

Introducción al 80386

Este libro presenta el microprocesador 80386 de Intel a ingenieros de *software*, de *hardware* y directivos de empresas de informática.

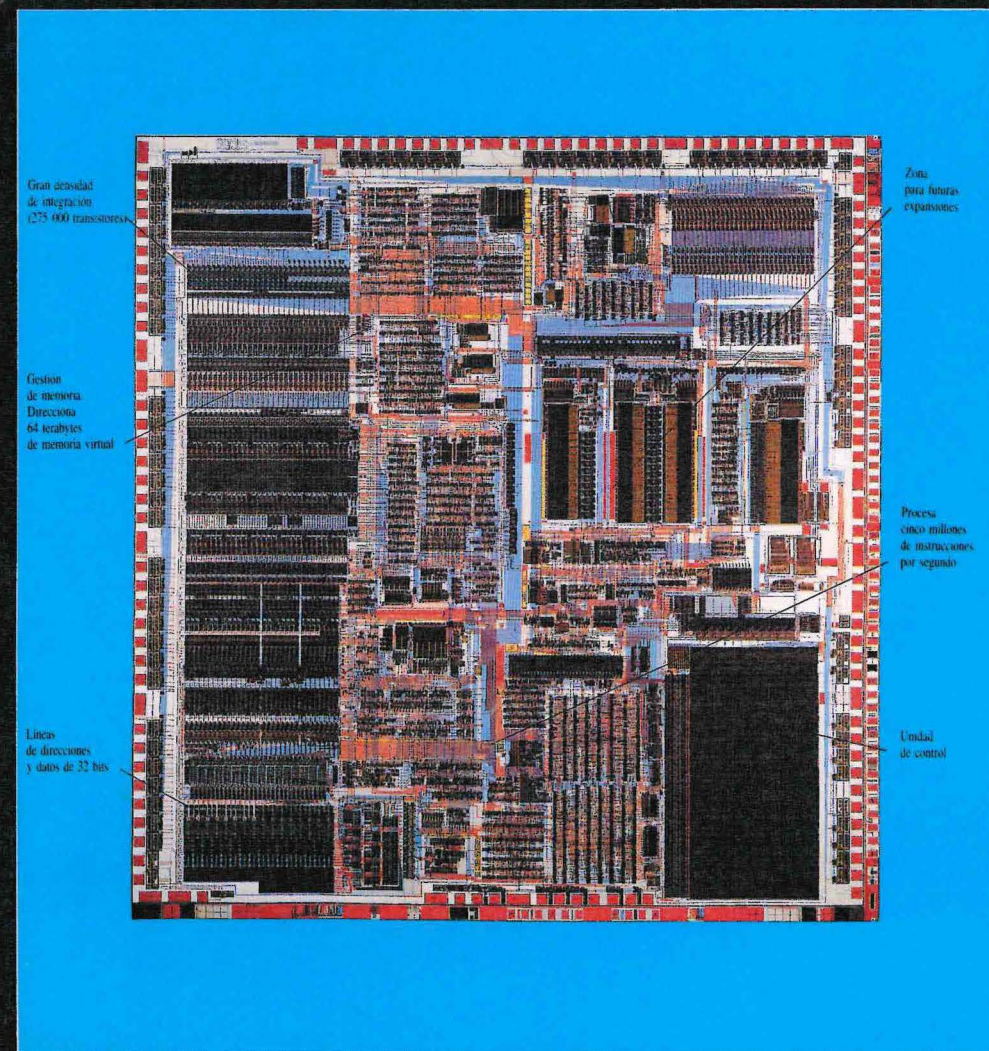
A estos últimos les interesará más en particular el capítulo 1, ya que resume las características y ventajas del 80386. Los dos capítulos siguientes están dirigidos a ingenieros de *software* de aplicaciones y de sistemas, respectivamente. El capítulo 2 describe la arquitectura del 80386 tal como la vería un programador en lenguaje ensamblador o un escritor de compiladores; el capítulo siguiente analiza aquellos aspectos de la arquitectura que soportan los sistemas operativos. Los lectores interesados en ejecutar el *software* existente del 8086 y/o del 80286 sobre el 80386, deberán leer el capítulo 4. El último capítulo es para ingenieros de *hardware* y describe principalmente el interfaz externo del 80386.

Debido a que todos los temas son tratados sin profundizar, puede ser interesante ojear aquella materia incluso sin ser expertos en ella.

OTROS TITULOS DE LA COLECCION

**80386 Guía del programador
y Manual de referencia**

**80386 Manual de referencia
Hardware**



Introducción al 80386

Introducción al 80386

R. - 6480



Indice

Introducción	9
1. Características principales	11
1.1. Arquitectura de 32 bits	12
1.2. Implementación de alto rendimiento	12
1.3. Soporte de memoria virtual	13
1.4. Protección configurable	14
1.5. Soporte de depuración ampliado	14
1.6. Compatibilidad de código objeto	15
1.7. Resumen	15
2. Arquitectura para la aplicación	17
2.1. Registros	17
2.1.1. Registros generales	18
2.1.2. <i>Flags</i> y puntero de instrucciones	19
2.1.3. Registros del coprocesador numérico	19
2.2. Memoria y direccionamiento lógico	20
2.2.1. Segmentos	20
2.2.2. Direcciones lógicas	20
2.2.3. Registros de descriptores y de segmentos	21
2.2.4. Modos de direccionamiento	23

Introducción

2.3. Tipos de datos e instrucciones	24
2.3.1. Tipos de datos principales	24
2.3.2. Tipos de datos del coprocesador numérico	24
2.3.3. Otras instrucciones	26
2.3.3.1. Instrucciones de pila	26
2.3.3.2. Instrucciones de transferencia de control	26
2.3.3.3. Instrucciones diversas	27
3. Arquitectura de sistema	29
3.1. Registros del sistema	29
3.2. Multitarea	29
3.2.1. Segmento de Estado de Tarea	30
3.2.2. Conmutación de tareas	31
3.3. Direccionamiento	31
3.3.1. Visión general de la traducción de direcciones	32
3.3.2. Segmentos	34
3.3.3. Páginas	35
3.3.4. Memoria virtual	39
3.4. Protección	40
3.4.1. Prioridades	40
3.4.2. Instrucciones prioritarias	42
3.4.3. Protección de segmento	42
3.4.4. Protección de páginas	43
3.5. Llamadas al sistema	43
3.6. Interrupciones y excepciones	45
3.6.1. Tabla de Descriptores de Interrupciones	46
3.6.2. Excepciones y registros de depuración	47
3.7. Entrada/Salida	47
4. Compatibilidad de la arquitectura	49
4.1. Compatibilidad con el 80286	49
4.2. Modos Real y Virtual 86	50
5. Implementación <i>hardware</i>	53
5.1. Diseño interno	53
5.2. Interfaz externo	55
5.2.1. Reloj	56
5.2.2. Buses de datos y de dirección	56
5.2.3. Definición de ciclo de <i>bus</i>	57
5.2.4. Control del ciclo de <i>bus</i>	57
5.2.5. Dimensionamiento dinámico del <i>bus</i>	60
5.2.6. Control y estado del procesador	60
5.2.7. Control del coprocesador	62

Este libro presenta el microprocesador 80386 de Intel a ingenieros de *software*, de *hardware* y directivos de empresas de informática.

A estos últimos les interesará más en particular el capítulo 1, ya que resume las características y ventajas del 80386. Los dos capítulos siguientes están dirigidos a ingenieros de *software* de aplicaciones y de sistemas, respectivamente. El capítulo 2 describe la arquitectura del 80386 tal como la vería un programador en lenguaje ensamblador o un escritor de compiladores; el capítulo siguiente analiza aquellos aspectos de la arquitectura que soportan los sistemas operativos. Los lectores interesados en ejecutar el *software* existente del 8086 y/o del 80286 sobre el 80386, deberán leer el capítulo 4. El último capítulo es para ingenieros de *hardware* y describe principalmente el interfaz externo del 80386.

Debido a que todos los temas son tratados sin profundizar, puede ser interesante ojear aquella materia incluso sin ser expertos en ella.

Nota: A lo largo de todo el texto, los términos clave están escritos en **negrita**.

Características principales

El 80386 es un microprocesador de 32 bits de alto rendimiento diseñado para realizar las aplicaciones más avanzadas sobre ordenadores, tanto actuales como futuras. Las estaciones de trabajo CAD/CAE, los terminales gráficos de alta resolución, la publicación editorial y la automatización de oficinas y fábricas, son ejemplos en la actualidad de aplicaciones bien asistidas por el 80386. Las aplicaciones futuras estarán probablemente más restringidas por la imaginación de los diseñadores del sistema que por la potencia y versatilidad del 80386.

El 80386 ofrece al diseñador de sistemas muchas posibilidades nuevas y potentes, que incluyen un rendimiento sin precedentes de tres a cuatro millones de instrucciones por segundo, una arquitectura completa de 32 bits, un espacio físico de direcciones de 4 gigabytes (2^{32} bytes), y soporte en el propio *chip* para memoria virtual paginada. Además de englobar lo más avanzado en tecnología de microprocesadores, el 80386 mantiene compatibilidad de código objeto con el abundante

software escrito para sus antecesores, el 8086 y el 80286. De especial interés es la capacidad de máquina virtual del 80386, que le permite al 80386 conmutar entre programas ejecutándose bajo diferentes sistemas operativos, tales como el UNIX y el MS-DOS. Esta facilidad permite a los OEM (del inglés, *Original Equipment Manufacturers*) incorporar directamente *software* de aplicaciones estándar de 16 bits en los nuevos diseños de 32 bits.

Combinando la potencia y el rendimiento de un superminiordenador con el bajo coste y versatilidad en el diseño de un microprocesador, el 80386 puede abrir nuevos mercados a los sistemas basados en microprocesadores. Las aplicaciones que no fueron posibles con los lentos microprocesadores o que fueron poco rentables con los superminiordenadores, ahora son posibles con el 80386. Aplicaciones tales como la visión artificial, reconocimiento del lenguaje, robots avanzados y sistemas expertos, que ya han sido muy experimentadas, se pueden introducir ahora en el mercado.

Para afrontar con eficacia aquellas aplicacio-

nes que son los retos del mañana, se requiere algo más que registros de 32 bits, instrucciones y buses. Estos recursos fundamentales son sólo el punto de partida para el 80386. Las siguientes secciones resumen la arquitectura del 80386 junto con sus características más innovadoras:

- Implementación de alto rendimiento.
- Soporte de memoria virtual.
- Protección configurable.
- Soporte de depuración ampliado.
- Compatibilidad del código objeto.

1.1. Arquitectura de 32 bits

La arquitectura de 32 bits del 80386 proporciona los recursos de programación necesarios para soportar directamente aplicaciones "grandes" —aquellas que se caracterizan por enteros de gran tamaño, estructuras grandes de datos, programas grandes (o un número grande de programas), etc—. El espacio físico de direcciones del 80386 es de 2^{32} bytes o 4 gigabytes; su espacio lógico de direcciones es de 2^{46} bytes o 64 terabytes. Los ocho registros generales de 32 bits del 80386 pueden ser usados alternativamente como operandos de instrucciones y como variables de modo de direccionamiento. Los tipos de datos que existen incluyen enteros de 8, 16 y 32 bits, y ordinales, decimales empaquetados o sin empaquetar, punteros y cadenas de bits, bytes, palabras y palabras dobles. El 80386 posee un repertorio completo de instrucciones para la manipulación de todos estos tipos, así como para el control de la ejecución. Los modos de direccionamiento del 80386 soportan accesos eficaces a los elementos de las estructuras de datos estándar: arrays, registros, arrays de registros y registros conteniendo arrays.

