
		80486	11	
		Pentium-Pentium 4	7	
VERR mem	VERR DATOSJ VERR PRUEBAB	8086	—	
		8088	—	
		80286	16	
		80386	11	
		80486	11	
		Pentium-Pentium 4	7	
00001111 00000000 oo101mmm desp		Microprocesador		Ciclos de reloj
Formato	Ejemplos	Microprocesador		Ciclos de reloj
VERW reg	VERW CX VERW DX VERW DI	8086	—	
		8088	—	
		80286	14	
		80386	15	
		80486	11	
		Pentium-Pentium 4	7	
VERW mem	VERW DATOSJ VERW PRUEBAB	8086	—	
		8088	—	
		80286	16	
		80386	16	
		80486	11	
		Pentium-Pentium 4	7	

WAIT	Espera para el coprocesador	
10011011 Ejemplo	Microprocesador	Ciclos de reloj
WAIT	8086	4
FWAIT	8088	4
	80286	3
	80386	6
	80486	6
	Pentium-Pentium 4	1
WBINVD	Caché de escritura aplazada invalida caché de datos	
00001111 00001001 Ejemplo	Microprocesador	Ciclos de reloj
WBINVD	8086	–
	8088	–
	80286	–
	80386	–
	80486	5
	Pentium-Pentium 4	2000+
WRMSR	Escribe en registro específico del modelo	
00001111 00110000 Ejemplo	Microprocesador	Ciclos de reloj
WRMSR	8086	–
	8088	–
	80286	–
	80386	–
	80486	–
	Pentium-Pentium 4	30-45

XADD Intercambio y suma				
00001111 1100000w 11rrrrr		O D I T	S Z A P C	
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
XADD reg,reg	XADD EBX,ECX XADD EDX,EAX XADD EDI,EBP	8086	—	
		8088	—	
		80286	—	
		80386	—	
		80486	3	
		Pentium-Pentium 4	3 o 4	
00001111 1100000w oorrrmmm desp				
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
XADD mem,reg	XADD DATOS5,ECX XADD [EBX],EAX XADD [ECX+4],EBP	8086	—	
		8088	—	
		80286	—	
		80386	—	
		80486	4	
		Pentium-Pentium 4	3 o 4	
XCHG Intercambio				
1000011w oorrrmmm		Microprocesador	Ciclos de reloj	
Formato	Ejemplos	Microprocesador	Ciclos de reloj	
XCHG reg,reg	XCHG CL,DL XCHG BX,DX XCHG DH,CL XCHG EBP,EBX XCHG EAX,EDI	8086	4	
		8088	4	
		80286	3	
		80386	3	
		80486	3	
		Pentium-Pentium 4	3	

XCHG mem,reg reg,mem	XCHG DATOSJ,CL	8086	17 + ea
	XCHG BYTES,CX	8088	25 + ea
	XCHG NUMEROS,ECX		
	XCHG [EAX],CX	80286	5
	XCHG CL,POPS	80386	5
		80486	5
		Pentium-Pentium 4	3
10010reg Formato	Ejemplos	Microprocesador	Ciclos de reloj
XCHG acu,reg reg,acu	XCHG BX,AX	8086	3
	XCHG AX,DI	8088	3
	XCHG DH,AL	80286	3
	XCHG EDX,EAX	80386	3
	XCHG SI,AX	80486	3
		Pentium-Pentium 4	2
XLAT	Traducción		
11010111 Ejemplo		Microprocesador	Ciclos de reloj
XLAT		8086	11
		8088	11
		80286	5
		80386	3
		80486	4
		Pentium-Pentium 4	4

XOR OR exclusivo			
000110dw oorrrmmm desp		O D I T	S Z A P C
Formato	Ejemplos	Microprocesador	Ciclos de reloj
XOR reg,reg	XOR CL,DL XOR AX,DX XOR CH,CL XOR EAX,EBX XOR ESI,EDI	8086	3
		8088	3
		80286	2
		80386	2
		80486	1
		Pentium-Pentium 4	1 o 2
XOR mem,reg	XOR DATOSJ,CL XOR BYTES,CX XOR NUMEROS,ECX XOR [EAX],CX	8086	16 + ea
		8088	24 + ea
		80286	7
		80386	6
		80486	3
		Pentium-Pentium 4	1 o 3
XOR reg,mem	XOR CL,DATOSL XOR CX,BYTES XOR ECX,NUMEROS XOR DX,[EBX+EDI]	8086	9 + ea
		8088	13 + ea
		80286	7
		80386	7
		80486	2
		Pentium-Pentium 4	1 o 2
100000sw oo110mmm desp datos			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
XOR reg,imm	XOR CX,3 XOR DI,1AH XOR DL,34H XOR EDX,1345H XOR CX,1834H	8086	4
		8088	4
		80286	3
		80386	2
		80486	1
		Pentium-Pentium 4	1 o 3

XOR mem,imm	XOR DATOS,3	8086	17 + ea
	XOR BYTE PTR[EDI],1AH	8088	25 + ea
	XOR PAPI,34H	80286	7
	XOR LISTA,'A'	80386	7
	XOR RANITA,1834H	80486	3
		Pentium-Pentium 4	1 o 3
0010101w datos			
Formato	Ejemplos	Microprocesador	Ciclos de reloj
XOR acu,imm	XOR AL,3	8086	4
	XOR AX,1AH	8088	4
	XOR EAX,34H	80286	3
		80386	2
		80486	1
		Pentium-Pentium 4	1

RESUMEN DEL CONJUNTO DE INSTRUCCIONES SIMD

Las instrucciones SIMD (una sola instrucción, múltiples datos) agrega una nueva dimensión al uso del microprocesador para realizar operaciones de multimedia y otro tipo de operaciones. Los registros XMM se enumeran del XMM_0 al XMM_7 y tienen 128 bits de anchura cada uno. En la figura B-1 se muestran los formatos de datos que se almacenan en los registros XMM y que utilizan las instrucciones SIMD.

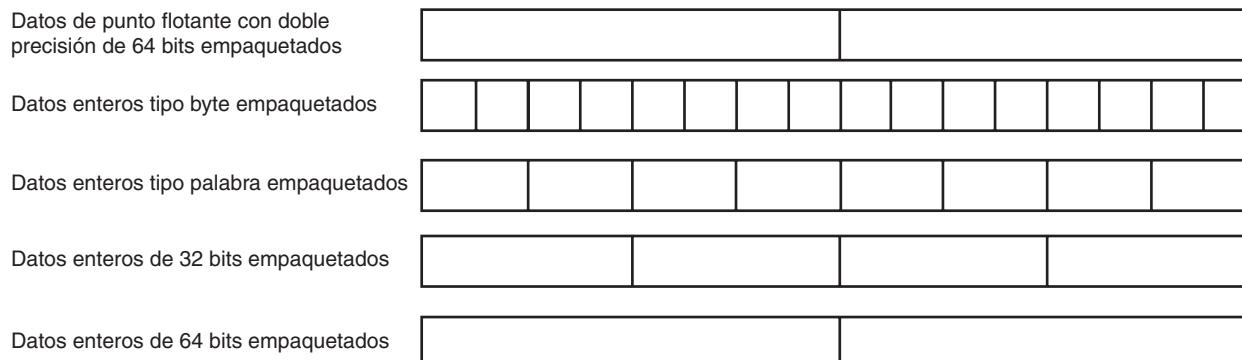


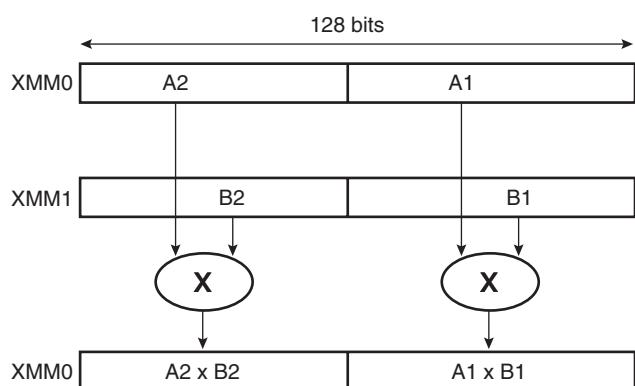
FIGURA B-1 Formatos de datos para los registros XMM de 128 bits en los microprocesadores Pentium III y Pentium 4.

En la memoria, los datos deben almacenarse como datos de 16 bytes, en una serie de posiciones de memoria a las cuales se accede mediante el uso de la sustitución OWORD PTR cuando se direccionan mediante una instrucción. La sustitución OWORD PTR se utiliza para direccionar una palabra octal de datos, o 16 bytes. Las instrucciones SIMD permiten operaciones con números de punto flotante con doble precisión, empaquetados y escalares. En la figura B-2 se muestra la operación de ambos formatos, en donde se muestran la multiplicación empaquetada y la multiplicación escalar. Tenga en cuenta que la instrucción escalar sólo copia el número de doble precisión de más a la izquierda en el registro de destino; no utiliza el número de más a la izquierda que está en el origen. Las instrucciones escalares fueron diseñadas para ser compatibles con las instrucciones del coprocesador de punto flotante.

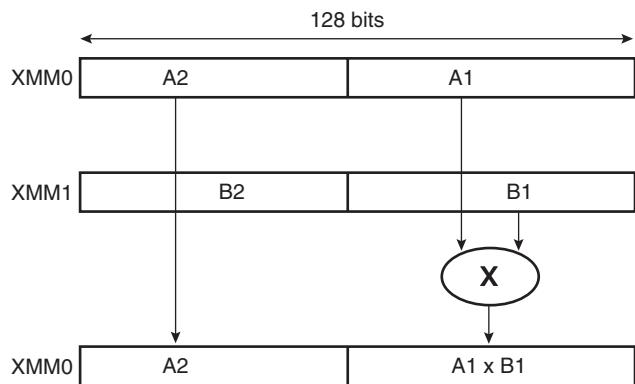
En esta sección del apéndice veremos los detalles acerca de muchas de las instrucciones SIMD y proporcionaremos ejemplos de su uso.

FIGURA B-2 Operación de punto flotante con doble precisión, en formato empaquetado y escalar.

Multiplicación de doble precisión empaquetada MULPD XMM0,XMM1



Multiplicación de doble precisión empaquetada MULSD XMM0,XMM1



INSTRUCCIONES DE MOVIMIENTO DE DATOS

MOVAPD	Mueve datos empaquetados y alineados de doble precisión; los datos deben alinearse en límites de 16 bytes
Ejemplos	
MOVAPD XMM0, OWORD DATOS3 ;copia DATOS3 a XMM0	
MOVAPD OWORD PTR DATOS4,XMM2 ;copia XMM4 a DATOS4	
MOVUPD	Mueve datos empaquetados y desalineados de doble precisión
Ejemplos	
MOVUPD XMM0, OWORD DATOS3 ;copia DATOS3 a XMM0	
MOVUPD OWORD PTR DATOS4,XMM2 ;copia XMM4 a DATOS4	
MOVSD	Mueve datos escalares empaquetados de doble precisión hacia una palabra cuádruple inferior
Ejemplos	
MOVSD XMM0, DWORD DATOS3 ;copia DATOS3 A XMM0	
MOVSD DWORD PTR DATOS4,XMM2 ;copia XMM4 a DATOS4	
MOVHPD	Mueve datos empaquetados de doble precisión hacia una palabra cuádruple superior
Ejemplos	
MOVHPD XMM0, DWORD DATOS3 ;copia DATOS3 A XMM0	
MOVHPD DWORD PTR DATOS4,XMM2 ;copia XMM4 a DATOS4	
MOVLPD	Mueve datos empaquetados de doble precisión hacia una palabra cuádruple inferior
Ejemplos	
MOVLPD XMM0, DWORD DATOS3 ;copia DATOS3 A XMM0	
MOVLPD DWORD PTR DATOS4,XMM2 ;copia XMM4 a DATOS4	
MOVMSKPD	Mueve máscara empaquetada de doble precisión
Ejemplos	
MOVMSKPD EAX,XMM1 ;copia 2 bits de signo en un registro de propósito general	

MOVAPS	Mueve 4 datos empaquetados y alineados de precisión simple; los datos deben alinearse en límites de 16 bytes
Ejemplos	
MOVAPS XMM0, OWORD DATOS3	;copia DATOS3 a XMM0
MOVAPS OWORD PTR DATOS4,XMM2	;copia XMM4 a DATOS4
MOVUPS Mueve 4 datos empaquetados y desalineados de precisión simple	
Ejemplos	
MOVUPS XMM0, OWORD DATOS3	;copia DATOS3 a XMM0
MOVUPS OWORD PTR DATOS4,XMM2	;copia XMM4 a DATOS4
MOVLPS Mueve 2 números empaquetados de precisión simple hacia una palabra cuádruple de menor orden	
Ejemplos	
MOVLPS XMM0, OWORD DATOS3	;copia DATOS3 a XMM0
MOVLPS OWORD PTR DATOS4,XMM2	;copia XMM4 a DATOS4
MOVHPS Mueve números empaquetados de precisión simple hacia una palabra cuádruple de mayor orden	
Ejemplos	
MOVHPS XMM0, OWORD DATOS3	;copia DATOS3 a XMM0
MOVHPS OWORD PTR DATOS4,XMM2	;copia XMM4 a DATOS4
MOVAPD Mueve datos empaquetados y alineados de doble precisión; los datos deben alinearse en límites de 16 bytes	
Ejemplos	
MOVAPD XMM0, OWORD DATOS3	;copia DATOS3 a XMM0
MOVAPD OWORD PTR DATOS4,XMM2	;copia XMM4 a DATOS4
MOVLHPS Mueve 2 números empaquetados de precisión simple, desde la palabra cuádruple de menor orden hacia la de mayor orden	
Ejemplos	
MOVLHPS XMM0,XMM1	;copia XMM1 inferior a XMM0 superior
MOVLHPS XMM3,XMM2	;copia XMM2 inferior a XMM3 superior

MOVHLPS	Mueve 2 números empaquetados de precisión simple, desde la palabra cuádruple de mayor orden hacia la de menor orden
Ejemplos	
MOVHLPS XMM0,XMM2	;copia XMM2 superior a XMM0 inferior
MOVHLPS XMM4,XMM5	;copia XMM5 superior a XMM4 inferior
MOVMSKPS Mueve 4 bits de signo de 4 números empaquetados de precisión simple hacia un registro de propósito general	
Ejemplos	
MOVMSKPS EBX,XMM0	;copia los bits de signo de XMM0 a EBX
MOVMSKPS EDX,XMM2	;copia los bits de signo de XMM2 a EDX

INSTRUCCIONES ARITMÉTICAS

ADDPD	Suma datos empaquetados de doble precisión
Ejemplos	
ADDPD XMM0,OWORD DATOS3	;suma DATOS3 a XMM0
ADDPD XMM2,XMM3	;suma XMM3 a XMM2
ADDSD Suma datos escalares de doble precisión	
Ejemplos	
ADDSD XMM0,OWORD DATOS3	;suma DATOS3 a XMM0
ADDSD XMM4,XMM2	;suma XMM2 a XMM4
ADDPS Suma 2 números empaquetados de precisión simple	
Ejemplos	
ADDPS XMM0,QWORD DATOS3	;suma DATOS3 a XMM0
ADDPS XMM3,XMM2	;suma XMM2 a XMM3

ADDLS	Suma datos escalares de precisión simple	
Ejemplos		
ADDLS	XMM0,DWORD DATOS3	;suma DATOS3 a XMM0
ADDLS	XMM7,XMM2	;suma XMM2 a XMM7
SUBPD	Resta datos empaquetados de doble precisión	
Ejemplos		
SUBPD	XMM0,OWORD DATOS3	;resta DATOS3 a XMM0
SUBPD	XMM2,XMM3	;resta XMM3 a XMM2
SUBSD	Resta datos escalares de doble precisión	
Ejemplos		
SUBSD	XMM0,OWORD DATOS3	;resta DATOS3 a XMM0
SUBSD	XMM4,XMM2	;resta XMM2 a XMM4
SUBPS	Resta 2 números empaquetados de precisión simple	
Ejemplos		
SUBPS	XMM0,QWORD DATOS3	;resta DATOS3 a XMM0
SUBPS	XMM3,XMM2	;resta XMM2 a XMM3
SUBLS	Resta datos escalares de precisión simple	
Ejemplos		
SUBLS	XMM0,DWORD DATOS3	;resta DATOS3 a XMM0
SUBLS	XMM7,XMM2	;resta XMM2 a XMM7
MULPD	Multiplica datos empaquetados de doble precisión	
Ejemplos		
MULPD	XMM0,OWORD DATOS3	;multiplica DATOS3 por XMM0
MULPD	XMM3,XMM2	;multiplica XMM2 por XMM3

MULSD	Multiplica datos empaquetados de doble precisión
Ejemplos	
MULSD XMM0,OWORD DATOS3	;multiplica DATOS3 por XMM0
MULSD XMM3,XMM6	;multiplica XMM6 por XMM3
MULPS	Multiplica 2 números empaquetados de precisión simple
Ejemplos	
MULPS XMM0,QWORD DATOS3	;multiplica DATOS3 por XMM0
MULPS XMM0,XMM2	;multiplica XMM2 por XMM0
MULSS	Multiplica un número de precisión simple
Ejemplos	
MULSS XMM0,DWORD DATOS3	;multiplica DATOS3 por XMM0
MULSS XMM1,XMM2	;multiplica XMM2 por XMM1
DIVPD	Divide datos empaquetados de doble precisión
Ejemplos	
DIVPD XMM0,OWORD DATOS3	;divide XMM0 entre DATOS3
DIVPD XMM3,XMM2	;divide XMM3 entre XMM2
DIVSD	Divide datos escalares de doble precisión
Ejemplos	
DIVSD XMM0,OWORD DATOS3	;divide XMM0 entre DATOS3
DIVSD XMM3,XMM6	;divide XMM3 entre XMM6
DIVPS	Divide 2 números empaquetados de precisión simple
Ejemplos	
DIVPS XMM0,QWORD DATOS3	;divide XMM0 entre DATOS3
DIVPS XMM0,XMM2	;divide XMM0 entre XMM2
DIVSS	Divide un número de precisión simple
Ejemplos	
DIVSS XMM0,DWORD DATOS3	;divide XMM0 entre DATOS3
DIVSS XMM1,XMM2	;divide XMM1 entre XMM2

SQRTPD	Encuentra la raíz cuadrada de datos empaquetados de doble precisión	
Ejemplos		
SQRTPD	XMM0,OWORD DATOS3	;encuentra raíz cuadrada de DATOS3, pone el resultado en XMM0
SQRTPD	XMM3,XMM2	;encuentra raíz cuadrada de XMM2, pone el resultado en XMM3
SQRTSD	Encuentra la raíz cuadrada de datos escalares de doble precisión	
Ejemplos		
SQRTSD	XMM0,OWORD DATOS3	;encuentra raíz cuadrada de DATOS3, pone el resultado en XMM0
SQRTSD	XMM3,XMM6	;encuentra raíz cuadrada de XMM6, pone el resultado en XMM3
SQRTPS	Encuentra la raíz cuadrada de 2 números empaquetados de precisión simple	
Ejemplos		
SQRTPS	XMM0,QWORD DATOS3	;encuentra raíz cuadrada de DATOS3, pone el resultado en XMM0
SQRTPS	XMM0,XMM2	;encuentra raíz cuadrada de XMM2, pone el resultado en XMM0
SQRTSS	Encuentra la raíz cuadrada de un número de precisión simple	
Ejemplos		
SQRTSS	XMM0,DWORD DATOS3	;encuentra raíz cuadrada de DATOS3, pone el resultado en XMM0
SQRTSS	XMM1,XMM2	;encuentra raíz cuadrada de XMM2, pone el resultado en XMM1
RCPPS	Encuentra el recíproco de un número empaquetado de precisión simple	
Ejemplos		
RCPPS	XMM0,OWORD DATOS3	;encuentra el recíproco de DATOS3, pone el resultado en XMM0
RCPPS	XMM3,XMM2	;encuentra el recíproco de XMM2, pone el resultado en XMM3

RCPSS	Encuentra el recíproco de un número de precisión simple	
Ejemplos		
RCPSS	XMM0,OWORD DATOS3	;encuentra el recíproco de DATOS3, pone el resultado en XMM0
RCPSS	XMM3,XMM6	;encuentra el recíproco de XMM6, pone el resultado en XMM3
RSQRTPS	Encuentra los recíprocos de datos empaquetados de precisión simple	
Ejemplos		
RSQRTPS	XMM0,OWORD DATOS3	;encuentra el recíproco de la raíz cuadrada de DATOS3
RSQRTPS	XMM3,XMM2	;encuentra el recíproco de la raíz cuadrada de XMM2
RSQRTSS	Encuentra el recíproco de la raíz cuadrada de un número escalar de precisión simple	
Ejemplos		
RSQRTSS	XMM0,OWORD DATOS3	;encuentra el recíproco de la raíz cuadrada de DATOS3
RSQRTSS	XMM3,XMM6	;encuentra el recíproco de la raíz cuadrada de XMM6
MAXPD	Compara y devuelve el máximo número empaquetado de punto flotante con doble precisión	
Ejemplos		
MAXPD	XMM0,OWORD DATOS3	;compara los números en DATOS3, el mayor se coloca en XMM0
MAXPD	XMM3,XMM2	;compara los números en XMM2, el mayor se coloca en XMM3
MAXSD	Compara datos escalares de doble precisión y devuelve el más grande	
Ejemplos		
MAXSD	XMM0,OWORD DATOS3	;compara los números en DATOS3, el mayor se coloca en XMM0
MAXSD	XMM3,XMM6	;compara los números en XMM6, el mayor se coloca en XMM3

MAXPS	Compara y devuelve el número empaquetado de precisión simple más grande	
Ejemplos		
MAXPS	XMM0,QWORD DATOS3	;compara los números en DATOS3, el mayor se coloca en XMM0
MAXPS	XMM0,XMM2	;compara los números en XMM2, el mayor se coloca en XMM0
MAXSS	Compara números escalares de precisión simple y devuelve el más grande	
Ejemplos		
MAXSS	XMM0,DWORD DATOS3	;compara los números en DATOS3, el mayor se coloca en XMM0
MAXSS	XMM1,XMM2	;compara los números en XMM2, el mayor se coloca en XMM1
MINPD	Compara y devuelve el mínimo número empaquetado de punto flotante con doble precisión	
Ejemplos		
MINPD	XMM0,OWORD DATOS3	;compara los números en DATOS3, el menor se coloca en XMM0
MINPD	XMM3,XMM2	;compara los números en XMM2, el menor se coloca en XMM3
MINSD	Compara datos escalares de doble precisión y devuelve el más pequeño	
Ejemplos		
MINSD	XMM0,OWORD DATOS3	;compara los números en DATOS3, el menor se coloca en XMM0
MINSD	XMM3,XMM6	;compara los números en XMM6, el menor se coloca en XMM3
MINPS	Compara y devuelve el número empaquetado de precisión simple más pequeño	
Ejemplos		
MINPS	XMM0,QWORD DATOS3	;compara los números en DATOS3, el menor se coloca en XMM0
MINPS	XMM1,XMM2	;compara los números en XMM2, el menor se coloca en XMM1

MINSS	Compara números escalares de precisión simple y devuelve el más grande	
Ejemplos		
MINSS	XMM0,DWORD DATOS3	;compara los números en DATOS3, el menor se coloca en XMM0
MINSS	XMM1,XMM2	;compara los números en XMM2, el menor se coloca en XMM1

INSTRUCCIONES LÓGICAS

ANDPD	Realiza una operación AND con datos empaquetados de doble precisión	
Ejemplos		
ANDPD	XMM0,OWORD DATOS3	;realiza un AND entre DATOS3 y XMM0
ANDPD	XMM2,XMM3	;realiza un AND entre XMM3 y XMM2
ANDNPD Realiza una operación NAND con datos empaquetados de doble precisión		
Ejemplos		
ANDNPD	XMM0,OWORD DATOS3	;realiza un NAND entre DATOS3 y XMM0
ANDNPD	XMM4,XMM2	;realiza un NAND entre XMM2 y XMM4
ANDPS	Realiza una operación AND con 2 datos de precisión simple	
Ejemplos		
ANDPS	XMM0,QWORD DATOS3	;realiza un AND entre DATOS3 y XMM0
ANDPS	XMM3,XMM2	;realiza un AND entre XMM2 y XMM3
ANDNPS	Realiza una operación NAND con 2 datos empaquetados de precisión simple	
Ejemplos		
ANDNPS	XMM0,DWORD DATOS3	;realiza un NAND entre DATOS3 y XMM0
ANDNPS	XMM7,XMM2	;realiza un NAND entre XMM2 y XMM7
ORPD	Realiza una operación OR con datos empaquetados de doble precisión	
Ejemplos		
ORPD	XMM0,OWORD DATOS3	;realiza un OR entre DATOS3 y XMM0
ORPD	XMM2,XMM3	;realiza un OR entre XMM3 y XMM2

ORPS	Realiza una operación OR con 2 números empaquetados de precisión simple	
Ejemplos		
ORPS	XMM0,OWORD DATOS3	;realiza un OR entre DATOS3 y XMM0
ORPS	XMM3,XMM2	;realiza un OR entre XMM2 y XMM3
XORPD	Realiza una operación OR exclusivo con datos empaquetados de doble precisión	
Ejemplos		
XORPD	XMM0,OWORD DATOS3	;realiza un OR exclusivo entre DATOS3 y XMM0
XORPD	XMM2,XMM3	;realiza un OR exclusivo entre XMM3 y XMM2
XORPS	Realiza una operación OR exclusivo con datos empaquetados de doble precisión	
Ejemplos		
XORPS	XMM0,OWORD DATOS3	;realiza un OR exclusivo entre DATOS3 y XMM0
XORPS	XMM2,XMM3	;realiza un OR exclusivo entre XMM3 y XMM2

INSTRUCCIONES DE COMPARACIÓN

CMPPD	Compara números empaquetados de doble precisión	
Ejemplos		
CMPPD	XMM0,OWORD DATOS3	;compara DATOS3 con XMM0
CMPPD	XMM2,XMM3	;compara XMM3 con XMM2
CMPSD	Compara datos escalares de doble precisión	
Ejemplos		
CMPSD	XMM0,QWORD DATOS3	;compara DATOS3 con XMM0
CMPSD	XMM3,XMM2	;compara XMM2 con XMM3
CMPISD	Compara datos escalares de doble precisión y establece EFLAGS	
Ejemplos		
CMPISD	XMM0,OWORD DATOS3	;compara DATOS3 con XMM0
CMPISD	XMM2,XMM3	;compara XMM3 con XMM2

UCOMISD	Compara números escalares desordenados de doble precisión y modifica EFLAGS
Ejemplos	
UCOMISD XMM0,QWORD DATOS3	;compara DATOS3 con XMM0
UCOMISD XMM3,XMM2	;compara XMM2 con XMM3
CMPPS Compara datos empaquetados de precisión simple	
Ejemplos	
CMPPS XMM0,OWORD DATOS3	;compara DATOS3 con XMM0
CMPPS XMM2,XMM3	;compara XMM3 con XMM2
CMPSS Compara 2 números empaquetados de precisión simple	
Ejemplos	
CMPSS XMM0,QWORD DATOS3	;compara DATOS3 con XMM0
CMPSS XMM3,XMM2	;compara XMM2 con XMM3
COMISS Compara datos escalares de precisión simple y modifica EFLAGS	
Ejemplos	
COMISS XMM0,OWORD DATOS3	;compara DATOS3 con XMM0
COMISS XMM2,XMM3	;compara XMM3 con XMM2
UCOMISS Compara números desordenados de precisión simple y modifica EFLAGS	
Ejemplos	
UCOMISS XMM0,QWORD DATOS3	;compara DATOS3 con XMM0
UCOMISS XMM3,XMM2	;compara XMM2 con XMM3

INSTRUCCIONES DE CONVERSIÓN DE DATOS

SHUFPD	Revuelve números empaquetados de doble precisión
Ejemplos	
SHUFPD XMM0,OWORD DATOS3	;revuelve DATOS3 con XMM0
SHUFPD XMM2,XMM2	;intercambia la palabra cuádruple superior e inferior en XMM2
UNPCKHPD Desempaquetar el número superior de doble precisión	
Ejemplos	
UNPCKHPD XMM0,OWORD DATOS3	;desempaquetar DATOS3 en XMM0
UNPCKHPD XMM3,XMM2	;desempaquetar XMM2 en XMM3
UNPCKLPD Desempaquetar el número inferior de doble precisión	
Ejemplos	
UNPCKLPD XMM0,OWORD DATOS3	;desempaquetar DATOS3 en XMM0
UNPCKLPD XMM3,XMM2	;desempaquetar XMM2 en XMM3
SHUFPS Revuelve números empaquetados de precisión simple	
Ejemplos	
SHUFPS XMM0,QWORD DATOS3	;revuelve DATOS3 con XMM0
SHUFPS XMM2,XMM2	;intercambia la palabra cuádruple superior e inferior en XMM2
UNPCKHPS Desempaquetar el número inferior de doble precisión	
Ejemplos	
UNPCKHPS XMM0,QWORD DATOS3	;desempaquetar DATOS3 en XMM0
UNPCKHPS XMM3,XMM2	;desempaquetar XMM2 en XMM3
UNPCKLPSD Desempaquetar el número inferior de doble precisión	
Ejemplos	
UNPCKLPSD XMM0,QWORD DATOS3	;desempaquetar DATOS3 en XMM0
UNPCKLPSD XMM3,XMM2	;desempaquetar XMM2 en XMM3

APÉNDICE C

Modificaciones del bit de bandera

En este apéndice sólo se muestran las instrucciones que modifican los bits de bandera. Cualquier instrucción que no esté listada no afecta a ninguno de los bits de bandera.

Instrucción	O	D	I	T	S	Z	A	P	C
AAA	?				?	?	*	?	*
?				*	*	?	*	?	
AAM	?				*	*	?	*	?
AAS	?				?	?	*	?	*
ADC	*				*	*	*	*	*
ADD	*				*	*	*	*	*
AND	0				*	*	?	*	0
ARPL						*			
BSF						*			
BSR						*			
BT									*
BTC									*
BTR									*
BTS									*
CLC									0
CLD		0							
CLI			0						
CMC									*
CMP	*				*	*	*	*	*
CMPS	*				*	*	*	*	*
CMPXCHG	*				*	*	*	*	*
CMPXCHG8B					*				
DAA	?				*	*	*	*	*
DAS	?				*	*	*	*	*
DEC	*				*	*	*	*	
DIV	?				?	?	?	?	?
IDIV	?				?	?	?	?	?
IMUL	*				?	?	?	?	*
INC	*				*	*	*	*	

Instrucción	<i>O</i>	<i>D</i>	<i>I</i>	<i>T</i>	<i>S</i>	<i>Z</i>	<i>A</i>	<i>P</i>	<i>C</i>
IRET	*	*	*	*	*	*	*	*	*
LAR						*			
LSL						*			
MUL	*				?	?	?	?	*
NEG	*				*	*	*	*	*
OR	0				*	*	?	*	0
POPF	*	*	*	*	*	*	*	*	*
RCL/RCR	*								*
REPE/REPNE						*			
ROL/ROR	*								*
SAHF					*	*	*	*	*
SAL/SAR	*				*	*	?	*	*
SHL/SHR	*				*	*	?	*	*
SBB	*				*	*	*	*	*
SCAS	*				*	*	*	*	*
SHLD/SHRD	?				*	*	?	*	*
STC									1
STD		1							
STI			1						
SUB	*				*	*	*	*	*
TEST	0				*	*	?	*	0
VERR/VERW						*			
XADD	*				*	*	*	*	*
XOR	0				*	*	?	*	0

APÉNDICE D

Respuestas a preguntas y problemas seleccionados con número par

CAPÍTULO 1

2. Herman Hollerith.
4. Konrad Zuse.
6. ENIAC.
8. Augusta Ada Byron.
10. Una máquina que almacena las instrucciones de un programa en el sistema de memoria.
12. 200 millones.
14. 16 Mbytes.
16. 1993.
18. 2000.
20. Millones de instrucciones por segundo.
22. Un bit binario almacena un 1 o un 0.
24. 1024 K.
26. 1,000,000.
28. 2 Gbytes.
30. 2 Gbytes.
32. 4 Gbytes o, si está habilitado el modo de direccionamiento de 36 bits, 64 Gbytes.
34. XMS o sistema de memoria extendida.
36. El sistema operativo en disco.
38. Un bus de 32 bits ahora extinto, diseñado en especial para el video y las unidades de disco duro.
40. Bus serial universal.
42. Sistema de memoria extendida.
44. 64 Kbytes.
46. Vea la figura 1-6.
48. Buses de direcciones, de control y de datos.
50. MRDC
52. Operación de lectura de memoria.
54. (a) número con signo de 8 bits (b) número con signo de 16 bits (c) número con signo de 32 bits (d) número de punto flotante de 32 bits (e) número de punto flotante de 64 bits.
56. (a) 156.625 (b) 18.375 (c) 4087.109375 (d) 83.578125 (e) 58.90625.