1.2. Implementación de alto rendimiento

Una arquitectura de 32 bits no garantiza un alto rendimiento. Para que esto ocurra y podamos aprovechar el poder de esta arquitectura, se requiere tecnología punta de semiconductores, una división cuidadosa de funciones

y prestar atención en las operaciones que se realizan fuera del *chip*, particularmente en la interacción entre el procesador y la memoria. El 80386 incorpora todo esto y proporciona un rendimiento más alto que cualquier microprocesador disponible en la actualidad.

El 80386 está implementado con la tecnología de Intel CHMOS III, un proceso de fabricación de semiconductor que combina la alta frecuencia de la tecnología HMOS con los modestos requerimientos de potencia de la tecnología CMOS. Con geometrías de 1.5 micras y dos láminas de metal, el 80386 alberga 275 000 transistores en un solo *chip*. Inicialmente están disponibles tanto la versión de 12 Mhz como la de 16 Mhz del 80386; funcionando sin estados de espera, el de 16 Mhz puede conseguir de forma ininterrumpida velocidades de ejecución de tres a cuatro millones de instrucciones por segundo.

Internamente, el 80386 se divide en seis unidades que operan de forma autónoma y en paralelo unas con otras con la necesaria sincronización. Todos los buses internos que conectan estas unidades son de 32 bits. Conectando en *pipeline* sus unidades funcionales, el 80386 puede superponer la ejecución de diferentes etapas de una instrucción y puede procesar simultáneamente varias instrucciones. Así, mientras una instrucción se ejecuta, otra se decodifica, y una tercera es traída desde la memoria.

Además de ejecutar en *pipeline* todas las instrucciones, el 80386 aplica un *hardware* dedicado a realizaciones de operaciones importantes. La unidad de multiplicación/división del 80386 puede ejecutar una multiplicación de 32 bits entre 9 y 41 ciclos de reloj, dependiendo del número de dígitos significativos; puede dividir operandos de 32 bits en 38 ciclos de reloj (sin signo) o en 43 ciclos (con signo). El desplazador de tambor del 80386 puede desplazar entre 1 y 64 bits en un solo ciclo de reloj.

Muchas de las aplicaciones de 32 bits, tales como los ordenadores multiusuario reprogrables, necesitan la protección y traducciones de direcciones lógicas a direcciones físicas que proporciona la unidad de manejo de memoria, o MMU (del inglés, *Memory Manage-*

ment Unit). No ocurre lo mismo con otras aplicaciones, como, por ejemplo, los sistemas en tiempo real embebidos, que no hacen uso de esa unidad. La mayoría de las arquitecturas de microprocesadores de 32 bits responden a esta dicotomía implementando la unidad de gestión de memoria en un *chip* opcional. La MMU del 80386, por el contrario, está incorporada sobre el propio *chip* del procesador, estando constituida por dos de las unidades funcionales del procesador conectadas en *pipeline*. El sistema operativo controla el funcionamiento de la MMU, permitiendo a un sistema en tiempo real, por ejemplo, renunciar a la traslación de páginas. La implementación de la unidad de manejo de memoria en el propio *chip* consigue un mejor rendimiento para aquellas aplicaciones que hacen uso de la MMU, mientras que no se produce ningún efecto negativo en el mismo en aquellas otras que no la usan. Este logro ha sido posible reduciendo los retardos en la propagación de la señal, usando los semiciclos de reloj que están disponibles en el propio *chip* y operando en paralelo.

Otra utilidad que es crucial para algunas aplicaciones e irrelevante para otras es la capacidad como "devorador de números"; particularmente para la aritmética de punto flotante de simple y doble precisión. Los operandos de punto flotante son grandes y el conjunto de operaciones sobre ellos es útil pero bastante complejo; se requieren muchos miles de transistores para implementar un conjunto estándar de operaciones de punto flotante, tales como las definidas por el estándar 754 de IEEE. Consecuentemente, el 80386 proporciona soporte *hardware* para operaciones numéricas en un *chip* coprocesador numérico separado. De hecho, cada uno de los dos *chips*, el coprocesador numérico 80287 o el 80387, de mayor rendimiento, pueden ser conectados al 80386. Los coprocesadores numéricos son invisibles para el *software* de las aplicaciones; estos *chips* amplían de forma eficaz la arquitectura del 80386 con registros, tipos de datos e instrucciones compatibles con la norma IEEE 754. Combinando un 80386 y un 80387 se puede ejecutar 1.8 millones de operaciones por segundo.

Cuando un procesador de 32 bits funciona a 16 Mhz, puede superar a todas las memorias excepto a las más rápidas, haciendo que el tiempo de acceso a la memoria se convierta en un posible cuello de botella llegando a saturarse. El bus del 80386 ha sido diseñado para hacer el mejor uso, tanto de las RAM estáticas, muy rápidas, como de las RAM dinámicas, más baratas que las anteriores. Para acceder a la memoria más rápida, como las *cachés*, el 80386 proporciona un ciclo de bus de acceso a memoria de dos ciclos de reloj. (Las *cachés* del 80386 pueden tener cualquier tamaño, desde una capacidad mínima de 4 kilobytes hasta ocupar el espacio físico de direcciones completo.) Para acceder a las memorias más lentas (o dispositivos I/O, entrada/salida) se puede emplear la facilidad del 80386 para gestionar direcciones en *pipeline*, que extiende el tiempo efectivo de dirección a memoria a tres impulsos de reloj, mientras se mantiene en dos para el procesador. A causa de la gestión interna en *pipeline* de la traducción de direcciones junto con la ejecución de instrucción, el 80386 generalmente calcula la dirección y definición del siguiente ciclo de bus durante el ciclo de bus actual. La gestión en *pipeline* de las direcciones da cuenta de esta información adelantada al subsistema de memoria, permitiendo que un banco de memoria decodifique durante el siguiente ciclo de bus mientras que otro banco está respondiendo durante el ciclo actual.

1.3. Soporte de memoria virtual

La memoria virtual permite que el tamaño máximo de un programa, o de un conjunto de éstos, sea gobernado por el espacio disponible en el disco, en lugar de serlo por el tamaño de la memoria física (RAM), que actualmente es del orden de cuatrocientas veces más cara. El resultado beneficia a los fabricantes (que pueden proporcionar niveles de rendimiento múltiple de un producto, los cuales difieren sólo en las configuraciones de memoria), a los programadores (que pueden

dejar al sistema operativo la gestión del almacenamiento en lugar de escribir rutinas específicas), y a los usuarios finales (que pueden ejecutar más y más grandes aplicaciones sin el problema de quedarse sin memoria).

La memoria virtual está implementada por un sistema operativo con soporte desde el *hardware*. El 80386 soporta sistemas de memoria virtual basados en segmentos o en páginas. La memoria virtual basada en segmentos es apropiada para sistemas pequeños de 16 bits cuyos segmentos tienen una longitud máxima de 64 kilobytes. El 80386, sin embargo, puede soportar segmentos de hasta 4 gigabytes; por consiguiente, los sistemas de mayor tamaño basados en el 80386 basarán su sistema de memoria virtual en las facilidades de petición de paginado del 80386. Para cada página, el 80386 provee los bits de presencia, modificación y acceso que se requieren para una implementación eficaz de la memoria virtual paginada. El 80386 envía automáticamente un *trap* al sistema operativo cuando una instrucción se refiere a una página no presente; cuando el sistema operativo ha intercambiado la página ausente desde el disco, el 80386 vuelve a ejecutar de forma automática la instrucción. Para asegurar el alto rendimiento de la memoria virtual, el 80386 provee en el propio *chip* una *caché* asociativa para la información de paginación. La *caché* (denominada *buffer* de consulta de traducción o TLB, siglas correspondientes al término inglés) contiene la información de mapeado correspondiente a las 32 páginas usadas más recientemente. Las páginas del 80386 tienen una longitud de 4 kilobytes; mediante el mapeado de 128 kilobytes de memoria, el TLB permite al 80386 traducir la mayoría de las direcciones en el propio *chip* sin la consulta de una tabla de páginas ubicadas en memoria. En los sistemas típicos, del 98 al 99 por 100 de las direcciones referidas coincidirán con una entrada en el TLB.

1.4. Protección configurable

Al poder ejecutar de tres a cuatro millones de instrucciones por segundo, el 80386 tiene

la potencia suficiente para soportar aplicaciones extremadamente sofisticadas consistentes en cientos o miles de módulos de programa. En tales aplicaciones, la cuestión no es si existirán errores, sino sobre todo cómo pueden ser encontrados y eliminados tan rápido como sea posible. Estas aplicaciones pueden ser depuradas de forma más rápida y consistente si el procesador verifica cada instrucción con arreglo a los criterios de protección. El grado y estilo que se deberían aplicar son, no obstante, inherentes a las especificaciones de la aplicación. Efectivamente, las aplicaciones en tiempo real compactas pueden trabajar mejor sin protección de ningún tipo. Un rango de diferentes necesidades de protección se satisface mejor con un rango de capacidades de protección que puedan ser empleadas de forma selectiva, como las que proporciona el 80386:

- Separación de los espacios de direcciones de las tareas.
- De cero a cuatro niveles de prioridad.
- Instrucciones prioritarias (por ejemplo, Halt —parada—).
- Segmentos tipificados (por ejemplo, códigos o datos).
- Derechos de accesos para segmentos y páginas (por ejemplo, sólo-lectura o sólo-ejecución).
- Chequeo del límite del segmento.

Todos los chequeos de protección se ejecutan en el *pipeline* del propio *chip* a fin de obtener el máximo rendimiento.

1.5. Soporte de depuración ampliado

Los cuatro registros de depuración incorporados en el 80386 pueden reducir de forma significativa el tiempo de depuración de un programa. Estos registros operan independientemente del sistema de protección y pueden por tanto ser usados por todas las aplicaciones, incluidas aquellas que se van a ejecutar sin protección. Y, aún más importante, estos registros proporcionan la capacidad de poner *breakpoints* (puntos de ruptura) en

los datos además de las más familiares instrucciones de *breakpoints* posibles. El 80386 controla las cuatro direcciones de *breakpoints* de forma simultánea sin reducir la velocidad de ejecución.

Los *breakpoints* de instrucción provocan una interrupción (de forma habitual a un programa depurador) cuando se ejecuta una instrucción; la mayoría de los procesadores proporcionan esta capacidad con una instrucción especial que el programa depurador escribe sobre la instrucción de interés. Al especificar las direcciones de los *breakpoints* de instrucción en registros, el 80386 elimina las complicaciones que se requieren para escribir instrucciones de *breakpoints* en código protegido o compartido. Los *breakpoints* de datos, que son una característica extraordinaria para un microprocesador, son una herramienta de depuración particularmente útil. Un *breakpoint* de datos puede interrumpir en el instante en el que se lee o se escribe una dirección. Empleando estos *breakpoints*, un programador puede, por ejemplo, localizar inmediatamente la instrucción responsable de que se haya sobrescrito erróneamente una estructura de datos.

Además de los registros de *breakpoints*, el 80386 proporciona facilidades de depuración más convencionales como la instrucción de *breakpoint* y ejecución paso a paso.