58. (a) 10111_2 , 27_8 y 17_{16} (b) 1101011_2 , 153_8 y $6B$ (c) 10011010110_2 , 2326_8 y $4D6_{16}$ (d) 1011100_2 , 134_8 y $5C_{16}$ (e) 10101101_2 , 255_8 y AD
60. (a) 0010 0011 (b) 1010 1101 0100 (c) 0011 0100. 1010 1101 (d) 1011 1101 0011 0010 (e) 0010 0011 0100. 0011
62. (a) 0111 0111 (b) 1010 0101 (c) 1000 1000 (d) 0111 1111
64. Byte es un número binario de 8 bits, palabra es un número binario de 16 bits, doble palabra es un número binario de 32 bits.
66. Intro es un 0DH y se utiliza para regresar el cursor/cabeza de impresión al margen izquierdo de la pantalla o página de papel.
68. LINEA1 DB ¿'Que hora es?'
70. (a) 0000 0011 1110 1000 (b) 1111 1111 1000 1000 (c) 0000 0011 0010 0000 (d) 1111 0011 0111 0100
72. char Fred1 = -34
74. Los números Little Endian se almacenan de manera que la porción menos significativa se encuentre en la posición de memoria de menor numeración, en tanto que los números Big Endian se almacenan de manera que la parte más significativa se encuentre en la posición de memoria de menor numeración.
76. (a) empaquetado = 00000001 00000010 y desempaquetado 0000001 00000000 00000010 (b) empaquetado = 01000100 y desempaquetado 00000100 00000100 (c) empaquetado = 00000011 00000001 y desempaquetado 00000011 00000000 00000001 (d) empaquetado = 00010000 00000000 y desempaquetado 00000001 00000000 00000000 00000000
78. (a) 89 (b) 9 (c) 32 (d) 1
80. (a) +3.5 (b) -1.0 (c) +12.5

CAPÍTULO 2

2. 16.
4. EBX.

6. Guarda la dirección de desplazamiento del siguiente paso en el programa.
8. No. Si suma +1 y -1 el resultado será cero, que es un número válido.
10. La bandera I.
12. El registro de segmento direcciona la dirección más baja en un segmento de memoria de 64 K.
14. (a) 12000H (b) 21000H (c) 24A00H (d) 25000H (e) 3F12DH.
16. DI.
18. SS más SP o ESP.
20. (a) 12000H (b) 21002H (c) 26200H (d) A1000H (e) 2CA00H.
22. A todos los 16M bytes.
24. El registro de segmento es un selector que selecciona el descriptor de una tabla de descriptores. También establece el nivel de privilegios de la petición y selecciona la tabla global o la tabla local.
26. A0000H-A0100H.
28. 0028000H-00290FFFH.
30. 3.
32. 64 K.
34.

0000 0011	1101 0000
1001 0010	0000 0000
0000 0000	0000 0000
0010 1111	1111 1111
36. A través de un descriptor almacenado en la tabla global.
38. Los registros invisibles para el programa son las porciones de los registros de segmento correspondientes a la caché y también los registros GDTR, LDTR e IDTR.
40. 4 K.
42. 1024.
44. La entrada cero o la primera entrada.
46. La TLB coloca en la caché los accesos más recientes a memoria por medio del mecanismo de paginación.
20. Los símbolos [] indican direccionamiento indirecto.
22. No se permiten las transferencias de datos de memoria a memoria.
24. MOV WORD PTR DATOS1, 5.
26. La instrucción MOV BX,DATOS copia la palabra de la posición de memoria DATOS1 en el registro BX, mientras que la instrucción MOV BX,OFFSET DATOS copia la dirección de desplazamiento DATOS en BX.
28. No hay error en la instrucción; sólo utiliza un estilo de direccionamiento alternativo.
30. (a) 11750H (b) 11950H (c) 11700H.
32. BP o como una versión extendida EBP.
34. CAMPOS STRUC
 F1 DW ?
 F2 DW ?
 F3 DW ?
 F4 DW ?
 F5 DW ?
 CAMPOS ENDS
36. Directo, relativo e indirecto.
38. El salto intersegmentos permite saltos entre segmentos o hacia cualquier posición dentro del sistema de memoria, mientras que el salto intrasegmentos permite un salto hacia cualquier posición dentro del segmento de código actual.
40. 32.
42. Corto.
44. JMP BX.
46. 2.
48. AX, CX, DX, BX, SP, BP, DI y SI en el mismo orden en el que se listan.
50. PUSHFD.

CAPÍTULO 4

2. AL, AH, BL, BH, CL, CH, DL y DH.
4. EAX, EBC, ECX, EDX, ESP, EBP, EDI y ESI.
6. No se pueden especificar tamaños de registros mixtos.
8. (a) MOV EDX,EBX (b) MOV CL,BL (c) MOV BX,SI (d) MOV AX,DS (e) MOV AH,AL.
10. #.
12. .CODE.
14. Código de operación.
16. La instrucción .EXIT devuelve el control al DOS, al final de un programa.
18. La directiva .Startup indica el inicio de un programa y carga el registro DS con la ubicación del segmento de datos, así como los registros SS y SP con la ubicación de la pila.
2. El bit D indica la dirección de flujo para los datos (REG hacia R/M o R/M hacia REG) y el bit W indica el tamaño de los datos (byte o palabra/doble palabra).
4. DI.
6. DS:[BX+DI].
8. MOV AL,[BX].
10. 8B 77 02.
12. La instrucción se ensamblará, pero si sólo modifica el valor del segmento de código sin modificar el valor del apuntador de instrucciones, el resultado producirá problemas en la mayoría de los casos.
14. 16.
16. Todos los registros de 16 bits se mueven a la pila con la instrucción PUSH.
18. (a) AX se copia a la pila. (b) Se recupera un número de 32 bits de la pila y se coloca en ESI. (c) El contenido tipo palabra de la posición de memoria del segmento de datos direccionada por BX se mete en la pila. (d) EFLAGS se mete en la pila. (e) Se recupera una palabra de la pila y se coloca en DS. (f) Se mete un número 4 de 32 bits en la pila.

20. Los bits 24-31 de EAX se almacenan en la posición 020FFH, los bits 16-23 de EAX se almacenan en la posición 020FEH, los bits 8-15 de EAX se almacenan en la posición 020FDH y los bits 0-7 de EAX se almacenan en la posición 020FCH. Después SP se decremente 4 veces para obtener un valor de 00FCH.
22. Hay muchas ubicaciones posibles, y SP = 0200H y SS = 0200H es una de ellas.
24. Ambas instrucciones cargan la dirección de NUMERO en DI. La diferencia es que MOV DI,OFFSET NUMERO se ensambla como un movimiento inmediato y LEA DI,NUMERO se ensambla como una instrucción LEA.
26. La instrucción LDS BX,NUMERO carga BX con la palabra que se almacena en la posición de memoria del segmento de datos NUMERO, y DS se carga a partir de la posición de memoria del segmento de datos direccionalada por NUMERO+2.
28.

```
MOV BX, NUMERO
MOV DX, BX
MOV SI, BX
```
30. CLD borra la bandera de dirección y STD la activa.
32. La instrucción LODSB copia en el registro AL un byte de datos de la posición de memoria del segmento de datos direccionalada por SI y después incrementa SI en uno, si la bandera de dirección es cero.
34. La instrucción OUTSB envía el contenido de la posición de memoria del segmento de datos direccionalada por SI al puerto de E/S direccionalado por DX; después SI se incrementa en uno, si la bandera de dirección es cero.
36.

```
MOV SI,OFFSET ORIGEN
MOV DI,OFFSET DEST
MOV CX,12
REP MOVSB
```
38. XCHG EBX,ESI.
40. La instrucción XLAT suma el contenido de AX a BX para formar una dirección de memoria del segmento de datos cuyo contenido se copia en AL.
42. El contenido del puerto de E/S 12H se copia en AL.
44. Un prefijo de sustitución de segmento es una instrucción de un byte que se agrega antes de casi cualquier instrucción, para modificar el segmento predeterminado que direcciona esa instrucción.
46.

```
XCHG AX,BX
XCHG ECX,EDX
XCHG SI,DI
```
48. Una directiva de lenguaje ensamblador es un comando especial para el ensamblador, que puede o no generar código o datos para la memoria.
50. LISTA1 DB 30 DUP(?).
52. La directiva .686 informa al ensamblador que debe usar el conjunto de instrucciones para los microprocesadores del Pentium Pro al Pentium 4.
54. Modelos.
56. El programa sale al DOS.
58. La instrucción USES permite al programador meter y sacar registros de manera automática en los procedimientos.

```
60. ALMACENA    PROC    NEAR
      MOV     [DI],AL
      MOV     [DI+1],AL
      MOV     [DI+2],AL
      MOV     [DI+3],AL
      RET
ALMACENA    ENDP
```

CAPÍTULO 5

2. No pueden usar registros de tamaños mixtos.
4. AX = 3100H, C = 0, A = 1, S = 0, Z = 0 y O = 0
6.

```
ADD AX,BX
ADD AX,CX
ADD AX,DX
ADD AX,SP
ADD DI,AX
```
8. ADC DX, BX.
10. El ensamblador no puede determinar el tamaño de la posición de memoria.
12. BH = 7FH, C = 1, A = ?, S = 0, Z = 0 y O = 0 (A no está definida).
14. DEC EBX.
16. La única diferencia entre SUB y CMP es que con CMP se pierde la respuesta.
18. DX (más significativa) y AX (menos significativa).
20. EDX-EAX.
22.

```
MOV DL,5
MOV AL,DL
MUL DL
MUL DL
```
24. AL.
26. División entre cero y desbordamiento de división.
28. AH.
30. DAA y DAS.
32. Divide AL entre 10. Esto hace que los números entre el 0 y el 99 decimal se conviertan en BCD desempaquetado en AH (cociente) y AL (residuo).
34.

```
XCHG AX,BX
ADD AL,DL
DAA
MOV DL,AL
XCHG AH,AL
ADD AL,DH
DAA
MOV DH,AL
XCHG AX,BX
ADD AL,CL
DAA
MOV CL,AL
XCHG AL,AH
ADD AL,AH
DAA
MOV CH,AL
```
36.

```
MOV BH,DH
AND BH,0001111B
```
38.

```
MOV SI,DI
OR SI,1FH
```
40.

```
OR AX,0FH
AND AX,1FFFH
XOR AX,0380H
```
42. TEST CH,4.
44. (a) SHR DI,3 (b) SHL AL,1 (c) ROL AL,3 (d) RCR EDX,1 (e) SAR DH,1.
46. Extra.

48. La instrucción SCASB se repite mientras que la condición sea igual, siempre y cuando CX no sea cero.
50. CMPSB compara el contenido tipo byte del byte en el segmento de datos direccionado por SI con el byte en el segmento extra direccionado por DI.
51. En DOS se muestra la letra C.

CAPÍTULO 6

2. Una instrucción JMP cercana.
4. Un salto lejano.
6. (a) cercano (b) corto (c) lejano.
8. El registro IP o EIP.
10. La instrucción JMP AX salta hacia la dirección de desplazamiento que se almacena en AX. Esto sólo puede ser un salto cercano.
12. La instrucción JMP[DI] salta hacia la posición de memoria direccionada por la dirección de desplazamiento almacenada en la posición de memoria del segmento de datos direccionada por DI. La instrucción JMP FAR PTR[DI] salta hacia la nueva dirección de desplazamiento almacenada en la posición de memoria del segmento de datos direccinada por Di y el nuevo segmento direccinado por la dirección de la posición de memoria del segmento de datos mediante DI+2. JMP [DI] es un salto cercano y JMP FAR PTR[DI] es un salto lejano.
14. JA evalúa la condición de una instrucción aritmética o lógica para determinar si el resultado está por encima. Si es así, se produce un salto; en caso contrario, no se produce ningún salto.
16. JNE, JE, JG, JGE, JL o JLE.
18. JA y JBE.
20. SETZ o SETE.
22. ECX.

```
24. MOV DI,OFFSET DATOSZ
    MOV CX,150H
    CLD
    MOV AL,00H
L1: STOSB
    LOOP L1
```

```
26. CMP AL,3
    JNE @C0001
    ADD AL,2
@C0001:
```

```
28. MOV SI,OFFSET BLOQUEA
    MOV DI,OFFSET BLOQUEB
    CLD
    .REPEAT
        LODSB
        STOSB
    .UNTIL AL == 0
```

```
30. MOV AL,0
    MOV SI,OFFSET BLOQUEA
    MOV DI,OFFSET BLOQUEB
    CLD
    .WHILE AL != 12H
        LODSB
        ADD AL,[DI]
```

```
MOV [DI],AL
INC DI
.ENDW
```

32. La instrucción CALL lejana mete a IP y CS en la pila. A continuación, los dos bytes que van después del código de operación se mueven hacia IP, los dos bytes que siguen se mueven hacia CS y se produce el salto.
34. RET.
36. Mediante el uso de NEAR o FAR a la derecha de la directiva PROC.
38. CUBO PROC NEAR USES AX DX
 MOV AX,CX
 MUL CX
 MUL CX
 RET
CUBE ENDP
40. SUMAS PROC NEAR
 MOV EDI,0
 ADD EAX,EBX
 ADD EAX,ECX
 ADD EAX,EDX
 ADC EDI,0
 RET
SUMAS ENDP
42. De INT 0 hasta INT 255.
44. El vector de interrupción se utiliza para detectar y responder a los errores de división.
46. La instrucción RET saca la dirección de retorno de la pila, mientras que IRET saca las banderas y la dirección de retorno de la pila.
48. Cuando la bandera de desbordamiento es 1.
50. CLI y STI.
52. Si el valor en el registro o la posición de memoria bajo prueba en el operando de destino se encuentra por debajo o por encima de los límites almacenados en la dirección de memoria por el operando de origen.
54. BP.

CAPÍTULO 7

2. No, los bytes deben definirse en C++ mediante el uso de char o de _int8.
4. EAX, EBX, ECX, EDX Y ES.
6. La pila del coprocesador de uso flotante.
8. Accede a los datos en el arreglo cadena1 mediante el uso del registro SI para indizar el elemento tipo cadena.
10. Si no se utilizan encabezados para un programa en C++, será mucho más pequeño.
12. No. INT 21H es una llamada de DOS de 16 bits que no puede usarse en el entorno de 32 bits de Windows.
14.

```
#include "stdafx.h"
#include <conio.h>

int _tmain(int argc, _TCHAR* argv[])
{
    char a = 0;
    while (a != '@')
    {
        a = _getche();
        _putch(a);
    }
    return 0;
}
```

16. La instrucción `_putch(10)` muestra la función de nueva línea y `_putch(13)` regresa el cursor al margen izquierdo de la pantalla.
18. Los módulos de ensamblador separados son los más flexibles.
20. El modelo plano debe usarse con el prototipo de C como en `.MODEL FLAT,C`, en tanto que la función que se enlaza a C++ debe hacerse pública.
22. Una palabra de 16 bit se define mediante la directiva `short o _int16`.
24. Ejemplos de eventos son: movimiento del ratón, pulsación de una tecla, etc. Los manejadores de eventos atrapan estos eventos para poder usarlos en un programa.
26. Sí. El editor de C++ puede usarse para editar un módulo en lenguaje ensamblador, pero el módulo debe usar la extensión `.TXT` en vez de `.ASM`.
28. `#define RDTSC _asm _emit 0x0f _asm _emit 0x31`
30. ;
; Función externa que desplaza en forma
cíclica un byte, 3 posiciones
; a la izquierda
;
.586 ; selecciona el Pentium y
; el modo de 32 bits
.model flat, C ; selecciona el modelo
plano con enlace de C/C++
.stack 1024 ; asigna espacio a la pila
.code ; inicio del segmento de
código

public DespCiclicoIzq3 ; define a
DespCiclicoIzq3 ; como una función
pública
DespCiclicoIzq3 proc ; define el
; procedimiento
DespCiclicoDatos:byte ; define un byte

 mov al,DespCiclicoDatos
 rol al,3
 ret
DespCiclicoIzq3 endp
32. ; Función que convierte
;
.model flat,c
.stack 1024
.code

Public Superior
Superior proc
Char:byte

 mov al,Char
 .if al >= 'a' && a; <= 'z'
 sub al,30h
 .endif
 Ret
Superior endp
34. Las propiedades contienen información sobre un objeto tal como los colores de primer y segundo planos, etc.
36. `_asm inc ptr;`
4. PUBLIC indica que hay una etiqueta disponible para otros módulos del programa.
6. EXTRN.
8. MACRO y ENDM.
10. Los parámetros se pasan a una macro a través de una lista de parámetros que va después de la palabra clave MACRO (en la misma línea).
12. La directiva LOCAL define etiquetas locales y debe estar en la línea que va inmediatamente después de la línea MACRO.
14. SUMAM MACRO LISTA, LONGITUD
PUSH CX
PUSH SI
MOV CX, LONGITUD
MOV SI, LISTA
MOV AX, 0
.REPEAT
 ADD AX, [SI]
 INC SI
.UNTILCXZ
POP SI
POP CX
ENDM
16. BOOL CssssDlg::PreTranslateMessage(MSG* pMsg)
{
 if (pMsg->message == WM_CHAR)
 {
 unsigned int tecla = pMsg->wParam;
 if (tecla < '0' || tecla > '9')
 return true;
 pMsg->wParam = tecla;
 }
 return CDialog::PreTranslateMessage(pMsg);
}
18. BOOL CssssDlg::PreTranslateMessage(MSG* pMsg)
{
 NumeroAleatorio++; //una
 variable
 global
 if (NumeroAleatorio == 63)
 NumeroAleatorio = 9;
 return CDialog::PreTranslateMessage(pMsg);
}
20. void CshiftrotateDlg::OnBnClickedButton1()
{
 Etiquetal.put_Caption("Despl Izq = ");
 shift = true;
 datos1 = 1;
 Etiqueta2.put_Caption("00000001");
 SetTimer(1,500,0);
}

void CshiftrotateDlg::OnBnClickedButton2()
{
 Etiquetal.put_Caption("Despl Ciclico
Izq = ");
 shift = false;
 datos1 = 1;
 Etiqueta2.put_Caption("00000001");
 SetTimer(1,500,0);
}

void CshiftrotateDlg::OnBnClickedButton3()
//nuevo botón Izquierda/
Derecha
{

CAPÍTULO 8

2. Cuando se ensambla el archivo PRUEBA.ASM, genera el archivo PRUEBA.OBJ y el archivo PRUEBA.EXE si no aparecen modificadores en la línea de comandos.

```

direccion = !direccion;           //nueva
                                variable
                                bool
if ( direccion )
{
    if ( shift )
        Etiqueta1.put_
Caption("Despl Izq = ");
else
    Etiqueta1.put_Caption
("Despl Ciclico Izq = ");
}
else
{
    if ( shift )
        Etiqueta1.put_Caption
("Despl Der = ");
else
    Etiqueta1.put_Caption
("Despl Ciclico Izq = ");
}

void CshiftrotateDlg::OnTimer(UINT nIDEvent)
{
    if ( nIDEvent == 1 )
    {
        CString temp = "";
        char temp1 = datos1;
        if ( shift )
            if ( direccion )
                _asm shl temp1,1;
            else
                _asm shr temp1,1;
        else
            if ( direccion )
                _asm rol temp1,1;
            else
                _asm ror temp1,1;
        datos1 = temp1;
        for (int a = 128; a > 0; a>>=1)
        {
            if ( ( temp1 & a ) == a )
                temp += "1";
            else
                temp += "0";
        }
        Etiqueta2.put_Caption(temp);
    }
    CDialog::OnTimer(nIDEvent);
}

```

22. Se inserta un manejador para el mensaje WM_RBUTTONDOWN en el programa para interceptar el botón derecho del ratón en el programa.

24. void CordBurbujaDlg::OnLButtonDown(UINT nFlags, CPoint point)

```

    {
        if (nFlags == 3)
        {
            //izquierda - después derecha
        }
        CDialog::OnLButtonDown(nFlags, point);
    }

void CordBurbujaDlg::OnRButtonDown(UINT nFlags, CPoint point)
{
    if (nFlags == MK_RBUTTON | MK_LBUTTON)
    {
        //derecha - después izquierda
    }
    CDialog::OnRButtonDown(nFlags, point);
}

```

26. El color del sombreado en el borde de los botones de comando.

28. Para convertir un número grande se divide varias veces entre el número 10. Despues de cada dígito, el residuo se almacena como un dígito significativo del resultado BCD.
30. 30H.
32. int ObtieneOct(void)
- ```

{
 CString temp;
 int resultado = 0;
 char temp1;
 GetDlgItemText(IDC_EDIT1, temp);
 for (int a = 0; a < temp.GetLength(); a++)
 {
 temp1 = temp.GetAt(a);
 _asm
 {
 shl resultado,3
 mov eax,0
 mov al,temp1
 sub al,30h
 or resultado,eax
 }
 }
 return resultado;
}

34. char Arriba(char temp)
{
 if (temp >= 'a' && temp <= 'z')
 _asm sub temp,20h;
 Return temp;
}

36. El sector de arranque es en donde se encuentra un programa con la secuencia inicial de instrucciones de arranque, que es el que carga el sistema operativo. La FAT es una tabla que contiene números que indican si un clúster está libre, defectuoso u ocupado. Si está ocupado, un FFFFH indica el final de una cadena de archivos o el siguiente número de clúster en la cadena. El director almacena información sobre un archivo o carpeta.

38. Sectores.

40. Un clúster es un agrupamiento de sectores.

42. 4 Gbytes.

44. 8.

46. 256.

48. CFile::CambiaNombre("PRUEBA.LST", "PRUEBA.LIS");

50. Un control ActiveX es un objeto común que puede utilizarse en un lenguaje de programación visual.

52. En la figura D-1 verá la salida (el control predefinido ListBox contiene la salida).


```

//código que se coloca en la función OnInitDlg
int valtemp = 1;
for ( int a = 0; a < 8; a++ )
{
    CString temp = "2^ = ";
    temp.SetAt(2, a + 0x30);
    temp += ObtieneNum(valtemp);
    Lista.InsertString(a, temp);
    Valtemp <= 1;
}

String CPotenciasDlg::ObtieneNum(int temp)
{
    char numero[10]

```


```

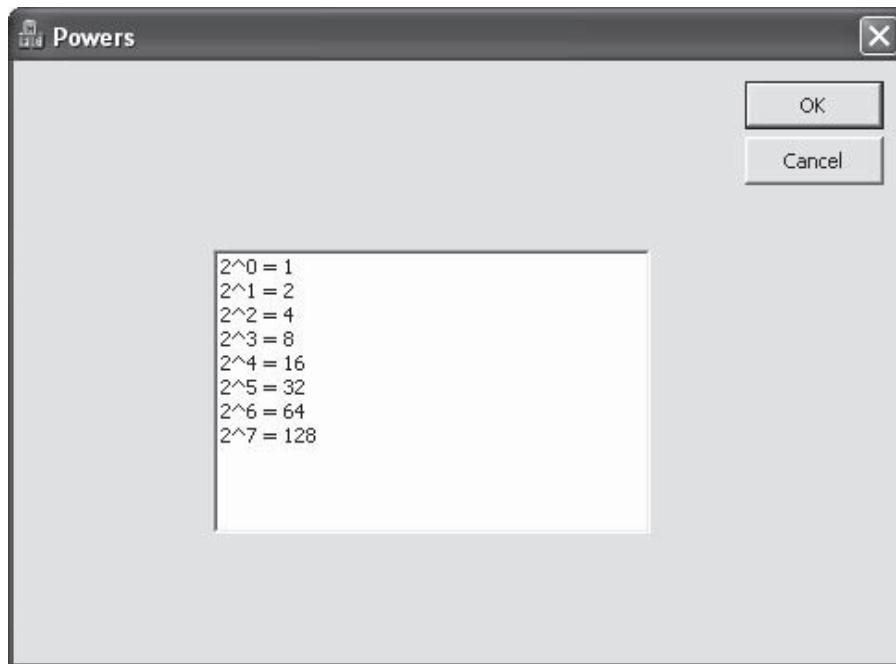


FIGURA D-1

```

 _asm
 {
 mov eax,temp
 mov ebx,10
 push ebx
 mov ecx,0
ciclol:
 mov edx,0
 div ebx
 push edx
 cmp eax,0
 jnz ciclol
ciclo2:
 pop edx
 cmp edx,ebx
 je ciclo3
 add dl,30h
 mov numero[ecx],dl
 inc ecx
 jmp ciclo2
ciclo3:
 mov byte ptr numero[ecx],0
 }
 return numero;
}

54. BOOL CPantallaDlg::PreTranslateMessage(MSG*
pMsg)
{
 char busca[] = {0x3f, 6, 0x5b, 0x4f,
 0x66, 0x6d, 0x7d, 7, 0x7f, 0x6f 0x77,
 0x7c, 0x39, 0x5e, 0x79, 0x71};
 char temp;
 if (pMsg->message == WM_KEYDOWN)
 {
 if (pMsg->wParam >= '0' &&
pMsg->wParam <= '9'
 || pMsg->wParam >= 'A'
 && pMsg->wParam <= 'F'
 || pMsg->wParam >= 'a' &&
pMsg->wParam <= 'f')
 {
 temp = pMsg->wParam -
0x30;
 if (temp > 9)
 temp -= 7;
 _asm //busca el código
de 7 segmentos
 {
 lea ebx,busca
 mov al,temp
 xlat
 mov temp,al
 }
 MuestraDigito(temp);
 //muestra el dígito
 }
 return true;
 }
 //termina con la pulsación de tecla
}
return CDialog::PreTranslateMessage(pMs
g);
}

```

## CAPÍTULO 9

2. Sí y no. El control de corriente de un cero lógico se reduce a 2.0 mA y la inmunidad al ruido se reduce a 350 mV.
4. Los bits de dirección A<sub>0</sub>-A<sub>7</sub>.
6. Una operación de lectura.
8. El ciclo de trabajo debe ser del 33%.
10. Que se está llevando a cabo una operación de escritura.
12. Que el bus de datos está enviando datos a la memoria o a un dispositivo de E/S.
14. IO/M, DT/R y SSO.

16. Son señales para el coprocesador que indican lo que está haciendo la cola del microprocesador.
18. 3.
20.  $14 \text{ MHz}/6 = 2.33 \text{ MHz}$ .
22. Las conexiones  $A_0-A_{15}$  del bus de direcciones
24. El enclavamiento transparente 74LS373.
26. Si se conectan demasiados dispositivos de memoria y de E/S en un sistema, los buses deben utilizar búferes.
28. 4.
30. Obtención y ejecución.
32. (a) Se envía la dirección junto con ALE. (b) Se permite tiempo para el acceso a memoria y la entrada READY se muestrea. (c) Se emite la señal de lectura o escritura. (d) Se transfieren datos y se desactiva la lectura o la escritura. (e) El estado de espera permite un tiempo adicional para el acceso a memoria.
36. Selecciona una o dos etapas de sincronización para READY.
38. La operación en modo mínimo se utiliza con más frecuencia en aplicaciones integradas y la operación en modo máximo se utilizaba con más frecuencia en las primeras computadoras personales.
10. Memoria estática de acceso aleatorio.
12. 250 ns.
14. Las entradas de dirección para muchas DRAMs se multiplexan, de manera que una entrada de dirección acepte dos bits de dirección distintos, con lo cual se reduce el número de terminales requeridas para direccionar memoria en una DRAM.
16. Por lo general, la cantidad de tiempo es igual a un ciclo de lectura y sólo representa una pequeña cantidad de tiempo en un sistema de memoria moderno.
18. Vea la figura D-2.
20. Una de las ocho salidas se vuelve un cero lógico, según lo indicado por las entradas de dirección.
22. Vea la figura D-3.
24. Lenguaje Verilog de descripción de hardware.
26. El bloque de arquitectura entre begin y end.
28.  $\overline{\text{MRDC}}$  y  $\overline{\text{MWTC}}$ .
30. Vea la figura D-4.
32. 5.
34. 1.
36.  $\overline{\text{BHE}}$  selecciona el banco de memoria superior y  $A_0$  selecciona el banco de memoria inferior.
38. Decodificadores separados y señales de escritura separadas.
40. Banco de memoria inferior.
42. library ieee;  
use ieee.std\_logic\_1164.all;  
  
entity DECODIFICADOR\_10\_28 is  
  
port (  
A23, A22, A21, A20, A19, A18, A17,  
A16, A0, BHE, MWTC: in STD\_LOGIC;  
SEL, LWR, HWR: out STD\_LOGIC  
);

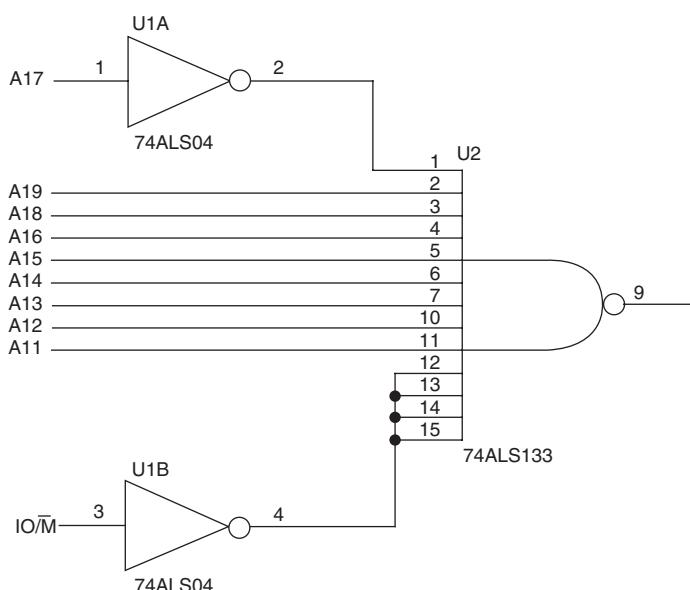


FIGURA D-2

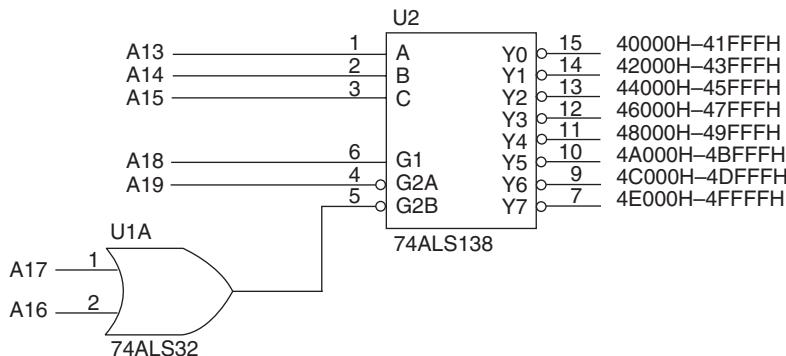


FIGURA D-3

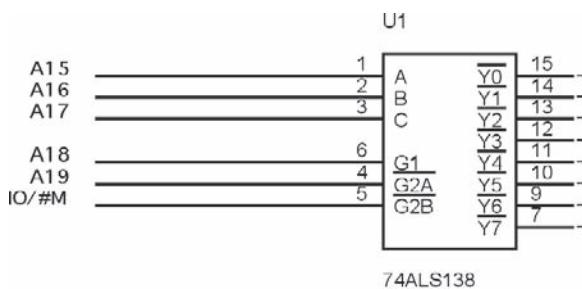


FIGURA D-4

```

end;
architecture V1 of DECODIFICADOR_10_28 is
begin
 SEL <= A23 or A22 or A21 or A20 or A19
 or A18 or (not A17) or (not A16);
 LWR <= A0 or MWTC;
 HWR <= BHE or MWTC;
end V1;

```

44. Vea la figura D-5.  
48. Sí, siempre y cuando no se acceda a una posición de memoria en la DRAM.

## CAPÍTULO 11

2. La dirección de E/S se almacena en el segundo byte de la instrucción.
4. DX.
6. La instrucción OUTSB transfiere el byte del segmento de datos direccionado por SI hacia el puerto de E/S direccionado por DX, después SI se incrementa en 1.
8. La E/S por asignación de memoria utiliza cualquier instrucción que transfiere datos desde o hacia la memoria para E/S, mientras que la E/S aislada requiere el uso de la instrucción IN o OUT.
10. La interfaz básica de salida es un enclavamiento que captura los datos de salida y los retiene para el dispositivo de salida.