1.6. Compatibilidad de código objeto

El 80386 ha sido precedido por dos generaciones de procesadores de la familia 86, el 80286 y el 8086. El 80386 es compatible a nivel binario con ambos. Esta compatibilidad protege las inversiones de *software*, permite una rápida entrada en el mercado, y puede proporcionar acceso a la vasta biblioteca de *software* escrita para los ordenadores basados en la familia 86.

El 80386 puede, por supuesto, ejecutar programas escritos para el 8086; también puede ejecutar de forma concurrente programas del 80286 y del 80386. Pero la característica más innovadora de la compatibilidad del 80386 es

la facilidad Virtual 86, que establece un entorno 8086 protegido dentro de la estructura multitarea del 80386. Complementando a la facilidad Virtual 86, el paginado del 80386 se puede usar para dar a cada tarea Virtual 86 un espacio de dirección de un megabyte en cualquier lugar del espacio físico de direcciones del 80386. Además, si el sistema operativo del 80386 soporta memoria virtual, las tareas del Virtual 86 pueden ser intercambiadas como otras tareas cualesquiera sin atención especial. En resumen, la facilidad Virtual 86 del 80386 permite que se ejecuten a la vez *software* de tres generaciones de la familia 86.

1.7. Resumen

El 80386 proporciona el funcionamiento en bruto necesario para implementar sistemas basados en microprocesadores de alto fin. La arquitectura del 80386 es flexible: los diseñadores de sistemas pueden elegir aquellas opciones que mejor se ajusten a las necesidades de la aplicación en lugar de estar vinculados a un tipo de máquina. Las completas utilidades de manejo de memoria, incluyendo el soporte para la segmentación, paginación y memoria virtual, están disponibles en el propio *chip*. Pueden usarse hasta cuatro niveles de protección para construir paredes estancas entre los componentes *software*, o renunciar completamente a esa protección. Las tareas de tipo Virtual 86 pueden enriquecer a los sistemas de 32 bits con la extraordinaria colección de *software* ya desarrollado para negocios y para otras máquinas de la familia 86.

La potencia y versatilidad del 80386 se puede aumentar mediante otros *chips* de Intel que ayudan a maximizar el rendimiento del sistema. Estos incluyen controladores de red de área local, controladores de DMA avanzado, controladores de disco y coprocesadores gráficos.

El tiempo de diseño y coste se puede reducir con ayuda de herramientas y equipos desarrollados por Intel. Estas herramientas incluyen compiladores, utilidades de carga (*load*) y acoplamiento (*linking*), sistemas operativos y un emulador (ICE™ 386). Se dispone de

cientos de equipos compatibles con el estándar industrial MULTIBUS I capaces de realizar funciones estándar ahorrando los costes del diseño y de pruebas; la serie de equipos MULTIBUS II de alto rendimiento está crecien-

do muy rápidamente. Finalmente, el equipo experimentado de Intel compuesto por ingenieros de aplicaciones y especialistas puede proporcionar una asistencia mundial en el campo del diseño.

Arquitectura para la aplicación

El 80386 proporciona al programador de aplicaciones en lenguaje ensamblador o al escritor de compiladores un conjunto extenso de recursos de 32 bits. El capítulo describe estos recursos en tres secciones: 1) registros, 2) memoria y direccionamiento lógico, y 3) tipos de datos e instrucciones.

2.1. Registros

Los ordenadores, incluido el 80386, proporcionan registros al programador que éste puede emplear para almacenar información local de forma muy rápida. Se puede acceder a los datos residentes en los registros sin ejecutar los ciclos del bus, mejorando así el tiempo de ejecución de instrucciones y dejando más anchura de banda de bus para otros procesadores, tales como los controladores de acceso directo a memoria. El 80386 proporciona a los programadores y compiladores ocho registros generales de uso general; un coprocesador numérico opcional 80287 u 80387 puede propor-

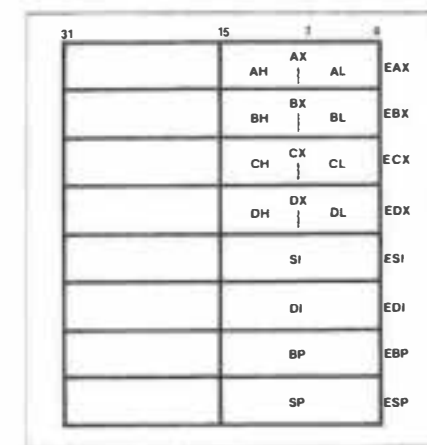


Figura 2.1. Registros de uso general

cionar otros ocho registros. Otros dos registros del 80386, que están orientados hacia el

control y estado del procesador en lugar del almacenamiento de datos, son muy importantes también para los programadores; estos dos registros son los registros de *flags* y el puntero de instrucciones.

2.1.1. Registros generales

Como muestra la figura 2.1, los registros generales del 80386 son de 32 bits, así como los caminos de datos internos del procesador, los *buses* de datos y los *buses* de dirección. Sin embargo, y siguiendo el precedente establecido por un cierto número de procesado-

res cuyos antecesores han sido máquinas de 16 bits, llamaremos **palabra** (*word*) del 80386 a un conjunto de 16 bits y **doble palabra** (*doubleword*, o *dword*) a uno de 32 bits.

Como muestra la figura 2.1, todos los registros generales pueden ser usados como registros de 16 ó 32 bits y cuatro de ellos, además, como registros de ocho bits. En casi todas las instrucciones, cualquier registro de uso general se puede especificar como un operando. Por ejemplo, dos registros cualesquiera pueden multiplicarse entre ambos. Y de forma similar, cualquier registro puede ser un registro base o índice en el cálculo de una dirección (analizado más adelante, en este capítulo).

tulo). Debido a que todos los programas de utilidad necesitan una pila (*stack*), el registro general ESP está definido de forma implícita como el puntero a la cabeza de la pila.

2.1.2. Flags y puntero de instrucciones

La figura 2.2 presenta el formato del registro de *flags* del 80386. Se pueden distinguir tres tipos de *flags*: de estado, de control y de sistema. El procesador pone a 1 los *flags* de estado después de muchas instrucciones para reflejar el resultado de la operación. Por ejemplo, cuando la comparación de dos operandos es igual, el procesador pone a 1 el *flag* cero. Otras instrucciones, sobre todo las de salto condicional, comprueban un *flag* de estado y se comportan de forma diferente de-

pendiendo del valor del *flag*. Los programadores pueden poner a 1 diversos *flags* de control para modificar la semántica de algunas instrucciones. Por ejemplo, la instrucción de búsqueda (*scan*) de cadenas busca hacia las direcciones más altas o más bajas dependiendo del valor del *flag* de dirección. Los *flags* del sistema sólo son usados por el sistema operativo y se pueden ignorar por parte de los programadores de aplicación. (Los *flags* del sistema serán estudiados en el capítulo 3.) De hecho, el sistema de protección del 80386 puede ser usado para prevenir que los programadores de aplicación alteren inadvertidamente los *flags* del sistema.

El puntero de instrucciones del 80386, denominado EIP, es de 32 bits; controla la búsqueda de instrucciones, incluida la prebúsqueda, y el procesador incrementa automáticamente este puntero después de ejecutar una instrucción. Las interrupciones, excepciones e instrucciones de transferencias de control, tales como los saltos y llamadas, alteran el puntero de instrucción.

2.1.3. Registros del coprocesador numérico

Los registros del coprocesador numérico que se muestran en la figura 2.3 mejoran el rendimiento de las aplicaciones numéricas. La conexión de un coprocesador numérico 80287 u 80387 a un 80386 agrega a este último los registros de dichos coprocesadores de forma efectiva. Un coprocesador numérico, además de reconocer enteros, decimales empaquetados y formatos de punto flotante de varias longitudes, mantiene internamente todos los valores en una pila de ocho registros de punto flotante de 80 bits de anchura. Las instrucciones numéricas se pueden referir de forma implícita a los elementos de la cabeza de la pila, o explícitamente a otros registros. El registro de estado mantiene el puntero a la cabeza de la pila, los *flags* que identifican las excepciones (por ejemplo, el desbordamiento), y los códigos de condición que reflejan el resultado de la última instrucción. El registro de control contiene bits de máscara y opcionales que

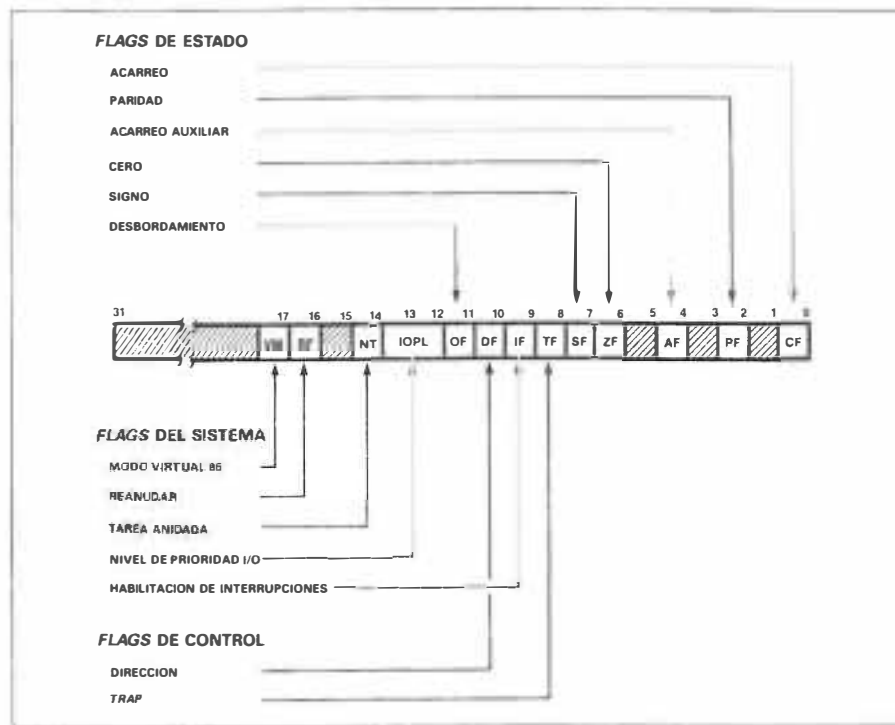


Figura 2.2 Registro de flags



Figura 2.3 Registros del coprocesador numérico

el programador puede poner a 1 para seleccionar el algoritmo de redondeo, la forma en que se va a modelar el infinito, y si las excepciones van a ser manejadas por el coprocesador o por el *software*.

2.2. Memoria y direccionamiento lógico

Los programas de aplicación del 80386 usan las direcciones lógicas para especificar las posiciones de los operandos en un espacio físico de direcciones de 4 gigabytes. El procesador traslada automáticamente estas direcciones lógicas a las direcciones físicas que transmite a través del *bus* del sistema. Como se estudia más a fondo en el capítulo 3, un sistema operativo basado en el 80386 puede conformar la visión que tiene un programa de aplicación de su espacio lógico de direcciones. Por ejemplo, un sistema operativo puede definir el espacio lógico de direcciones como se define en muchas arquitecturas, como un sencillo *array* de 2^{32} bytes. De forma alternativa, un sistema operativo puede organizar el espacio lógico de direcciones como una colección de *segmentos* de longitud variable. También puede definir muchos segmentos o sólo unos pocos, dependiendo de que sea lo más apropiado según su visión de la memoria lógica; el 80386 no impone el uso de los segmentos, sino que permite que se empleen de la forma en que soporten mejor las necesidades de la aplicación. Cuando lea esta sección, tenga en cuenta que la medida en la que un programa de aplicación usa de forma activa los segmentos, depende del marco establecido por el sistema operativo.

2.2.1. Segmentos

Como ya se mencionó, un sistema operativo puede definir el espacio de dirección lógica del 80386 como uno o más segmentos. Los segmentos son unidades lógicas que mapean bien las estructuras de programación y cu-

ya longitud es intrínsecamente variable. Por ejemplo, un procedimiento de 1516 bytes encaja exactamente en un segmento de 1516 bytes, lo mismo que un *array* de 8 megabytes (por ejemplo, un *buffer* de pantalla de $1028 \times 1028 \times 8$ bits) encaja perfectamente en un segmento del mismo tamaño. Al proporcionar soporte arquitectónico a los segmentos (por ejemplo, la posibilidad de que los segmentos pueden estar protegidos individualmente, o ser compartidos selectivamente entre las diferentes tareas), el 80386 mejora el rendimiento de los sistemas que eligen los segmentos como mecanismo de estructuración. (Las páginas, descritas en el capítulo 3, son de tamaño fijo y no mapean bien las construcciones de programación, pero, por otra parte, están mejor adaptadas a determinadas funciones del sistema operativo tales como el intercambio (*swapping*)).

Un segmento del 80386 puede tener cualquier tamaño desde 1 byte hasta 4 gigabytes. Para todos los segmentos, el sistema operativo mantiene un *descriptor*, definido por la arquitectura, que especifica los atributos del segmento. Estos atributos incluyen unas direcciones base y límite (de longitud) de 32 bits, e información de protección, que guarda contra el uso incorrecto de un segmento. Como los descriptors son mantenidos por el sistema operativo, éstos se analizarán con más amplitud en el capítulo 3. Los programas de aplicación tratan sólo indirectamente con los descriptors, refiriéndose a los segmentos mediante direcciones lógicas.

2.2.2. Direcciones lógicas

Debido a que un programa puede referirse a múltiples segmentos, una dirección lógica del 80386 debe poder identificar un segmento. Por tanto, una dirección lógica del 80386 consta de dos partes, un *selector de segmento* de 16 bits y un *desplazamiento (offset)* de 32 bits dentro del segmento seleccionado (véase figura 2.4). La parte selectora de la dirección lógica identifica a un descriptor de segmento. Conceptualmente, el procesador determina la dirección de un segmento usando el

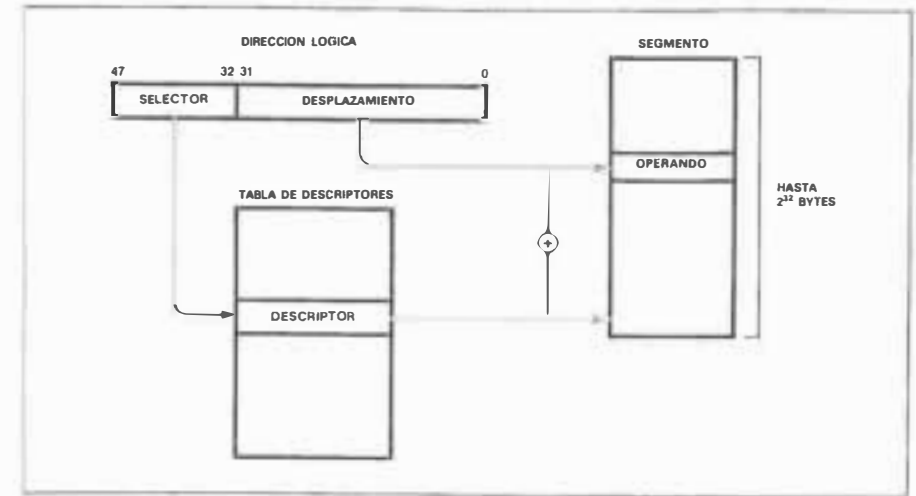


Figura 2.4 Traducción de la dirección lógica

selector como un índice en la *tabla de descriptors* mantenida por el sistema operativo. Añadiendo la parte de desplazamiento de la dirección lógica a la dirección de base obtenida del descriptor del segmento, se produce la dirección del operando.

2.2.3. Registros de descriptors y de segmentos

Para hacer más eficiente el direccionamiento lógico, el 80386 proporciona seis registros de *descriptor de segmento* (véase figura 2.5). En efecto, estos registros actúan como una *cache* controlada por el programador que elimina los selectores de la mayor parte de las instrucciones, y permite que la mayoría de las direcciones lógicas se traduzcan en el propio *chip* sin tener que consultar la tabla de descriptors.

Las referencias a direcciones de la mayor parte de los programas se acumulan en unos pocos rangos de direcciones de tamaño pequeño (éste es el principio de "localidad de las referencias" que hace que la memoria virtual re-

sulte práctica). Por ejemplo, si un procedimiento se almacena en un segmento, muchas instrucciones van a ser probablemente buscadas en el propio segmento antes de que el control pase a otro procedimiento en un segmento distinto. El 80386, bajo el control del programa, explota esta "localidad" de las referencias almacenando los selectores y los descriptors usados más recientemente en sus registros del *chip*. Esto permite que la gran mayoría de las direcciones lógicas se traduzcan sin que se produzcan referencias a memoria que consuman demasiado tiempo.

En cualquier instante estarán direccionados seis segmentos como máximo, el *segmento de código (code segment)*, el *segmento de pila (stack segment)*, y a lo sumo cuatro *segmentos de datos (data segments)*. Los registros de segmento CS, SS, DS, ES, FS y GS contienen los selectores para estos segmentos. Los registros de descriptors correspondientes contienen los descriptors adecuados. Si es necesario, un programa puede hacer direccionable un nuevo segmento cargando un nuevo selector de segmento en un registro de

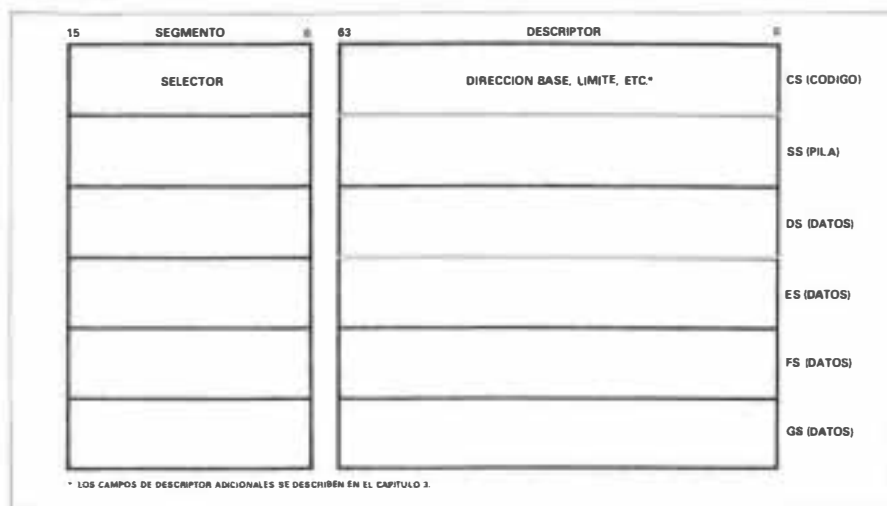


Figura 2.5. Registros de descriptores y de segmentos

segmento. El procesador mantiene los registros de descriptores automáticamente, cargando el descriptor adecuado en el momento en que un programa modifica un registro de segmento. (De hecho, los registros de descriptores sólo pueden ser cargados por el procesador, ya que son innaccesibles para los programas.) Hay que hacer notar que el puntero de instrucciones contiene el desplazamiento de la instrucción actual en el segmento de código en curso (definido por el registro CS), y que el registro ESP contiene el desplazamiento de la cabeza de la pila en el segmento de pila actual (definido por el registro SS).

Para mejorar la eficacia de la codificación de las instrucciones, la mayoría de las instrucciones no nombran los registros de segmento. En lugar de eso, el 80386 selecciona automáticamente un registro de segmento en función de la instrucción que está siendo ejecutada. Por ejemplo, una instrucción Jump (salto) se refiere de forma implícita al registro CS y una instrucción Push (introducir en

la pila) usa el registro SS. Si es necesario, un programador puede dirigir de forma explícita al 80386 para que use un segmento particular en una instrucción, precediendo a ésta con un **prefijo de selección de segmento** de 1 byte. Este prefijo obliga al procesador a usar un registro de segmento particular para la traducción de la dirección en la instrucción siguiente.

Hay que hacer notar que un segmento cuya dirección base es 0 y cuyo límite es de 4 gigabytes, define un espacio lógico de direcciones de 4 gigabytes. Debido a que el procesador selecciona los registros de segmento automáticamente, una instrucción puede nombrar un operando en cualquier parte de este espacio de 4 gigabytes con un único desplazamiento de 32 bytes. Si, como se ilustra en la figura 2.6, todos los registros descriptores son cargados con direcciones de base de 0 y límites de 4 gigabytes, los segmentos desaparecerán. Cada byte del espacio lógico de direcciones, ya se trate de una instrucción, de una variable o de un elemento de la pila, se

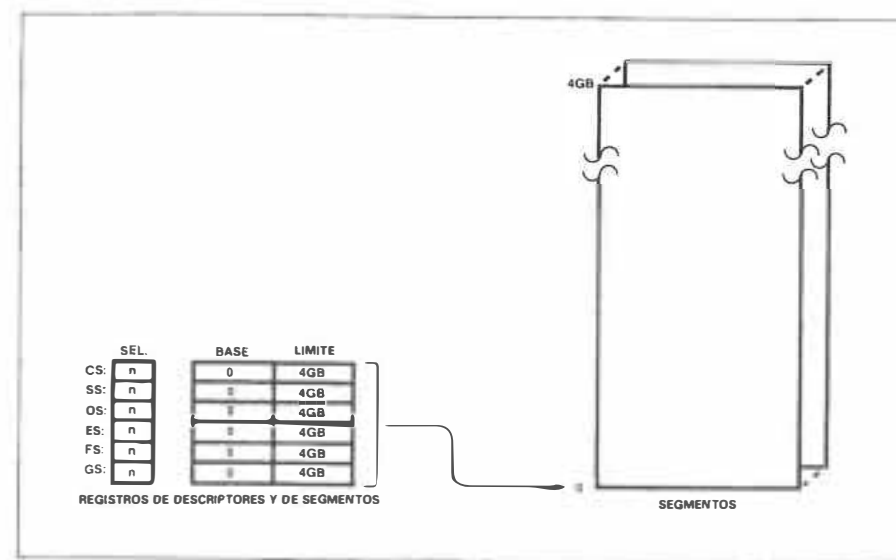


Figura 2.6. Un espacio lógico de direcciones de 4 gigabytes

direcciona con un solo desplazamiento de 32 bits. Así, los registros de segmento proporcionan, al 80386, seis espacios lógicos de direcciones de 4 gigabytes cada uno como máximo, direccionables instantáneamente. Cuando estos segmentos coinciden, un programa ve un solo espacio lógico de direcciones de 4 gigabytes, idéntico al proporcionado por arquitecturas de 32 bits menos flexibles.