12. Inferior.
14. 4.
16. Elimina los rebotes mecánicos de un interruptor.
18. Vea la figura D-6.
20. Vea la figura D-7.
22. Vea la figura D-8.
24. Si el puerto es de 16 bits, no hay necesidad de habilitar la mitad inferior ni la superior.
26. D<sub>47</sub>-D<sub>40</sub>.
28. El grupo A es el puerto A y PC<sub>4</sub>-PC<sub>7</sub>, mientras que el grupo B es el puerto B y PC<sub>3</sub>-PC<sub>0</sub>.
30. RD.
32. Entradas.
34. La entrada estroboscópica enclava los datos de entrada y activa la bandera completa del búfer y la petición de interrupción.
36. RETRASO PROC NEAR USES ECX
   
MOV ECX, 7272727
   
D1: LOOPD D1
   
RET
   
RETRASO ENDP
38. La señal estroboscópica (STB).
40. La terminal INTR se habilita mediante la activación del bit INTE en PC<sub>4</sub> (puerto A) o PC<sub>2</sub> (puerto B).
42. Cuando se envían datos al puerto, OBF se vuelve un 0 y cuando se envía ACK al puerto, OBF se vuelve un 1.
44. El grupo o puerto A contiene los datos bidireccionales.
46. Se envía el comando 01H a la pantalla LCD.
48. ;Muestra la cadena con terminador nulo direccional por DS:BX  
;utiliza una macro llamada ENVIA para enviar datos a la pantalla
 ;
 MUESTRA PROC NEAR USES BX
 SEND 86H,2,1 ;mueve el cursor
 a la posición 6
 .WHILE BYTE PTR [BX] != 0
 SEND [BX],0,1
 INC BX
 .ENDW
 RET
 MUESTRA ENDP

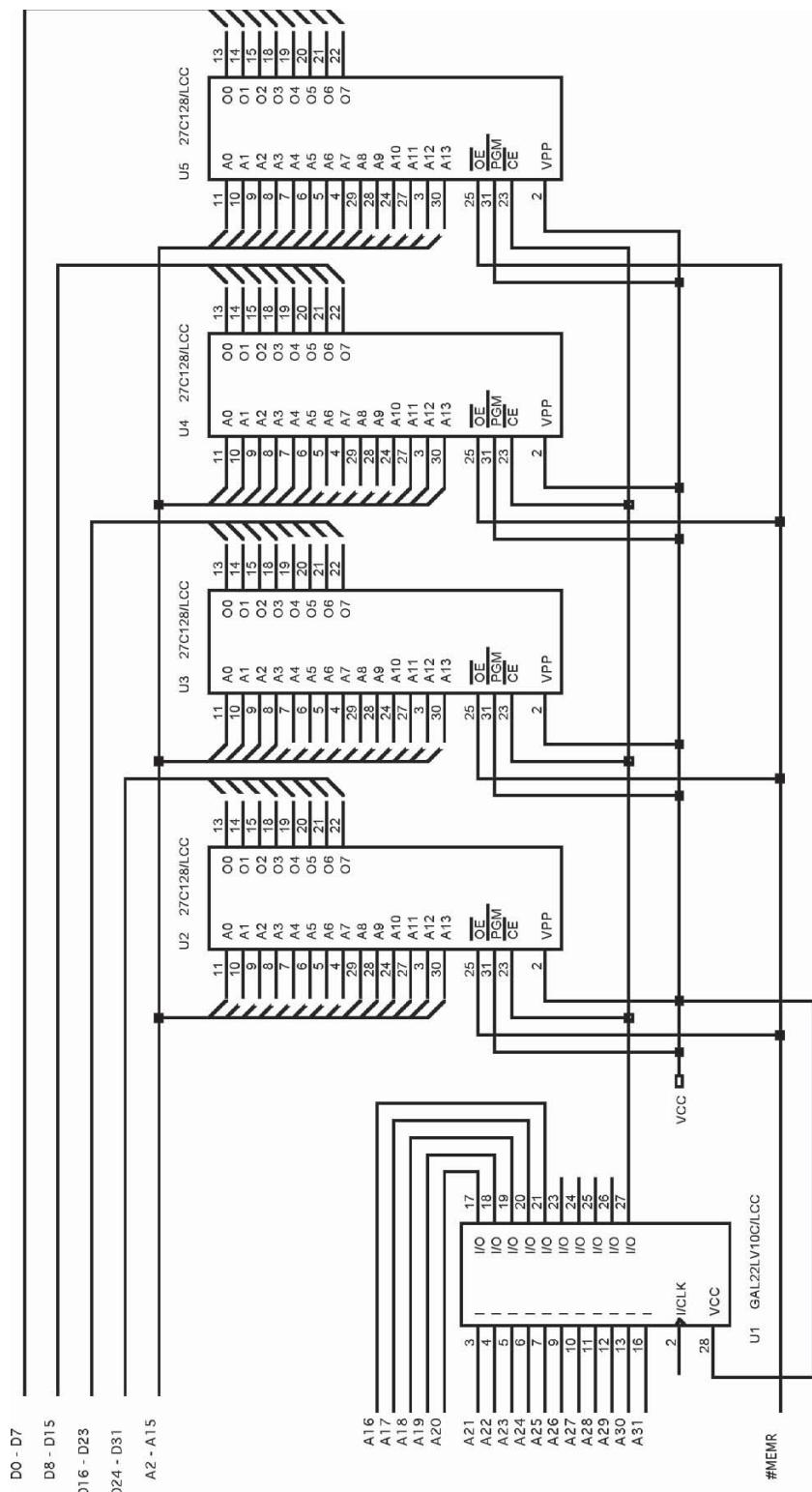


FIGURA D-5

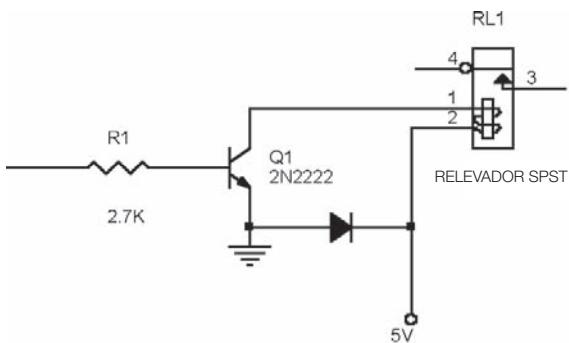


FIGURA D-6

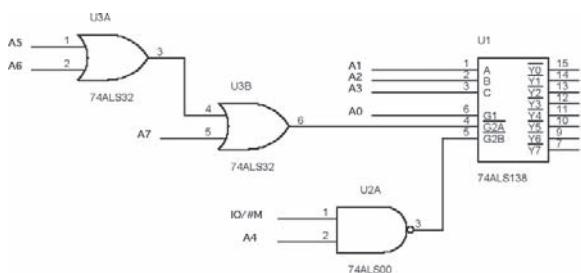


FIGURA D-7

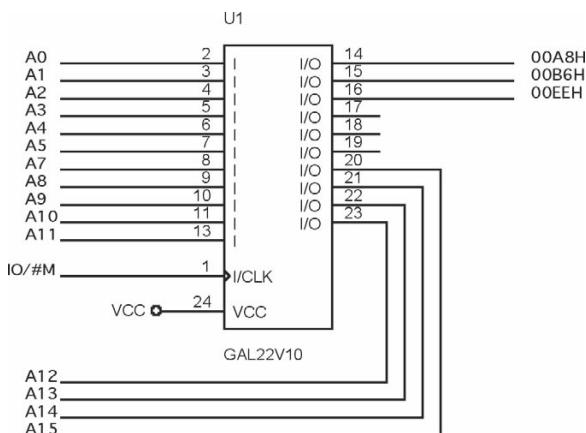


FIGURA D-8

50. Las únicas modificaciones requeridas son que en vez de cuatro filas hay tres filas y tres resistencias elevadoras conectadas al puerto A, así como cinco columnas para conectar al puerto B. Desde luego que el software también requiere de pequeñas modificaciones.

54. 6.

58. El menos significativo.

62. Datos que se envían un bit a la vez sin ningún pulso de reloj.

64. LINEA EQU 023H  
LSB EQU 020H  
MSB EQU 021H  
FIFO EQU 022H

```

MOV AL,10001010B ;habilita divisor
OUT LINEA,AL de Baudios
MOV AL,60 ;programa
OUT LSB,AL velocidad de
MOV AL,0 MSB,AL Baudios
MOV AL,00011001B ;programa 7 bits
OUT LINEA,AL de datos, paridad
;par, un bit de
MOV AL,00000111B ;habilita
OUT FIFO,AL transmisor y
;y receptor

```

66. Simplex = recibir o enviar datos; half-duplex = recibir y enviar datos; pero sólo en una dirección a la vez; y full-duplex = recibir y enviar datos al mismo tiempo.

68. ENVIAS PROC NEAR

```

MOV CX,16
.REPEAT
 IN AL,LSTAT ;obtiene
 .REPEAT registro
 TEST AL,20H de estado de
 ;prueba
 ;bit TH
 .UNTIL !ZERO?
 LODSB ;carga
 OUT DATOS,AL datos
 .UNTILCXZ ;transmite
 .REPEAT
 RET
 DATOS,AL
 ENVIAS ENDP

```

70. 0.01 V.

72.

```

.MODEL TINY
.CODE
.STARTUP
MOV DX,400H
.WHILE 1
MOV CX,256
MOV AL,0
.REPEAT
 OUT DX,AL
 INC AL
 CALL RETRASO
.UNTILCXZ
MOV CX,256
.REPEAT
 OUT DX,AL
 DEC AL
 CALL RETRASO
.UNTILCXZ
.ENDW
RETRASO PROC NEAR

```

;retraso de tiempo de 39 microsegundos  
RETRASO ENDP  
END

74. INTR indica que el convertidor ha completado una conversión.

76. Vea la figura D-9.

## CAPÍTULO 12

2. Una interrupción es una llamada a una subrutina iniciada por hardware o por software.

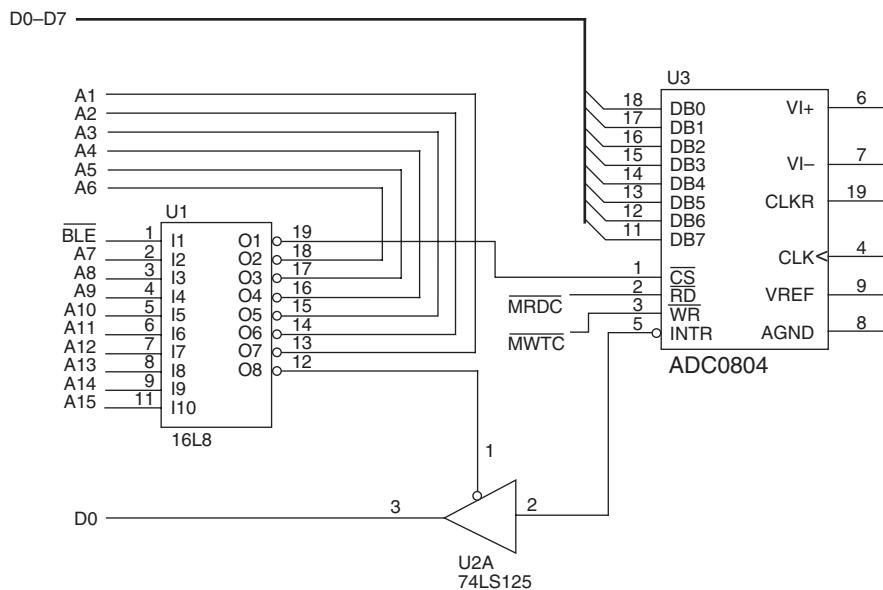


FIGURA D-9

4. Una interrupción sólo utiliza tiempo de la computadora cuando se activa.

6. INT, INT<sub>3</sub>, INTO, CLI y STI.

8. En el primer 1 Kbyte del sistema de memoria en modo real y en cualquier parte en modo protegido.

10. Del 00H al 1FH.

12. En cualquier parte del sistema de memoria.

14. Una interrupción en modo real mete CS, IP y FLAGS en la pila, mientras que una interrupción en modo protegido mete CS, EIP y EFLAGS en la pila.

16. La instrucción INTO se produce si se activa el desbordamiento.

18. La instrucción IRET saca las banderas y la dirección de retorno de la pila.

20. El estado de la estructura de la interrupción se almacena en la pila, de manera que se restaure cuando se produzca el retorno. Se borran las banderas de interrupción y de rastreo.

22. La bandera T controla si está habilitado o deshabilitado el rastreo.

24. La bandera T se habilita o deshabilita mediante la manipulación directa de los bits de bandera, ya que no hay instrucciones para controlarla.

26. No.

28. Sensible al borde y al nivel.

30. Una FIFO es una estructura de memoria tipo “primero en entrar, primero en salir”.

32. Vea la figura D-10

34. Una cadena tipo margarita es cuando se aplica un OR a las señales para generar una sola señal.

36. Un controlador de interrupciones programable.

38. Son las ocho entradas de petición de interrupción.

40. A una entrada de petición de interrupción.

42. Una OCW es una palabra de control de operación.

44. ICW<sub>2</sub>.

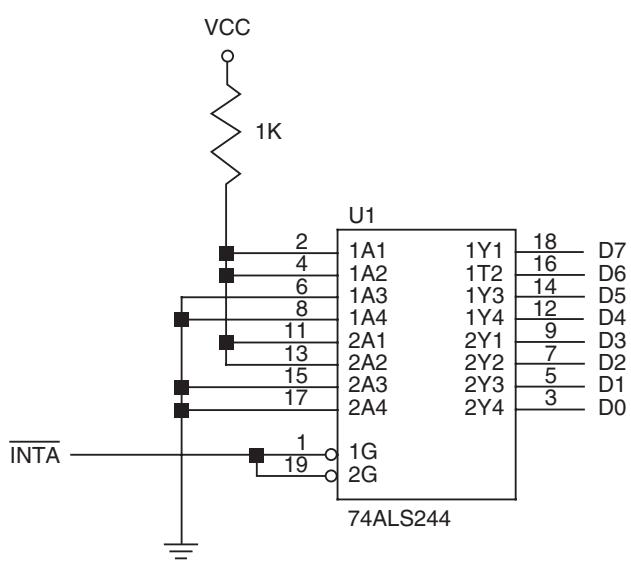
46. Programa la sensibilidad y uno o varios dispositivos 8259.