2.2.4. Modos de direccionamiento

Son dos los modos de direccionamiento del 80386, el **inmediato** y el de **registro**, según estén localizados los operandos en instrucciones o registros, respectivamente. Es más, el 80386 proporciona los modos de direccionamiento necesarios para referirse eficazmente a los elementos que se encuentran en las estructuras de datos de la memoria, tales co-

mo los *arrays*, registros (estructuras), *arrays* de registros y registros conteniendo *arrays*. Un programa especifica la parte de desplazamiento de una dirección lógica usando uno de los modos de direccionamiento de memoria del 80386. El 80386 calcula la parte de desplazamiento de una dirección lógica mediante la siguiente fórmula:

$$\text{desplazamiento total (offset)} = \text{base} + (\text{índice} \cdot \text{escala}) + \text{desplazamiento}$$

Un desplazamiento puede calcularse usando alguna o todas las variables de **base**, **índice** o **desplazamiento**. Las variables de base e índice son los valores de los registros de uso general, mientras que el valor de desplazamiento está contenido en la instrucción. Cualquier registro de uso general puede servir como registro de base o registro de índice. El valor en el registro de índice puede corresponder a una escala (ir multiplicando por) 1, 2,

Tabla 2.1 Instrucciones y tipos de datos principales

Tipo	Tamaño	Instrucciones
Entero, ordinal	8, 16, 32 bits	Mover, Intercambiar, Trasladar, Comparación lógica, Convertir, Transformar, Desplazar, Desplazamiento doble, Rotar, Not, Negación, And, Or, Exclusive Or, Sumar, Restar, Multiplicar, Dividir, Incrementar, Disminuir, Conversión (Mover con signo/sin extensión)
Decimal sin empaquetar	1 dígito	Ajustar para: Sumar, Restar, Multiplicar, Dividir.
Decimal empaquetado	2 dígitos	Ajustar para: Sumar, Restar
String (byte, palabras, dobles palabras)	0-4 G bytes	Mover, Cargar, Almacenar, Comparar, Explorar, Repetir
Cadena de bits	1-4 G bits	Comparación lógica, Comparación lógica y poner a 0, Comparación lógica y poner a 1, Comparación lógica y Complementar, Explorar, Insertar, Quitar
Puntero próximo*	32 bits	(Similar a los ordinales).
Puntero lejano	48 bits	Cargar

* Un puntero próximo es un desplazamiento de 32 bits en un segmento definido por los registros de segmento y los registros de descriptores. Un puntero lejano es una dirección lógica completa, es decir, un selector y un desplazamiento.

4 u 8, proporcionando un método directo para referirse a registros o a *arrays* de esas longitudes. Un valor de desplazamiento puede tener una longitud de 8 ó 32 bits y el procesador lo interpreta como un valor en complemento a dos con signo.

Las combinaciones más significativas de ba-

se, índice y desplazamiento producen los siguientes modos de direccionamiento de memoria del 80386:

- **Directo:** sólo desplazamiento.
- **Indirecto mediante registro:** sólo base.
- **Base:** base + desplazamiento.
- **Indexado:** índice (escala).
- **Indexado con desplazamiento:** índice (escala) + desplazamiento.
- **Base indexado:** base + índice (escala).
- **Base indexado con desplazamiento:** base + índice (con escala) + desplazamiento.

2.3. Tipos de datos e instrucciones

Esta sección describe las instrucciones que los programadores usan con mayor frecuencia. Como la mayoría de las instrucciones operan sobre tipos de datos específicos (por ejemplo, enteros), tanto éstos como las instrucciones se estudiarán juntos. Las instrucciones prioritarias, incluidas aquellas que ejecutan operaciones de entrada/salida (I/O) y que gestionan las interrupciones, se tratarán en el siguiente capítulo.

2.3.1. Tipos de datos principales

La tabla 2.1 muestra los tipos de datos y las instrucciones del 80386. En esta tabla sólo aparecen las instrucciones de uso más frecuente. También se han omitido variantes de instrucciones tales como (para el caso de la Rotación) Rotar a la Izquierda, Rotar a la Derecha y Rotar a través del *flag* de acarreo (*carry*).

En la figura 2.7 aparecen ejemplos de cómo se almacenan en memoria los tipos básicos de datos. Los elementos de múltiples bytes pueden estar localizados en cualquier dirección de byte. Dependiendo del diseño del *bus*, se podrá necesitar ciclos de *bus* adicional para acceder a un operando localizado en una dirección que no es múltiplo de su tamaño. Por tanto, para conseguir un rendimiento óptimo

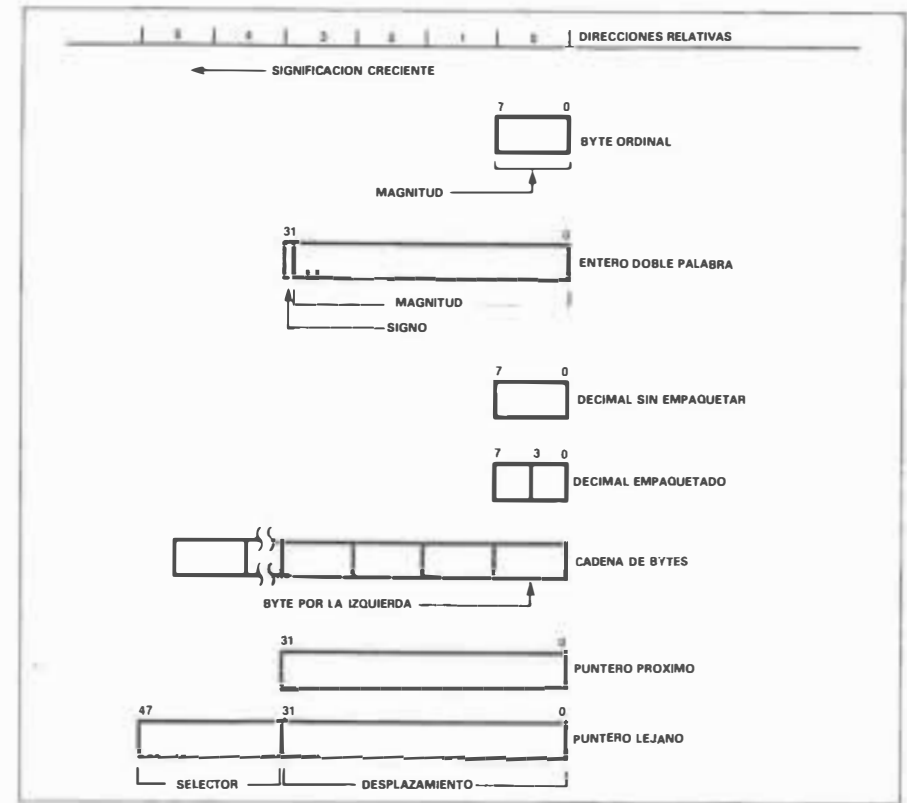


Figura 2.7 Almacenamiento de los diferentes tipos de datos

independientemente del diseño del *bus*, la mayoría de los programas ubican operandos de una palabra dentro de fronteras de una palabra, operandos de dobles palabras dentro de fronteras de dobles palabras, etc.

2.3.2. Tipos de datos del coprocesador numérico

Un coprocesador numérico 80287 u 80387 complementa al 80386 con los tipos de datos

e instrucciones mostrados en la tabla 2.2. La mayoría de las aplicaciones numéricas almacenan valores de entrada y resultados de salida en tipos enteros, reales o decimales empaquetados, y reservan los **reales temporales** para valores intermedios, donde su rango ampliado y su precisión minimizan los problemas de redondeo, desbordamiento por defecto (*underflow*) y desbordamiento (*overflow*), en operaciones complejas. De acuerdo con este modelo, un coprocesador numérico ejecuta la mayoría de las operaciones sobre valores reales

Tabla 2.2 Instrucciones y tipos de datos principales del coprocesador numérico

Tipo	Tamaño	Instrucciones
Entero	16, 32, 64 bits	Cargar, Almacenar, Comparar, Sumar, Restar, Multiplicar, Dividir
Decimal empaquetado	18 dígitos	Cargar, Almacenar
Real	32, 64 bits	Cargar, Almacenar, Comparar, Sumar, Restar, Multiplicar, Dividir
Real temporal	80 bits	Sumar, Restar, Multiplicar, Dividir, Raíz cuadrada, Escalar por una potencia de dos, Resto parcial, Parte entera, Cambiar signo, Valor absoluto, Extraer exponente y mantisa, Comparar, Examinar, Comparación lógica, Tangente, Arco tangente, $2^x - 1$, $Y \cdot \text{Log}_2(X+1)$, $Y \cdot \text{Log}_2(X)$, Constantes (Pi, etc.) (el 80387 suma Seno, Coseno)

temporales almacenados en sus registros. Cualquier tipo de valor que se cargue en la pila de registros se convierte automáticamente en un real temporal. Un valor real temporal almacenado en un registro se puede convertir a cualquier otro tipo mediante la instrucción Store.

La figura 2.8 muestra cómo se almacenan en memoria los tipos de datos numéricos.

2.3.3. Otras instrucciones

No todas las instrucciones del 80386 están relacionadas con los tipos de datos. Los siguientes párrafos examinan las instrucciones que no tienen relación con los tipos.

2.3.3.1. Instrucciones de pila

Una pila del 80386 es una pila de dobles palabras cuya base y cuya cabeza están defini-

das por los registros SS y ESP, respectivamente. La instrucción Push introduce una doble palabra en la pila y la instrucción Pop saca la doble palabra situada en la cabeza de la pila hacia un registro o hacia la memoria. La instrucción Push All introduce los registros de uso general en la pila, mientras que Pop All hace lo contrario.

Las instrucciones Enter (introducir) y Leave (abandonar) están previstas para lenguajes de alto nivel con estructura de bloques. La instrucción Enter construye la estructura de la pila que los compiladores emplean para enlazar (*link*) llamadas de procedimientos. La instrucción Leave elimina esta estructura de la pila como preparación para el retorno al procedimiento que hizo la llamada.

2.3.3.2. Instrucciones de transferencia de control

Las instrucciones Jump (salto) transfieren el control a otra instrucción modificando el valor del puntero de instrucciones. La instrucción de destino puede estar en el mismo segmento de código (como máximo a 2^{12} bytes de distancia) o en uno diferente. El operando de un Jump dentro del mismo segmento es un puntero próximo (*near pointer*), esto es, el desplazamiento de la instrucción de destino dentro del segmento de código en curso; así, un Jump puede ser dirigido a cualquier posición dentro del segmento, sea del tamaño que sea. El operando de un Jump entre segmentos diferentes es un puntero lejano (*far pointer*), que permite que el control pueda ser transferido a cualquier punto en cualquier segmento. (La parte selectora del puntero lejano reemplaza el valor del registro CS, mientras que la parte de desplazamiento reemplaza el valor de EIP) También está disponible un conjunto completo de instrucciones de salto condicional, que se bifurcan o no según el valor de un *flag* de estado; estas instrucciones pueden transferir a posiciones situadas como máximo a 2^{32} bytes de distancia.

Los procedimientos y funciones (subrutinas) pueden ser llamados con la instrucción Call (llamada), y una rutina llamada puede retor-

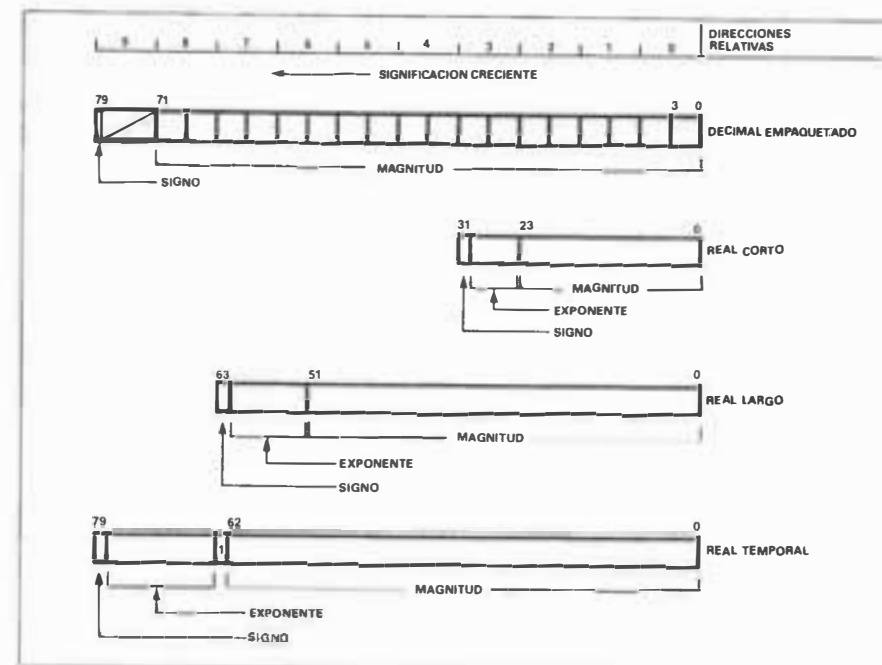


Figura 2.8. Ejemplos de almacenamiento de los diferentes tipos de datos de coprocesador numérico

nar con la instrucción Return. Al igual que los Jump, las llamadas dentro de un segmento de código tienen como operandos punteros próximos que especifican un nuevo valor para el puntero de instrucciones, mientras que las llamadas para un diferente segmento de código tienen como operandos punteros lejanos que cambian el registro CS además del EIP. Las instrucciones Call introducen la dirección de la siguiente instrucción en la pila y luego cargan el puntero de instrucciones (y el registro CS, si la transferencia es hacia un segmento diferente). La instrucción Return saca el valor o los valores guardados de la pila y los carga en EIP, y CS, si corresponde. Las instrucciones Call pueden anidarse y usarse de forma recursiva indefinidamente, sujetas úni-

ca y exclusivamente al tamaño de la pila.

Para controlar bucles, el 80386 cuenta con las instrucciones Loop (bucle) y Loop condicional además de la instrucción Jump (salto) condicional. Las instrucciones Loop usan el registro ECX como contador de repeticiones; estas instrucciones decrementan a ECX y finalizan el bucle cuando el valor del registro es cero. Las instrucciones Loop condicional pueden concluir un bucle prematuramente si un *flag* toma un valor especificado. Mientras que las instrucciones Loop están diseñadas para comprobar el "final del bucle" al final del mismo, añadiendo una instrucción Jump (salto) si ECX es cero, se implementa un test de "final de bucle" al principio del mismo, que permite al bucle ser ejecutado cero veces.

2.3.3.3. Instrucciones diversas

La instrucción Bound (frontera) del 80386 puede usarse para comprobar que el subíndice de un *array* está dentro de las fronteras del *array*. Hay instrucciones para poner *flags* a 0 o a 1, y para cargar y almacenar el byte

de estado del registro de *flags*. El 80287 y el 80387 proporcionan las instrucciones que un sistema operativo necesita para inicializar el coprocesador, manejar las excepciones del mismo y salvar y recuperar su estado. Finalmente, por supuesto, el 80386 tiene una instrucción de No Operación.

3

Arquitectura de sistema

El propósito de la arquitectura de un sistema es soportar sistemas operativos, pero éstos son muy variados respecto a sus necesidades. Es por ello por lo que el 80386 proporciona un conjunto de recursos que los diseñadores e implantadores de sistemas operativos pueden emplear de forma selectiva. En efecto, la arquitectura del sistema del 80386 se puede configurar para adecuar las necesidades del sistema operativo bajo desarrollo.

3.1. Registros del sistema

Además de los registros descritos en el capítulo anterior, un sistema operativo usa a veces los registros del 80386 mostrados en la figura 3.1. (En secciones posteriores de este capítulo se hacen alusiones a estos registros, por lo que los presentamos aquí para que sirva de referencia.) Por lo general, es el 80386 quien usa los registros del sistema; el sistema operativo inicializa los registros del sistema y luego los ignora en el transcurso de la opera-

ción normal. No obstante, el sistema operativo puede usar un registro del sistema para manejar una excepción. Por ejemplo, cuando tiene lugar una falta de página, el procesador carga las direcciones que faltan en CR2, el manipulador de faltas de páginas del sistema operativo usa la dirección para encontrar la entrada a la tabla de páginas asociada. Los registros del sistema son inaccesibles normalmente para los programas de aplicación, ya que sólo las instrucciones prioritarias pueden operar sobre ellos. (Las excepciones, faltas de página e instrucciones prioritarias serán explicadas más adelante en este capítulo.)

3.2. Multitarea

Muchas de las facilidades de la arquitectura de sistema del 80386 soportan directamente sistemas operativos multitarea, aunque, por supuesto, el 80386 se puede usar en aplicaciones unitarea que sean exigentes. La multitarea es una técnica para gestionar el trabajo

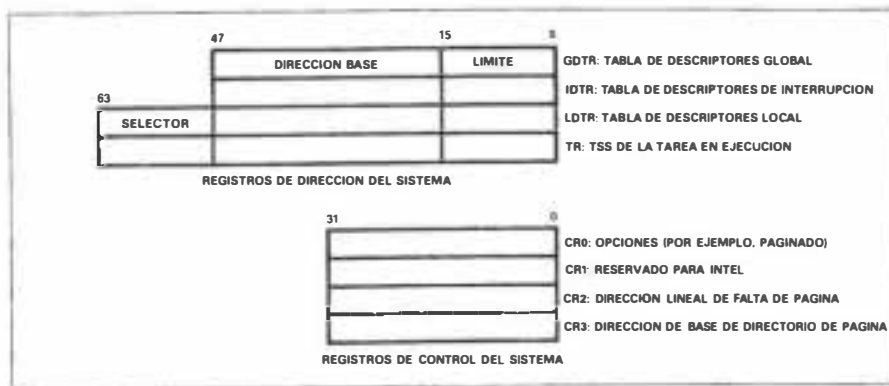


Figura 3.1. Registros del sistema

de un sistema de ordenador cuando ese trabajo consiste en actividades múltiples; tres de estas actividades podrían ser editar un fichero, compilar otro y transmitir un tercero a otro ordenador. En un sistema multitarea, cada actividad que puede realizarse en paralelo con otras actividades se representa por una **tarea** (*task*). (En esta introducción, el término "tarea" se considera equivalente al término "proceso" (*process*).) Cada tarea ejecuta un programa consistente en instrucciones y en valores iniciales de datos. Más de una tarea puede ejecutar el mismo **programa**; por ejemplo, en un sistema multitarea de tiempo compartido, varias tareas (las correspondientes a cada usuario) suelen emplear el mismo compilador o el mismo editor. La relación entre programas y tareas se podría comparar con la existente entre las partituras y la interpretación musical: un programa es un texto que describe un algoritmo, y una tarea es una ejecución de ese algoritmo.

Los programas que las tareas ejecutan están diseñados como si fueran a correr sobre procesadores dedicados que comparten una memoria común; esto es, excepto por pausas ocasionales para comunicar o sincronizarse con otras tareas, una tarea se ejecuta teóricamente de forma continua en paralelo con to-

das las demás tareas. De hecho, no obstante, las tareas se ejecutan en rodajas de tiempo sobre un solo procesador.

El sistema operativo multitarea simula a los procesadores múltiples asignando a cada tarea un "procesador virtual". En cualquier instante, el sistema operativo asigna el procesador real a alguno de los procesadores virtuales, ejecutando por tanto la tarea asociada. Para mantener la ilusión de que hay un procesador por tarea, el sistema operativo conmuta con frecuencia el procesador real con un procesador virtual diferente. La arquitectura del sistema del 80386 soporta esta operación crítica de conmutación de tareas mediante los Segmentos de Estado de Tarea (*Task State Segments*) y mediante instrucciones que conmutan tareas.

3.2.1. Segmento de Estado de Tarea

Un Segmento de Estado de Tarea (TSS) es una de las estructuras de datos definidas por la arquitectura de sistema del 80386. En efecto, estas estructuras de datos son "tipos de datos" para el sistema operativo. Un TSS (véase figura 3.2) se corresponde con lo que algunos

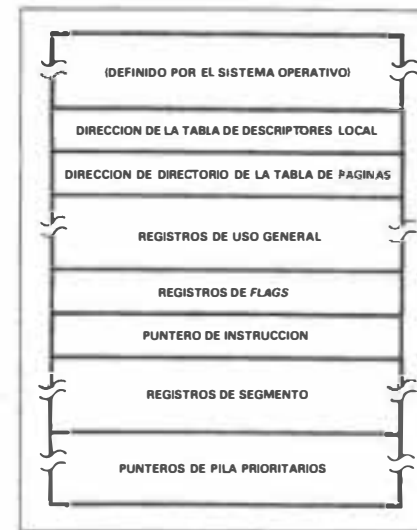


Figura 3.2. Campos principales del Segmento de Estado de Tarea

sistemas operativos llaman un bloque de control de tarea; mantiene el estado del procesador virtual de la tarea. Cada tarea del 80386 se representa por un TSS, que se divide en dos partes. La parte inferior del TSS está definida por la arquitectura del 80386 y contiene los valores de los registros del procesador. La parte superior puede estar definida por el sistema operativo para mantener los datos relativos a la tarea, tales como la prioridad de asignación, los descriptores de fichero, etc. Para crear una nueva tarea, el sistema operativo crea un TSS y lo inicializa con los valores que debería tener al comenzar su ejecución. El 80386 mantiene entonces la parte inferior del TSS, mientras que el sistema operativo se responsabiliza de la parte superior.

3.2.2. Conmutación de tareas

El sistema operativo gestiona la ejecución de las tareas en el procesador de acuerdo a una

política de asignación temporal. Esta política fija el orden en que se van a ejecutar las tareas. Debido a que las políticas de asignación de las tareas son tan diversas, el 80386 deja esta elección al sistema operativo. Una vez que el sistema operativo decide ejecutar una nueva tarea, puede dirigir, no obstante, al procesador para que ejecute el núcleo de la conmutación o intercambio de la tarea, denominado a veces el intercambiador de contexto (*context switch*).

El 80386 almacena un selector y un descriptor para el TSS de la tarea que se está ejecutando en su **Registro de Tarea** (*Task Register*), TR. Para conmutar tareas, el sistema operativo ejecuta una instrucción Jump (salto) cuyo operando es un selector para el TSS de la nueva tarea. El procesador ejecuta la instrucción Jump TSS (salto a TSS) almacenando primero sus registros en el TSS actual y cargando después TR con el selector (y su descriptor correspondiente) especificado por la instrucción. Una vez obtenida la dirección del nuevo TSS, el procesador carga sus registros con los valores en la nueva TSS. La ejecución continúa con la instrucción indicada por el puntero de instrucciones de la nueva tarea. Para reanudar más tarde la ejecución de la tarea anterior, el sistema operativo ejecuta un Jump TSS o salto al TSS de la tarea anterior; la ejecución de esta tarea continúa con la instrucción siguiente a la Jump TSS en la que se suspendió la tarea. El cambio de tarea descrito aquí tarda en realizarse 17 microsegundos (a 16 MHz., sin estados de espera).