48. El nivel de petición de interrupción más reciente se convierte en el nivel de interrupción más bajo después de ser atendido.

50. INT 8 hasta INT 0FH.

| Output | Pin | Data Bus Connection |
|--------|-----|---------------------|
| 1Y1    | 1A1 | D7                  |
| 1T2    | 1A2 | D6                  |
| 1Y3    | 1A3 | D5                  |
| 1Y4    | 1A4 | D4                  |
| 2Y1    | 2A1 | D3                  |
| 2Y2    | 2A2 | D2                  |
| 2Y3    | 2A3 | D1                  |
| 2Y4    | 2A4 | D0                  |

74ALS244



**FIGURA D-10**

## CAPÍTULO 13

2. Cuando se coloca un 1 en HOLD, el programa deja de ejecutarse y los buses de direcciones, datos y control entran en su estado de alta impedancia.
4. E/S a memoria.
6. DACK.
8. El microprocesador se encuentra en su estado de retención y el controlador de DMA tiene el control de los buses.
10. 4.
12. El registro de comandos.
16. Una unidad de almacenamiento portátil USB es un dispositivo que actúa como dispositivo de almacenamiento mediante el uso de una memoria Flash.
18. Pistas.
20. Cilindro.
22. Vea la figura D-11.
24. Las cabezas en una unidad de disco duro están diseñadas en forma aerodinámica para transportarse en un cojín de aire a medida que el disco gira, por lo cual se les conoce como cabezas voladoras.
26. El mecanismo de posicionamiento mediante el motor de velocidad gradual es ruidoso y no muy preciso, mientras que el mecanismo de posicionamiento de bobina es silencioso y muy preciso, ya que su posición puede ajustarse en forma continua.
28. Un CD-ROM es un dispositivo óptico para almacenar música o datos digitales y tiene una capacidad aproximada de 660 M o 700 M (80 minutos) bytes.
30. Un monitor TTL utiliza señales TTL para generar una pantalla y un monitor análogo utiliza señales análogas.
32. Cyan, magenta y amarillo.
34. 600 líneas con 800 elementos horizontales.
36. El estilo más reciente de conector de entrada de video digital para todo tipo de equipo de video
38. 16 millones de colores.

## CAPÍTULO 14

2. Palabra (16 bits,  $\pm 32$  K), doble palabra (32 bits,  $\pm 2$  G) y palabra cuádruple (64 bits,  $\pm 9 \times 10^{18}$ ).
4. Precisión simple (32 bits), doble precisión (64 bits) y precisión temporal (128 bits).

6. (a) –7.75 (b) .5625 (c) 76.5 (d) 2.0 (e) 10.0 (f) 0.0
8. El microprocesador continúa ejecutando instrucciones de microprocesador (tipo entero), mientras que el coprocesador ejecuta una instrucción de punto flotante.
10. Copia el registro de estado del coprocesador en AX.
12. Mediante la comparación de los dos registros y después mediante la transferencia de la palabra de estado hacia el registro AX. Si la instrucción SAHF se ejecuta después, puede usarse una instrucción JZ para evaluar el resultado de la instrucción de comparación del coprocesador.
14. FSTSW AX.
16. Los datos siempre se almacenan como números de precisión temporal de 80 bits.
18. 0.
20. Afín permite infinito positivo y negativo, mientras que proyectivo hace la suposición de que el infinito es sin signo.
22. Precisión extendida (temporal).
24. Se copia el contenido de la parte superior de la pila en la posición de memoria DATOS como un número de punto flotante.
26. FADD ST,ST(3).
28. FSUB ST(2),ST.
30. La división directa divide la parte superior de la pila entre el contenido de una posición de memoria y devuelve el cociente a la parte superior de la pila. La división inversa divide la parte superior de la pila entre el contenido de la posición de memoria y devuelve el resultado a la parte superior de la pila. Si no existe un operando, entonces la división directa divide ST(1) entre ST y la división inversa divide ST entre ST(1).
32. Realiza una instrucción MOV hacia ST si la condición es que el valor esté por debajo.
34. RECIP PROC NEAR  
MOV TEMP, EAX  
FLD TEMP  
FLD1  
FDIVR  
FSTP TEMP  
MOV EAX, TEMP  
RECIP ENDP  
TEMP DD ?
36. Encuentra la función  $2^x - 1$ .
38. FLDPI.
40. Indica que el registro ST(2) está libre.
42. El estado del equipo.

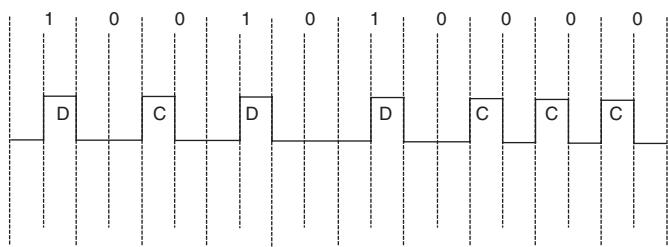


FIGURA D-11

```

44. CAPR PROC NEAR
 FLDPI
 FADD ST, ST(1)
 FMUL F
 FMUL C1
 FLD1
 FDIVR
 FTSP XC
 RET
 CAPR ENDP

```

46. En el software moderno nunca se utiliza.

```

48. TOT PROC NEAR
 FLD R2
 FLD1
 FDIVR
 FLD R3
 FLD1
 FDIVR
 FLD R4
 FLD1
 FDIVR
 FADD
 FADD
 FLD1
 FDIV
 FADD R1
 FSTP RT
 RET
 TOT ENDP

```

```

50. PROD PROC NEAR
 MOV ECX, 100
 .REPEAT
 FLD ARREGLO1[ECX*8-8]
 FMUL ARREGLO2[ECX*8-8]
 FSTP ARREGLO3[ECX+8-8]
 .UNTILCXZ
 RET
 PROD ENDP

```

```

52. POT PROC NEAR
 MOV TEMP, EBX
 FLD TEMP
 F2XM1
 FLD1

```

```

FADD
MOV TEMP, EAX
FLD TEMP
FYL2X
FSTP TEMP
MOV ECX, TEMP
RET
POT ENDP

```

```

54. GANA PROC NEAR
 MOV ECX, 100
 .REPEAT
 FLD DWORD PTR VOUT[ECX*4-4]
 FDIV DWORD PTR VIN[ECX*4-4]
 CALL LOG10
 FIMUL VEINTE
 FSTP DWORD PTR GDB[ECX*4-4]
 .UNTILCXZ
 RET
 VEINTE DW 20
 GANA ENDP

```

56. La instrucción EMMS borra la pila del coprocesador para indicar que la unidad MMX ha terminado de utilizar la pila.

58. La saturación con signo se produce cuando se suman números tipo byte y tienen valores de 7FH para un desbordamiento y 80H para un subdesbordamiento.

60. La instrucción FSAVE almacena todos los registros MMX en memoria.

62. Una sola instrucción, múltiples datos.

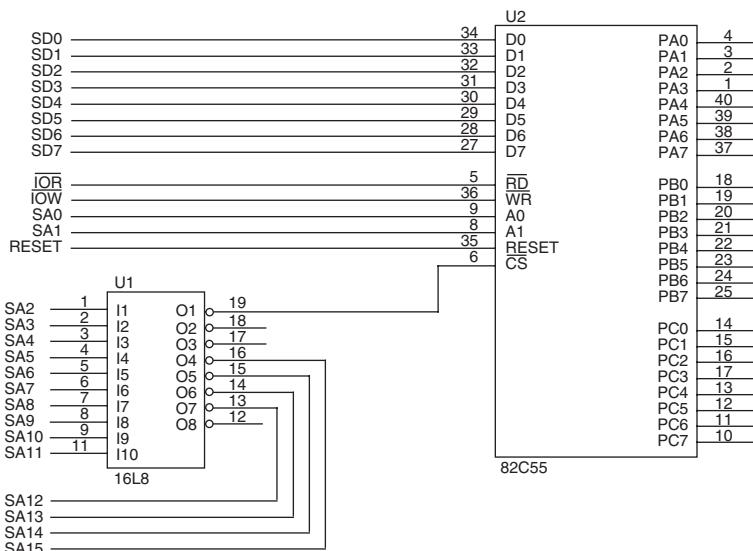
64. 128 bits.

66. 16.

68. Sí.

## CAPÍTULO 15

2. De 8 o de 16 bits, dependiendo de la configuración de zócalo.
4. Vea la figura D-12.



**FIGURA D-12**

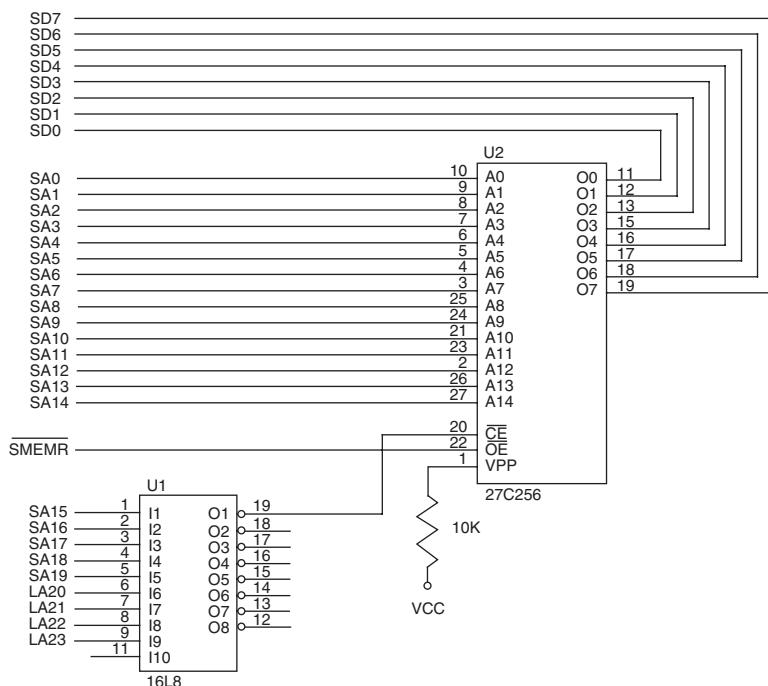


FIGURA D-13

6. Vea la figura D-13.
8. Vea la figura D-14.
12. 16 bits.
14. La memoria de configuración identifica el fabricante y también información sobre las interrupciones.
16. Es la señal de habilitación de comandos/bus que se pone en nivel alto para indicar que el bus PCI contiene un comando, y en nivel bajo para indicar datos.
18. 

```
MOV AX, 0B108H
 MOV BX, 0
 MOV DI, 8
 INT 1AH
```
20. 2.5 GHz.
22. Sí.
24. COM<sub>1</sub>.
28. 1.5 Mbps, 12 Mbps y 480 Mbps
30. 127.
32. 5 metros.
34. Un bit adicional que se envía en el flujo de datos si se envían más de seis unos en una fila.
36. De 1 a 1023 bytes.
38. El bus PCI transfiere datos a 33 MBps, mientras que el bus AGP transfiere datos a 2 GBps (8×).

## CAPÍTULO 16

2. Las mejoras de hardware incluyen temporizadores internos, entradas de interrupción adicionales, lógica de selección de chip, puertos de comunicaciones en serie,

- terminales en paralelo, controlador de DMA y un controlador de interrupciones.
4. 10 MHz.
6. 3 mA.
8. El punto en el que aparece la dirección.
10. 260 ns para la versión de 16 MHz que opera a 10 MHz.
12. 

```
MOV AX, 1000H
 MOV DX, OFFFEH
 OUT DX, AX
```
14. 10 en la mayoría de las versiones del 80186/80188, incluyendo las interrupciones internas.
16. Los registros de control de interrupciones controlan una sola interrupción.
18. El registro de sondeo de interrupciones reconoce la interrupción, mientras que el registro de estado de sondeo de interrupciones no la reconoce.
20. 3.
22. El temporizador 2.
24. Determina si funciona la habilitación del bit de conteo.
26. El bit ALT selecciona ambos registros de comparación, de manera que pueda programarse la duración de los tiempos de salida de 1 lógico y 0 lógico.
28. 

```
MOV AX, 123
 MOV DX, OFF5AH
 OUT DX, AX
 MOV AX, 23
 ADD DX, 2
 OUT DX, AX
 MOV AX, 0C007H
 MOV DX, OFF58H
 OUT DX, AX
```

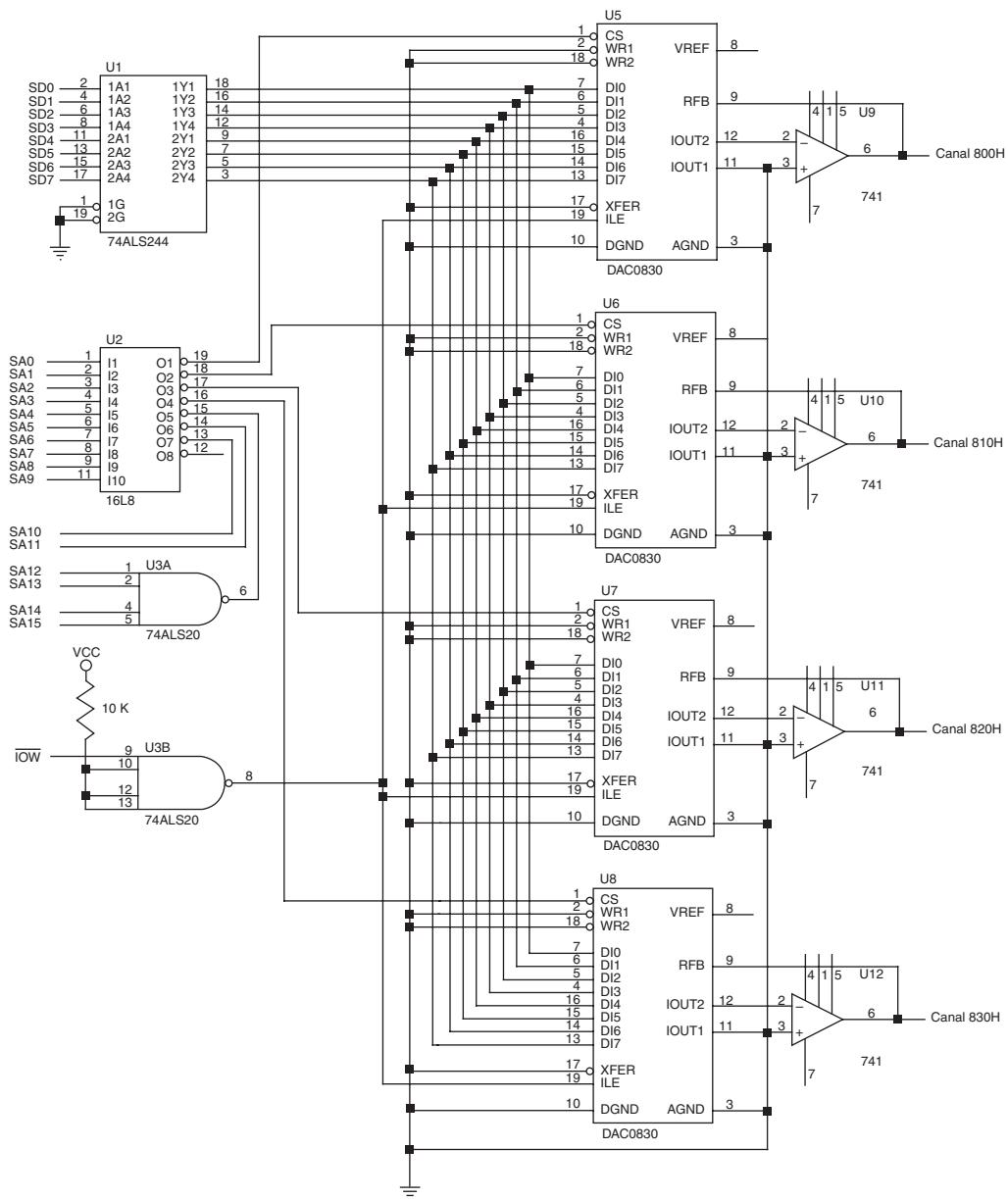


FIGURA D-14

30. 2.
32. Se coloca un 1 lógico en los bits CHG/NOCHG y START/STOP del registro de control.
34. 7.
36. Chip.
38. 15.
40. Determina la operación de las terminales PCS5 y PCS6.
42. 

```
MOV AX,1001H
 MOV DX,0FF90H
 OUT DX,AX
 MOV AX,1048H
 OUT DX,AX
```

44. IG.
46. Verifica el acceso de lectura.
48. Un RTOS es un sistema operativo en tiempo real que tiene un tiempo predecible y garantizado para el acceso de los procesos.

## CAPÍTULO 17

2. 64T.
4. Vea la figura D-15.

6. El sistema de memoria tiene hasta 4 Gbytes y las señales de habilitación de banco seleccionan uno o más de los bancos de memoria de 8 bits.
8. La canalización permite que el microprocesador envíe la dirección de la siguiente posición de memoria mientras obtiene los datos de la operación de memoria anterior. Esto permite a la memoria un tiempo adicional para acceder a los datos.
10. 0000H-FFFFFH.
12. LA E/S tiene la misma dirección que los modelos anteriores del microprocesador. La diferencia es que la E/S se ordena como un espacio de 32 bits, con cuatro bancos de 8 bits que se seleccionan mediante las señales de habilitación de banco.
14. La terminal  $\bar{BS16}$  hace que el microprocesador funcione con un bus de datos de 8 bits.
16. Los primeros cuatro registros de depuración ( $DR_0$ - $DR_3$ ) contienen direcciones de punto de interrupción; los registros  $DR_4$  y  $DR_5$  se reservan para uso de Intel;  $DR_6$  y  $DR_7$  se utilizan para controlar la depuración.
18. Los registros de prueba se utilizan para probar el búfer de traducción adelantada.
20. El bit PE cambia al microprocesador al modo protegido si está activado, y al modo real si está en cero.
22. El direccionamiento de índice escalado utiliza un factor de escala de 1, 2, 4 u 8 veces para escalar el direccionamiento de byte, palabra, doble palabra o palabra cuádruple.
24. (a) La dirección en el segmento de datos en la posición a la que apunta EBX por 8 más ECX; (b) la dirección en el arreglo DATOS del segmento de datos a la que apunta la suma de EAX más EBX; (c) la dirección en la posición DATOS del segmento de datos; (d) la dirección en el segmento de datos al que apunta EBX.
26. Tipo 13 (0DH).
28. La tabla de descriptores de interrupción y sus descriptores de interrupción.
30. Un selector aparece en un registro de segmento y selecciona un descriptor de una tabla de descriptores. También contiene el nivel de privilegio solicitado de la petición.
32. El registro de la tabla de descriptores globales.
34. Como un descriptor direcciona hasta 4 G de memoria y existen 8 K descriptores locales y 8 K globales disponibles a la vez, 4 G por 16 K = 64 T.
36. El TSS almacena los enlaces y registros de una tarea, para que las tareas puedan comutarse con eficiencia.
38. El cambio se produce cuando se coloca un 1 lógico en el bit PE de  $CR_0$ .
40. El modo virtual, que simula el DOS en modo protegido, establece espacios de 1M de memoria que operan en el modo real.
42. 4 K.
44. El 80486 tiene una caché interna de 8 K y también contiene un coprocesador.
46. Los conjuntos de registros son casi idénticos.
48. PCHK y  $DP_0$ - $DP_3$ .
50. 8 K.
52. Una ráfaga es cuando se leen o se escriben cuatro números de 32 bits entre la caché y la memoria.
54. Autoprueba integrada.

## CAPÍTULO 18

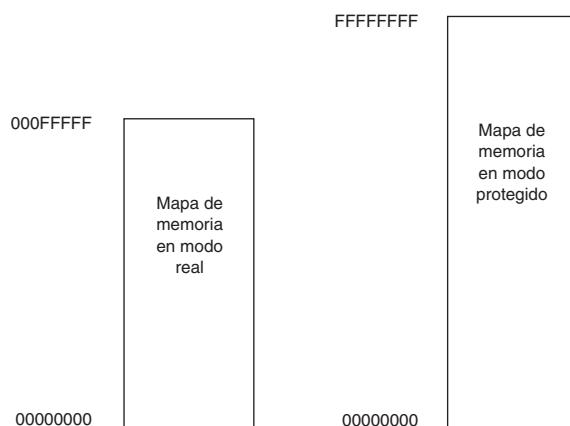


FIGURA D-15

2. 64 Gbytes.
4. Estas terminales generan y comprueban la paridad.
6. La terminal de ráfaga lista se utiliza para insertar un estado de espera en el ciclo del bus.
8. 18.5 ns.
10.  $T_2$ .
12. Una caché de datos de 8 Kbytes y una caché de instrucciones de 8 Kbytes.
14. Sí, si una es instrucción del coprocesador y si las instrucciones de enteros no son dependientes.
16. El modo SSM se utiliza para la administración de energía en la mayoría de los sistemas.
18. 38000H.
20. La instrucción CMPXCH8B compara el número de 64 bits en EDX:EAX con un número de 64 bits almacenado en la memoria. Si son iguales, ECX:EBX se almacena en la memoria. Si no son iguales, el contenido de la memoria se mueve hacia EDX:EAX.
22. ID, VIP, VIF y AC.
24. Para acceder a páginas de 4 M, se quitan las tablas de páginas y sólo se utiliza el directorio de páginas con una dirección de desplazamiento de 22 bits.
26. El Pentium Pro es una versión mejorada del Pentium que contiene tres unidades de enteros, una unidad MMX y un bus de direcciones de 36 bits.

28. 36 bits de dirección en A<sub>3</sub> hasta A<sub>35</sub> (A<sub>0</sub>-A<sub>2</sub> se codifican en las señales de selección de banco).
30. El tiempo de acceso en un Pentium de 66 MHz es de 18.5 ns y en el Pentium Pro a 66 MHz el tiempo de acceso es de 17 ns.
32. Se compra SDRAM de 72 bits para aplicaciones de memoria ECC, en vez de memoria de 64 bits.
16. El registro ECX direcciona el número MSR cuando se ejecuta la instrucción RDMSR. Después de la ejecución, EDX:EAX contiene el contenido del registro.

```
18. PRUEBAS PROC NEAR
 CPUID
 BT EDX,800H
 RET
PRUEBAS ENDP
```

20. EDX al registro EIP y el valor en ECX al registro ESP.
22. Anillo 3.
24. Pentium Pro.
26. El Pentium 4 requiere una fuente de energía con un conector de 12 V adicional para la tarjeta principal. Debe usarse una fuente de energía compatible con el Pentium 4.

```
28. bool Hyper()
{
 __asm
 {
 bool Estado = true;
 mov eax,1
 cpuid
 mov temp1,31h
 bt edx,28 ;verifica si hay
 hyper-threading
 jc Hyper1
 mov Estado,0
 }
 return Estado;
}
```

## CAPÍTULO 19

2. 512 K, 1 M o 2 M.
4. La caché del Pentium Pro está en la tarjeta principal; la caché del Pentium 2 está en el cartucho y opera a una velocidad alta.
6. 64 Gbytes.
8. 242.
10. Las señales de lectura y escritura se desarrollan mediante el conjunto de chips, en vez del microprocesador.
12. 8 ns después de acceder a la primera palabra cuádruple. De todas formas se requieren 60 ns para acceder a la primera palabra cuádruple.
14. Se han agregado registros específicos del modelo para SYSENTER\_CS, STSENTER\_SS y SYSENTER\_ESP.

---

# ÍNDICE

---

- #, 77-78  
, 78  
.BREAK, 195  
.CODE, 78  
.COM, 239  
.CONTINUE, 195  
.DATA, 81  
.ELSE, 194  
.ELSEIF, 194  
.ENDIF, 194  
.ENDW, 195  
.EXE, 139  
.EXIT, 78  
.IF, 192\*193  
.LISTALL, 138, 193  
.LST, 79  
.MODEL, 78, 108, 139  
.REPEAT, 196  
.STACK, 118  
.STARTUP, 78  
.UNTIL, 196  
.UNTILCXZ, 196  
.WHILE, 195  
:, 79  
4004, 5  
4040, 5  
8008, 6  
8080, 6  
8085, 6  
8086, 7  
8088, 7  
80286, 7  
80386, 7  
80486, 8  
\_asm, 80, 214, 216  
\_emit, 235
- A**  
AAA, 164  
AAD, 164  
AAM, 164, 261  
AAS, 166  
Ábaco, 2  
Acarreo auxiliar, 53
- Acarreo, 52, 153, 154, 156, 158  
    control, 206  
Acceso directo a memoria, 472-511  
    80186/80188, 629-631  
    controlador, 474-485  
    generación de señales de control, 473-474  
    operaciones básicas, 473  
    sincronización, 473  
Active X, controles, 255-258  
Acumulador, 51  
ADA, 5  
ADC, 152  
ADD, 149  
AGP, 14, 18, 604  
ALE, 296  
ALGOL, 5  
ALIGN, 136  
Alineación, comprobación, 54  
Altair 8800, 6  
Alto, 207  
AND, 166-167  
Anillo, 62  
Apuntador  
    de instrucciones, 52, 57, 183  
    libre, 57  
Archivo  
    apuntador, 274-275  
    de acceso aleatorio, 276-278  
    de acceso secuencial, 271-276  
    ejecución, 270  
    objeto, 239  
Archivos en disco, 268-278  
Área del sistema, 20-21  
ASCII, 33-35  
    aritmética, 164-166  
Asíncronos, datos seriales, 416  
ASSUME, 137  
ATA, 18, 503
- B**  
Babbage, Charles, 2, 4, 5, 23  
Banderas, 52-54  
    Instrucciones de suma, 150  
Bardeen, John, 4
- Base, 28, 32  
    apuntador, 51  
dirección, 61  
índice, 51  
BASIC, 5  
Baudios, velocidad de, 420  
BCD, 5, 35-36  
    aritmética, 163-164  
    coprocesador, 515  
desempaquetado, 35  
empaquetado, 25  
BCH, 31-32  
Bibliotecas, 242-245  
Binario, 28  
BIOS, 18, 21  
Bit, 5  
    instrucciones de evaluación, 171  
    instrucciones de exploración, 175  
Bit implícito, 41  
Bloqueo, 207  
Bomar Brain, 3  
Bootstrap (secuencia inicial de instrucciones de arranque), 268  
BOUND, 207-208  
Brattain, Walter, 4  
BSF y BSR, 175  
BSWAP, 132  
BT, BTC, BTR y BTS, 171  
Bus, 16, 24-27  
    árbitro, 491-493  
    búfer, 298-301  
    ciclos, 301  
    COM, puertos seriales, 595-597  
    demultiplexión, 296-298  
    interfaz, 574-604  
    ISA, 574-584  
    LPT, impresora en paralelo, 592-595  
    PCI, 584-592  
    sincronización, 301-305  
    USB (bus serial universal), 597-603  
BUSY, 206  
Byron, Augusta, Ada, 2, 5  
Byte, 6, 23  
    con signo, 36  
    sin signo, 36

- Byte de derechos de acceso, 62  
BYTE, 84, 91, 134
- C**  
C/C++, 5  
  aplicaciones DOS, 214-221  
  aplicaciones Windows, 221-230  
  creación de aplicaciones, 250  
  objetos de ensamblador separados, 230-235
- Caché, 662  
  L1, 12  
  L2, 12
- CAD, 8
- CALL, 95, 137, 197, 198-201
- Canalización, 662
- CBW, 161
- CDQ, 162
- CD-ROM, 19
- CE, 316
- Cercano, 94, 183
- Cero, 53
- CFile, 272
- char, 38
- CISC, 7, 14
- CLC, 206
- CLD, 122
- CLI, 121, 205
- Clúster, 269
- CMC, 206
- CMOV, 132-133
- CMP, 156
- CMPS, 177
- CMPXCHG, 157
- CMPXCHG8B, 157
- COBOL, 5
- CodeView, 78
- Código de operación, 73, 79, 106
- Código, segmento de, 57
- COleDateTime, 279
- Colossus, 4
- COM, puertos seriales, 595-597
- Comentario, 79
- COMMAND.COM, 19-20
- Comparación, 156-157
- Complementos, 32-33, 171  
  a dos, 37, 171
- Con signo, 37
- Condicionales, saltos, 188-190
- Conectores  
  CENT36, puerto de impresora, 369, 593  
  DB15, video, 507  
  DB25, puerto de impresora, 369, 593  
  DB25, RS-232C, 464  
  DB9, video, 506  
  DVI-D, 507  
  USB, 598
- Conteo, 51
- Continuación, 54
- Control, bus de, 26-27  
  terminales para memoria, 316
- Controladores, 19
- Conversiones  
  7 segmentos, 264
- ASCII, 33-35  
BCD, 35-36  
BCH, 31-32  
binario a BCD, 165  
byte, 36-38  
complementos, 32-33  
de datos, 260-268  
decimal, 29-31  
doble palabra, 39-40  
números reales, 41-42  
palabra, 38-39  
punto flotante, 41-42  
unicode, 34
- Coprocessador aritmético, 513-569  
  arquitectura, 518-522  
  datos  
    formatos, 514-517  
    instrucciones de transferencia, 523-524  
  instrucciones aritméticas, 524-526  
  instrucciones de comparación, 526-527  
  instrucciones de control 528-530  
  instrucciones trascendentales, 527-528  
  listado de instrucciones, 530-546  
  operaciones de constantes, 528  
  programación, 547-551
- Coprocessador numérico (vea coprocesador aritmético)
- CPU, 22  
CPUID, 723-724, 755-758
- CS, 316
- Cuarteto, 5
- CWD, 161
- D**  
DAA, 163  
DAS, 163  
Datos  
  bus de, 25-26  
  cifrado, 283  
  constante, 77  
  estructuras, 93-94  
  instrucciones de movimiento, 104-143  
  MMX, 552  
  registro de, 51  
  segmento de, 83  
  terminales de, 315  
  transferencia de datos tipo cadena, 121-128  
  variable, 77
- DB, 35, 36, 37, 134
- DC, características, 289-290
- DD, 39, 134
- DDR, 357
- DEBUG, 85, 228
- DEC, 155
- Decodificador  
  de 2 a 4 líneas, 330  
  de 3 a 8 líneas, 328-330  
  dirección de E/S, 372-380  
  dirección de memoria, 325-334  
  NAND, 325-326  
  PLD, 330-334
- Definiciones de segmento completo, 140-141
- Desbordamiento, 53, 158
- Descriptor, 60  
  de interrupción, 439  
  global, 60  
  local, 60
- Desplazamiento cíclico, instrucciones, 174-176
- Desplazamiento, 55, 81, 183
- Desplazamiento, dirección de, 55
- Desplazamiento, instrucciones, 172-175
- Desviado, exponente, 42
- DI, 122
- Diagramas de terminales  
  16550, 417  
  2716, 317  
  4016, 319  
  41256, 324  
  4464, 322  
  74LS138, 328  
  74LS139, 331  
  80186, 611  
  80188, 612  
  80286, 651  
  80386, 658  
  80386EX (integrado), 659  
  80486, 699  
  8086, 289  
  8088, 289  
  80C188EB, 636  
  8228, 310  
  8237, 475  
  8254, 407  
  8259A, 451  
  8284A, 294  
  8289, 492  
  82C55, 381  
  ADC0804, 426  
  DAC080, 424  
  LM70, 644  
  PAL16L8, 333  
  Pentium, 710  
  Pentium II, 741  
  Pentium III, 752  
  Pentium 4, 753  
  Procesador Pentium Pro, 728  
  USBN9603, 601
- Dígito, 28
- Dirección, 53, 106  
  bandera, 122  
  base, 61  
  bus, 25  
  de desplazamiento, 55  
  de segmento, 55  
  decodificación, 326-334  
  física, 65  
  lineal, 65  
  terminales de, 315
- Directorio raíz, 268
- DIV, 160
- División, 160-162
- DMA (vea Acceso directo a memoria)
- Doble  
  palabra, 23, 39-40  
  precisión, 41