3.3. Direccionamiento

El espacio físico de direcciones de la mayoría de los ordenadores se organiza como un simple *array* de bytes. Con el desarrollo de las unidades de manejo de memoria (MMU), las arquitecturas de ordenadores empiezan a distinguir entre el espacio físico de direcciones implementado por la memoria *hardware* y el espacio lógico de direcciones que ve el programador. La MMU traduce las direcciones lógicas presentadas por los programas a las di-

recciones físicas que se envían por el *bus*. La mayoría de las arquitecturas consideran el espacio lógico de direcciones de una tarea como una colección de elementos de alguno de los siguientes tipos:

Bytes

El espacio lógico de direcciones consiste en un conjunto de bytes sin ninguna otra estructura (esto se denomina a veces un espacio de dirección "lineal" o "plano"). No se requiere ninguna traducción MMU, ya que una dirección lógica es exactamente equivalente a una dirección física.

Segmentos

El espacio lógico de direcciones se compone de unos pocos o muchos segmentos, cada uno de los cuales compuestos a su vez por un número variable de bytes. Una dirección lógica está dividida en dos partes, un número de segmento y un desplazamiento dentro del segmento. La MMU traduce una dirección lógica a una dirección física.

Páginas

El espacio lógico de direcciones se compone de muchas páginas, formadas cada una de ellas por un número fijo de bytes. Una dirección lógica es un número de página más un despla-

miento dentro de la página. La MMU traduce una dirección lógica a una dirección física.

Segmentos paginados

El espacio lógico de direcciones está formado por segmentos que a su vez lo están por páginas. Una dirección lógica es un número de segmento y un desplazamiento. La MMU traduce la dirección lógica a un número de página y un desplazamiento y traduce luego éstos a una dirección física.

Cada una de estas ideas encajan bien en algunos sistemas, mientras que en otros no lo hacen tan bien. Por ejemplo, la aproximación "lineal" es apropiada para sistemas simples embebidos, mientras que los sistemas que manejan y protegen de forma separada las estructuras de programas individuales se adaptan mejor a la idea segmentada de la memoria. Técnicamente, el 80386 ve la memoria como una colección de segmentos que opcionalmente están paginados. En la práctica, la arquitectura del 80386 soporta sistemas operativos que usen cualquiera de estas cuatro formas de ver la memoria.

3.3.1. Visión general de la traducción de direcciones

La figura 3.3 muestra los fundamentos de la traducción de direcciones lógicas a físicas del 80386. La secuencia de operaciones que aparecen en la figura es fundamental tanto para el direccionamiento como para la protección. Se describe aquí en forma esquemática para establecer claramente una idea global antes de considerar hechos tales como la memo-

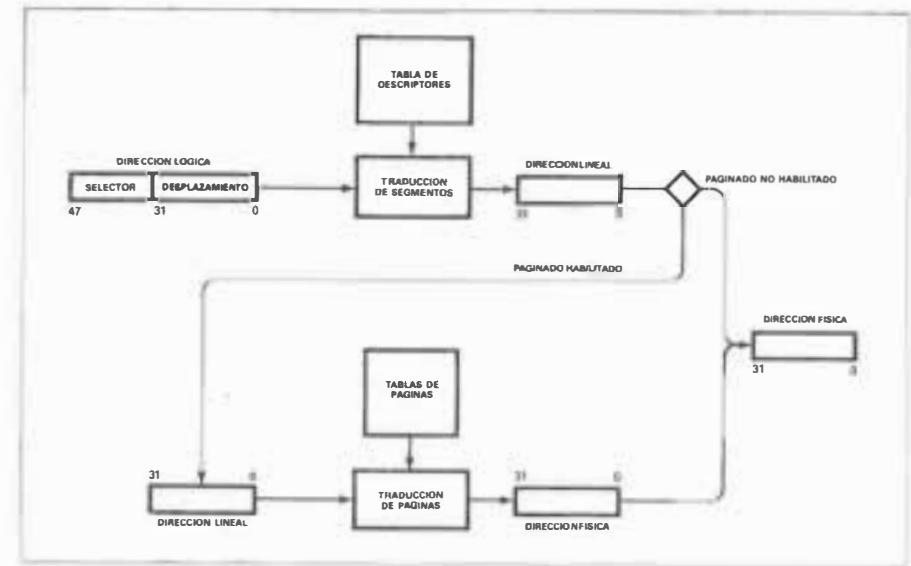


Figura 3.3. Visión general de la traducción de direcciones

ria virtual y la protección. En secciones siguientes se explican con más detalles los distintos pasos de la traducción y se muestra cómo se pueden adaptar para ajustarse a las necesidades de un sistema particular.

Como se describió en el capítulo anterior, los modos de direccionamiento del 80386 producen el desplazamiento de 32 bits del operando destino. Combinado con un selector de segmento, este desplazamiento forma una dirección lógica de dos partes: el selector identifica al segmento destino y el desplazamiento localiza al operando en el segmento. En la gran mayoría de las instrucciones, el selector se especifica de forma implícita como el contenido de un registro de segmento.

Un selector es un índice de la **tabla de descriptores de segmento**; es decir, un número de segmento. Cada entrada a una de estas tablas contiene la dirección base de un segmento. El procesador añade el desplazamiento

to a la dirección base del segmento para producir una dirección lineal de 32 bits. Si el paginado no está habilitado, el procesador considera la dirección lineal como la dirección física y la emite a través de las patillas del *bus* de direcciones.

Si el paginado está habilitado, el 80386 traduce la dirección lineal a una dirección física con ayuda de **tablas de páginas** (*page tables*). El concepto de una tabla de páginas es similar al de una tabla de descriptores excepto en que cada entrada a la tabla de páginas contiene la dirección base física de una **página** de 4 kilobytes.

Debido a que el 80386 incluye las dos unidades tradicionales de estructuración del espacio de direcciones (segmentos y, opcionalmente, páginas), y debido a que los segmentos pueden ser muy grandes (4 gigabytes, como máximo), la técnica de direccionamiento del 80386 es muy flexible. Un sistema opera-

tivo puede proporcionar una tarea con un solo espacio de direcciones plano, un espacio de direcciones plano paginado, un espacio de direcciones segmentado o un espacio de direcciones segmentado y paginado.

A pesar de toda esta flexibilidad, el mecanismo multietapa de traducción de direcciones del 80386 sigue siendo bastante rápido. Por regla general, el 80386 calcula un desplazamiento y traduce la dirección lógica resultante a una dirección física en 1.5 ciclos de reloj. Además, el tiempo de traducción de la dirección es transparente a la aplicación debido a que la MMU residente en el propio *chip* del 80386 traduce las direcciones en paralelo junto con otras actividades del procesador (excepto cuando una instrucción de Salto o de Llamada interrumpe temporalmente la gestión en *pipeline*).

3.3.2. Segmentos

El segmento es la unidad que proporciona el 80386 para definir el espacio lógico de direcciones de una tarea; esto es, el espacio lógico de direcciones de una tarea contiene uno o más segmentos. Los sistemas operativos difieren sustancialmente en la forma en que definen este espacio lógico de direcciones. Por ejemplo, un sistema de tiempo real embebido puede definir este espacio como una única entidad compartida por todas las tareas y por el propio sistema operativo; en otras palabras, un solo segmento es compartido por la totalidad del sistema. En el otro extremo, el caso más complejo, un sistema puede hacer corresponder a cada estructura de datos y a cada procedimiento, un segmento diferente, haciendo que el espacio lógico de direcciones de una tarea contenga decenas o centenares de espacios de direcciones, cada uno correspondiente a un procedimiento o una estructura de datos. Entre estos extremos puede estar un sistema de tiempo compartido de propósito general en el que las tareas se ejecutan en espacios lógicos de direcciones separados, y en el que el código de una tarea está separado de sus datos y, a su vez, el código y los datos de la aplicación están separados del código y da-

tos del sistema operativo. La utilidad de segmentación del 80386 es suficientemente versátil como para soportar cada uno de estos ejemplos y muchos otros más.

Como se describió en el capítulo 2, una instrucción hace referencia a un operando de la memoria mediante una dirección lógica dividida en dos partes: un selector de segmento y un desplazamiento dentro del segmento. En principio, el 80386 traduce la dirección lógica a una dirección lineal usando el selector para buscar el descriptor del segmento en una tabla de descriptors de segmentos. El descriptor contiene la dirección base del segmento en el espacio lineal de direcciones, al sumarle el desplazamiento se obtiene la dirección lineal del operando. En la práctica, la traducción de dirección lógica a física se optimiza mediante selectores implícitos y descriptors basados en registros. Como consecuencia, la consulta a la tabla de descriptors sólo tiene lugar para instrucciones que cargan nuevos selectores en los registros de segmento (por ejemplo, una llamada (Call) a un procedimiento en un segmento diferente cambia el selector contenido en el registro CS).

Aunque rara vez ocurre en la práctica, es muy conveniente pensar en que el procesador traduce las direcciones lógicas consultando los descriptors en las tablas de descriptors de segmento, ya que de ello se deduce que los descriptors de las tablas de descriptors de segmento de una tarea definen el espacio lógico de direcciones de la misma. Sin un descriptor, una tarea no tiene modo de generar una dirección lineal.

Una tabla de descriptors de segmento es un *array* de descriptors; la figura 3.4 muestra el formato lógico de un descriptor. El campo de la dirección base ha sido explicado ya. El campo *límite* especifica la longitud del segmento; el 80386 usa el campo *límite* para comprobar que la parte de desplazamiento de una dirección lógica es válida (que cae realmente dentro del segmento). Los atributos del segmento se refieren principalmente a la protección y serán descritos más adelante.

Cada tarea puede tener una anchura de sistema y un espacio lógico de direcciones privado; estos conceptos se representan mediante

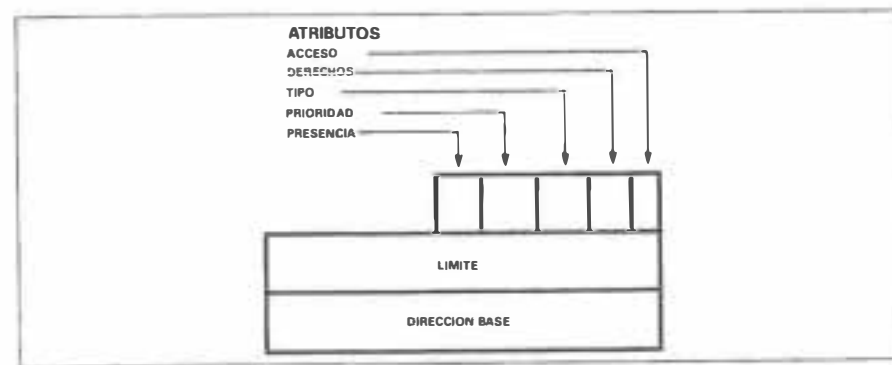


Figura 3.4 Campos principales de un descriptor

la Tabla de Descriptor Global (GDT) y la Tabla de Descriptor Local (LDT), respectivamente. (Cada selector contiene un bit que lo asocia con una u otra tabla.) Estas tablas de descriptors pueden contener a lo sumo 8192 descriptors cada una, y juntas definen el espacio lógico de direcciones de una tarea. Esto es, para hacer que un nuevo segmento sea direccionable por una tarea, el sistema operativo debe insertar un descriptor para ese segmento en la GDT o en la LDT de la tarea. En sistemas protegidos, la GDT y la LDT pueden estar constituidas por estructuras privilegiadas para que sólo el sistema operativo pueda modificarlas.