- DOS, 18  
 double, 42  
 DQ, 42, 134  
 DRAM, 319-326  
 DT, 134  
 DUMPBIN, 243  
 DUP, 136  
 DVD, 19  
 DW, 39, 134  
 DWORD, 39, 84
- E**  
 E/S  
     8255, 380-383  
     aislada, 364  
     bancos, 375  
     circuitos básicos, 369-371  
     dirección, 22  
         decodificación, 372-380  
     espacio, 22  
     instrucciones, 363-364  
     interfaz, 362-430  
         convertidor de análogo a digital, 425-428  
         convertidor de digital a análogo, 424-425  
         de comunicaciones programable, 416-424  
         detección de fallo de energía, 443  
         entrada básica, 365-367  
         LCD, 388-392  
         LED, 383-387  
         matriz de teclado, 394-399  
         motor de velocidad gradual, 392-394  
         respaldo de batería, 443  
         salida básica, 367  
         temporizador, 406-416  
         velocidad y dirección de motor de CD,  
             412-416  
     introducción, 130-131  
     mapa, computadora personal, 365-366  
     nivel de privilegios, 53  
     por asignación de memoria, 364  
     protocolo de intercambio, 367-368  
     puerto, 22  
 ECC, 341  
 EDIT, 78  
 EDO, 357  
 EEPROM, 317  
 EISA, 17  
 EMS, 20  
 END, 141  
 Endian, 38  
 ENDM, 245  
 ENDP, 137, 197  
 ENDS, 93  
 Enfoque, 250  
 ENIAC, 4, 5  
 Enigma, máquina, 4  
 Enlazador, 239-240  
 Enmascaramiento, 167  
 Ensamblador, 239-240  
 ENTER, 208  
 EPIC, 15  
 EPROM, 316  
 EQU, 137
- ESC, 207  
 Estado de espera, 302, 670-672  
 Estados T, 301  
 Etiqueta, 79, 183, 186  
 Exponente, 42  
 EXTERN, 231  
 EXTRN, 240-241
- F**  
 Faggin, Federico, 4  
 FAR, 120  
 Faraday, Michael, 2  
 FAT, 268  
 Filtrado, 252  
 Flash, memoria, 317  
 float, 42  
 FLOWMATIC, 3  
 FORTRAN, 5  
 Fuente de energía, requerimientos  
     8086, 289  
     8088, 289  
 Función (vea Procedimiento)
- G**  
 G, 7  
 GDTR, 64  
 Generador de reloj, 293-296  
 Granularidad, 61  
 GUI, 8
- H**  
 H, 78  
 Hexadecimal, 18, 32  
 HLDA, 472  
 HLT, 207  
 Hoff, Ted, 4  
 HOLD, 472  
 Hollerith, Herman, 3  
 Horner, algoritmo de, 261  
 Hyper-threading, 15, 754-755
- I**  
 IBM, 3  
 iCOMP, 11  
 IDE, 503  
 Identificación, 54  
 IDIV, 160  
 IDTR, 54  
 IMUL, 158  
 IN, 130-131, 363  
 INC, 151  
 INCLUDE, 247  
 Índice  
     de destino, 51  
     escalado, 75, 92-93, 111  
 Inmediato, 73  
 INS, 127, 363  
 Instrucciones  
     de comparación de cadenas, 176-177  
     de salto, 182-192  
         lógicas, 166-172  
 INT, 203  
 int, 39
- INTA, 443-446  
 Interrupción, 53, 121, 434-469  
     80186/80188, 618-624  
     cadena tipo margarita, 448-451  
     controlador programable, 451-465  
     de teclado, 446-467-469  
     expansión, 448-451  
     hardware, 442-448  
     instrucciones, 203-205, 438  
 INTR, 442  
 introducción, 202-207  
 modo protegido, 439  
 modo real, 438-439  
 NMI, 442  
 propósito de, 434-435  
 reloj en tiempo real, 465-467  
 vectores de, 202-203, 435-438
- Intersegmentos, 94-183  
 INTO, 205  
 INTR, 205, 442, 443-446  
 Intrasegmentos, 95-183  
 Intro, tecla, 34  
 IO/M, 301  
 IORC, 26  
 IOWC, 26  
 IRET, 202, 204  
 ISA, 17, 574-584  
 ISP, 202, 474  
 Itanium, 15
- J**  
 Jacquard, Joseph, 2  
 JAVA, 5  
 JCXZ, 190  
 JECXZ, 190  
 Jerga computacional, 1  
 JMP, 94, 182
- K**  
 K, 6  
 Kilby, Jack, 4  
 KIP, 5
- L**  
 LAHF, 129  
 LDS, LES, LFS, LGS y LSS, 120-121  
 LDTR, 65  
 LEA, 118-119  
 LEAVE, 208  
 Lejano, 94, 183  
 Lenguaje ensamblador, 4, 113, 134-143  
 Lenguaje máquina, 4, 105-113  
 LIB, 242  
 Listo, 289, 302, 306-308  
 LOCAL, 246  
 LODS, 122-123  
 LOOP, 85, 190  
 LOOPE, 192  
 LOOPNE, 192  
 LPT, impresora en paralelo, 592-595
- M**  
 M, 7  
 MACRO, 245

- Macro, 245-247  
 Mantisa, 41  
 Máquina analítica, 2  
 MASM, 78, 86, 104, 134  
 Mazor, Stan, 4  
**Memoria**  
 bancos, 341, 348, 351  
 conexiones de terminales, 315-316  
 convencional, 55  
 corrección de errores, 338-341  
 decodificación de dirección, 326-334  
 direccionamiento, 55-65  
 DOS, 17  
 DRAM, 319-326, 355-358  
 estática, 318  
 física, 26-27  
 flash, 337-338  
 intercalada, 662  
 interfaz, 314-358  
     80386DX/80486 (32 bits), 348-351  
     8086/80186/80286/80386SX (16 bits), 341-348  
     8088/80188 (8 bits), 335-341  
     Pentium-Pentium 4 (64 bits), 351-354  
 mapa, 747  
 paginación, 65-68, 694-698  
 párrafo, 56  
 protegida, 60-65, 674-682  
 real, 55  
 refresco, 355  
 ROM, 316-318  
 SRAM, 318-319  
 superior, 56  
 tiempo de acceso, 303, 614, 669  
 unidad de selección de chip (80186/80188), 631-635  
 volátil, 318  
 Windows, 21  
**Memoria en disco**  
 CD-ROM, 503-504  
 duro, 500-503  
 DVD, 504  
 floppy estándar, 496  
 micro-floppy, 499-500  
 mini-floppy, 496-498  
 WORM, 504  
 MFC, 224  
 MFT, 268  
**Microprocesador**  
 arquitectura, 49-68  
 banderas, 52  
 bus, 16  
 diagrama de bloques, 16  
     80186, 80188, 609  
 espacio de E/S, 22  
 extensión de 64 bits, 759-761  
 información general, 22-27  
 mapa de memoria, 17, 21  
 memoria física, 26-27  
 modelo de programación, 50-55  
 modo  
     de direccionamiento, 72-98  
     protegido, 60-65  
     real, 55-59  
 programa, 22  
 tamaños de bus y de memoria, 25  
 tareas, 22  
     vista conceptual, 15  
**MIP**, 7  
**ML**, 239  
**MMX**, tecnología, 9, 24  
 instrucciones  
     conjunto, 553-554  
     listado, 555-562  
 introducción, 531-563  
 registros, 552  
     tipos de datos, 552  
**MOD**, campo, 107  
**Modelo de programación**, 50-55, 78  
**Modelos**, 139  
**Modo**  
     protegido, 60-65, 105, 674-682  
     cambiar a, 682-692  
     real, 55-59, 105  
**Modos de direccionamiento**, 72-98  
 base  
     más índice, 75, 85-87  
     relativa más índice, 75, 90-91  
 de índice escalado, 75, 92-93, 111  
 de memoria de programa, 94-96  
     directo, 94  
     indirecto, 95-96  
     relativo, 95  
 de pila, 96-98  
 de registro, 73, 76-77  
     indirecto, 75, 82-85  
     relativo, 75, 88-89  
 directo, 73, 80-82  
 especial, 109-110  
 inmediato, 73, 77-80  
 por desplazamiento, 81-82  
**Motor de velocidad gradual**, 392-394  
**MOV**, 72, 76, 105-113  
**MOVS**, 73, 124-127  
**MOVSX**, 132, 160  
**MOVZX**, 132, 161  
**MRDC**, 26  
**MUL**, 158  
**Multiplicación**, 158-159  
**MWTC**, 26  
  
**N**  
**NEG**, 171  
**NMI**, 442  
**NOP**, 207  
**NOT**, 171  
**Notación posicional**, 28-29  
**NTFS**, 268  
**Números reales**, 41-42  
  
**O**  
**Octal**, 28  
**OE**, 316  
**OEM**, 504  
**OFFSET**, 84, 119  
**OnInitDialog**, 250  
**OnTimer**, 256  
**Operando**, 73, 79  
**OR**, 167-169  
**ORG**, 137  
**Origen**  
     índice, 52  
     módulo, 239  
**OUT**, 130-131, 363  
**OUTS**, 128, 363  
**OverDrive**, 9  
**OWORD**, 84, 563  
  
**P**  
**Paginación**, 65-68  
**Páginas**  
     directorio de, 66  
     tabla de, 66  
**Palabra**, 23, 38-39  
     cuádruple, 24  
     octal, 24  
**Pantalla**  
     interfaz LCD, 388-392  
     interfaz LED, 383-387  
**RGB**, 506-511  
**TTL**, 505-506  
     video, 504-511  
**Paridad**, 52  
**Párrafo**, 56  
**PASCAL**, 5  
**Pascal**, Blaise, 2  
**PC integrada**, 8  
**PCB**, 617  
**PCI Express (Xpress)**, 592  
**PCI**, 14, 17, 584-592  
**Pel**, 8  
**Pentium** 4, 14  
**Pentium II**, 13  
**Pentium III**, 14  
**Pentium Pro**, 12  
**Pentium**, 9  
**Pila**  
     apuntador, 52, 96  
     direccionamiento de memoria, 96-98  
     registro de segmento, 54  
     segmento de, 83, 96  
**Píxel**, 8  
**POP**, 86, 115-118  
**POPA**, 97, 226  
**POPF**, 226  
**PowerPC**, 11  
**Precisión simple**, 41  
**Prefijo**  
     de sustitución de segmento, 133-134  
     de tamaño de dirección, 106  
     de tamaño de registro, 106  
**PreTranslateMessage**, 251  
**Primer/último flip-flop**, 479  
**PROC**, 137, 197  
**Procedimiento**, 137, 197-202  
**Programa**, 22  
     8255, 382  
     cargador, 58

- flujo, control de, 192-197  
modular, 239-247
- Programmer's WorkBench, 78, 85
- PROM, 316
- PTR, 84, 120, 134
- PUBLIC, 240-241
- Puerto, 22, 24  
fijo, 130  
variable, 131
- Punto flotante, 41-42  
coprocesador, 515-517
- PUSH, 96, 113-115
- PUSHA, 97, 114
- PUSHF, 114
- Q**
- QWORD, 84
- R**
- R/M, 106-108
- RAMBUS, 14
- Ratón, 258-260
- RCL, 175
- RCR, 175
- RD, 301
- RDTSC, 724-726
- Refresco, 355
- Registros  
código, 54, 57  
de control, 65-66, 672, 719  
de datos, 54  
de pila, 54  
de propósito especial, 52-54  
de segmento, 54-55  
del coprocesador, 518-522  
depuración y prueba, 673  
específicos del modelo, 726-727, 758-759  
extra, 54  
invisibles para el programa, 50  
MMX, 552  
modos de direccionamiento, 72-98  
multipropósito, 51-52  
predeterminados, 56-57  
SSE, 563  
visibles para el programa, 50
- Reinicio, 293, 298, 473
- REP, 124
- REPE, 176
- REPNE, 176
- Resta, 153-157  
con acarreo, 155
- RET, 197, 201-202
- Retorno (vea RET)  
de carro (CR), 34
- Retrasos de tiempo, 387
- Reubicable, 58-59  
salto, 185
- RISC, 11, 14
- RLL, 501-502
- ROL, 175
- ROM, 316-318
- ROR, 175
- RPG, 5
- RPL, 62
- S**
- SAHF, 129
- SAL, 173
- Salto de línea (LF), 34
- SAR, 173
- SATA, 18, 503
- SBB, 155
- SCAS, 176
- SCSI, 502
- SDRAM, 14, 357
- Sector, 269
- Segmento  
de código, 57  
de dirección, 55  
de pila (vea pila)  
instrucciones MOV, 112, 113  
predeterminado, 56-57  
prefijo de sustitución, 133-134  
registros, 54-55  
modo protegido, 63
- SHL, 173
- SHLD, 174
- Shockley, William, 4
- short, 39, 94, 183
- SHR, 173
- SHRD, 174
- SI, 122
- Signo, 53
- SIMD, 5, 24, 563
- Sin signo, 36
- Sincronización de escritura, 301, 305
- Sincronización de lectura, 302-305
- Sistema con subprocesamiento, 646-650
- Sistema operativo en tiempo real, 636-642, 643
- Sistemas numéricos, 27-33
- SMALL, 81
- SRAM, 318-319
- SSE, tecnología, 563-569  
conjunto de instrucciones, 564-565  
formatos de datos, 564  
registros, 563
- STC, 206
- STD, 122
- STI, 121, 205
- STOS, 123-124
- STRUC, 93
- SUB, 154
- Subrutina, 137, 197-202
- Suma, 148-153  
con acarreo, 152
- T**
- Tablas de búsqueda, 264-268
- Tabulating Machine Company, 3
- Tarea anidada, 53
- TASM, 78, 104
- Teclado  
exploración de códigos, 248-249  
interfaz de matriz de, 394-399  
interrupción de, 446, 467-469
- Temporizador, 406-416  
80186/80188, 624-629
- TEST, 170-171, 206
- THIS, 91, 137
- TI, 62
- Tiempo de acceso, 303, 614, 669
- TINY, 78
- TLB, 67
- TPA, 16, 18-20
- TR, 65
- Trampa, 53
- Turing, Alan, 4
- U**
- Unicode, 34
- Unidad de almacenamiento portátil USB, 500
- Unidad de selección de chip, 631-635
- Universidad de Pennsylvania, 4
- USB, 18, 597-603
- USE16 y USE32, 141
- USES, 138, 197
- V**
- VESA, 9, 17
- VGA, 8
- VHDL, 331
- Video  
pantalla (vea pantalla, video)  
señales, 504-505
- Virtual  
interrupción, 54  
modo, 54, 692-693  
pendiente, 54
- Von Neumann, John, 4
- W**
- W, 107
- WAIT, 206-207, 306, 308
- WE, 316
- WIN32, 61, 221
- Windows, 8  
colores del sistema, 267  
DDK, 214  
mapa de memoria, 21
- WM\_CHAR, 251
- WM\_TIMER, 256
- WORD, 84
- WR, 301
- WYSIWYG, 8
- X**
- XADD, 153
- XCHG, 128-129
- Xeon, 13
- XLAT, 129, 264
- XMS, 16
- XOR, 160-170
- Z**
- Z3, 3
- Z80, 7
- Zuse, Konrad, 3