Como su nombre indica, todas las tareas comparten la Tabla de Descriptor Global; los sistemas operativos normalmente colocan en la GDT los descriptors de los segmentos que son compartidos por todo el sistema. El segmento de código (o segmentos) del sistema operativo es un buen ejemplo de un segmento que debería ser accesible a todas las tareas y cuyo descriptor, por consiguiente, está localizado habitualmente en la GDT. Por contraste, cada tarea puede tener su propia Tabla de Descriptor Local (LDT). El 80386 mantiene la dirección de la LDT de la tarea en curso en su Registro de la Tabla de Descriptor Local (LDTR), pero vuelve a recargar este regis-

tro (como lo hace con los registros de uso general y los registros de segmento) desde el TSS de la nueva tarea, al producirse la conmutación de tareas.

Las tareas pueden compartir un segmento de tres formas diferentes (véase figura 3.5):

1. Un segmento cuyo descriptor está en la GDT es compartido por todas las tareas.
2. Las tareas que comparten una LDT, comparten también los segmentos descritos en dicha LDT; este planteamiento es apropiado para tareas que cooperan estrechamente.
3. Los descriptors de diferentes LDT pueden apuntar al mismo segmento; estos descriptors se denominan *alias*. Los "alias" permiten que las tareas puedan compartir un segmento individual, en lugar de tener que compartir todos los segmentos de una tabla de descriptor.

3.3.3. Páginas

Si un espacio lógico de direcciones de una tarea se compone de uno o más segmentos, un sistema operativo puede subdividir el espacio lineal de direcciones en páginas. Para un sistema operativo, las páginas son unidades

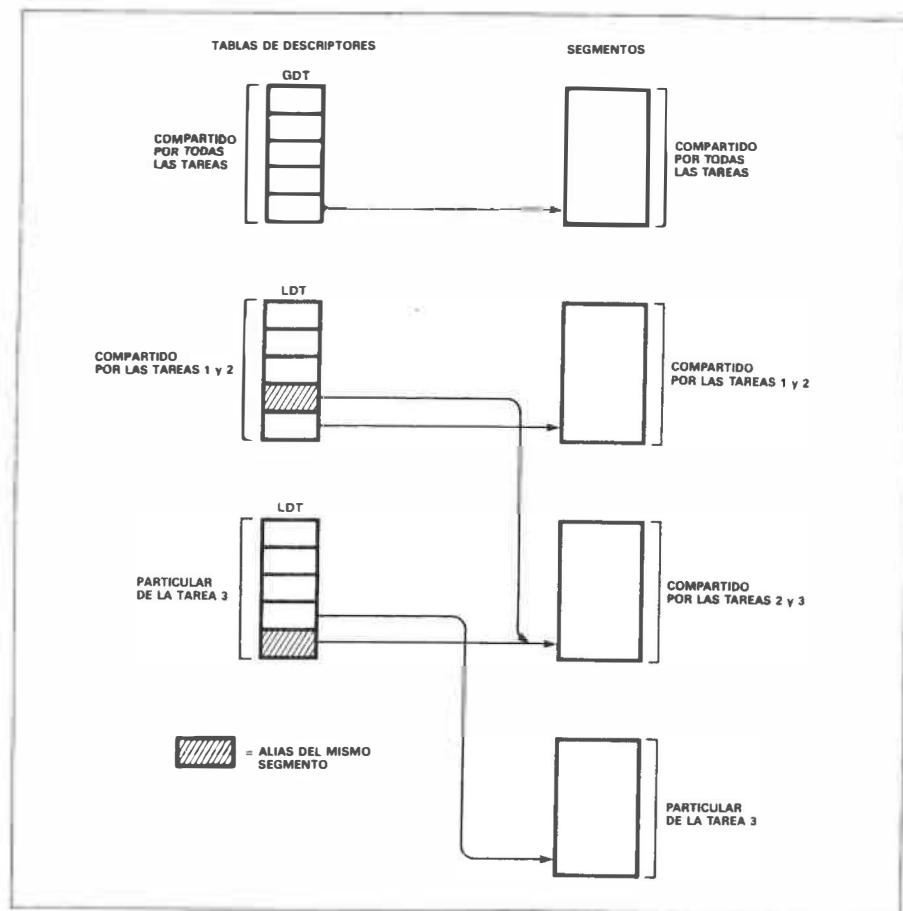


Figura 35. Diferentes maneras de compartir segmentos

prácticas para la asignación y la reubicación, debido a que todas ellas poseen el mismo tamaño. Las páginas también proporcionan un método de protección para determinadas partes de segmentos grandes y, lo que es muy importante, proporcionan una unidad adecuada para la implementación de la memoria virtual.

Estas aplicaciones del paginado serán estudiadas en las siguientes secciones.

Una página del 80386 tiene un tamaño de 4K bytes. Este tamaño está en consonancia con la tendencia en la industria hacia páginas más grandes, y ayuda a conseguir un mejor rendimiento de dos formas. Primero, propor-

ciona una tasa de aciertos de página en *cache* bastante alta, dando el tamaño de la memoria *cache* que puede incluirse razonablemente en el propio *chip* con la tecnología actual. (La *cache* de página del 80386 se describe brevemente.) Segundo, los 4K bytes son una unidad eficiente para la transferencia a/y desde el disco; la mayoría de los sistemas operativos que corren sobre máquinas con tamaños de página más pequeños deben agrupar las páginas en "racimos", para mantener aceptablemente bajo el número de transferencias al disco.

Un sistema operativo basado en el 80386 habilita el paginado poniendo a 1 el bit PG (Habilitar paginado) del registro de control 0 con una instrucción prioritaria. Cuando el paginado se habilita, el procesador traduce una dirección lineal a una dirección física con ayuda de tablas de páginas. Estas tablas son equivalentes a las tablas de descriptores de seg-

mento; lo mismo que la tabla de descriptores de segmento de una tarea define su espacio lógico de direcciones, la tabla de páginas de una tarea define su espacio físico de direcciones. De forma similar a los superminiordenadores y los grandes *mainframes*, las tablas de páginas del 80386 correspondientes a una tarea se disponen de acuerdo a una jerarquía de dos niveles, como se indica en la figura 3.6. Cada tarea puede tener su propio directorio de tabla de páginas. El registro de sistema del 80386 denominado CR3 (Base de Directorio de Tablas de Páginas) apunta al directorio de tablas de páginas de la tarea en curso; el procesador pone al día el CR3 en cada cambio de tarea, obteniéndose la nueva dirección del directorio a partir del nuevo TSS de la nueva tarea. Un directorio de tablas de páginas tiene a su vez una longitud de una página, y contiene entradas para 1024 tablas de pági-

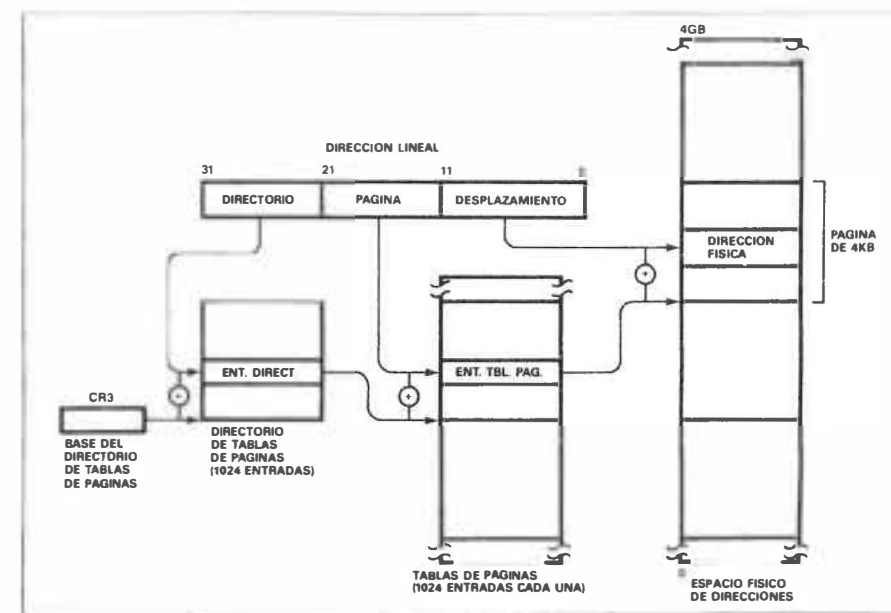


Figura 36. Traducción de dirección lineal a dirección física

nas como máximo. Las tablas de páginas tienen también una longitud de una página, y las entradas describen 1024 páginas. Así, cada tabla de páginas se corresponde con 4 megabytes, y cada directorio puede corresponderse a lo sumo con 4 gigabytes, que sería el espacio físico de direcciones de 32 bits completo.

La figura 3.6 muestra en términos funcionales cómo el 80386 traduce una dirección lineal a una dirección física cuando el paginado está habilitado. El procesador usa los 10 bits superiores de la dirección lineal como un índice dentro del directorio. La entrada del directorio seleccionada contiene la dirección de una tabla de páginas. El procesador suma los 10 bits centrales de la dirección lineal a la dirección de la tabla de páginas, para obtener un índice a la entrada de la tabla de páginas que describe la página objetivo. Sumando los 12 bits inferiores de la dirección lineal a la dirección de la página se obtiene la dirección física de 32 bits.

Para evitar un exceso de consultas a la tabla de páginas, el 80386 guarda la información correspondiente a las 32 páginas usadas más recientemente en un *buffer de consulta para la traducción* (TLB). Sólo cuando no se encuentra la información de mapeado de una página en el TLB, el procesador consulta un

directorio o tabla de páginas con base en la memoria. Como regla, de un 98 a un 99 por 100 de las referencias a direcciones “aciertan” (la página se encuentra en el TLB), no requiriéndose referencias a memoria para la traducción. Cuando ocurre una “falta de página” (la página no se encuentra en el TLB), el procesador reemplaza una entrada más antigua del TLB por la nueva; el principio de localidad de las referencias sugiere que la nueva entrada será probablemente usada de nuevo en un futuro cercano.

Aunque la habilitación del paginado no incrementa el tiempo de traducción de las direcciones, sí hace que el tiempo de ejecución de la instrucción varíe ligeramente debido a las ocasionales faltas de página del TLB. Inhabilitando el paginado, los sistemas en tiempo real pueden eliminar esta posible respuesta variable en el tiempo.

La figura 3.7 muestra el contenido básico de una entrada de la tabla de páginas (PTE). Las entradas al directorio son idénticas, excepto en que el campo de dirección de la página se interpreta como la dirección física de una tabla de páginas, y no como una página.

Las tareas pueden compartir páginas individuales o tablas de páginas en su totalidad. Las entradas en distintas tablas de páginas que

apuntan a la misma página son “alias” una de la otra, al igual que los descriptors con la misma dirección base. La estructura en dos niveles de la tabla de páginas del 80386 hace más fácil que se compartan páginas entre tareas, compartiendo tablas de páginas enteras. Como la dirección de una página compartida de esta manera existe en una única tabla de páginas, el sistema operativo tiene que actualizar sólo una entrada en una tabla de páginas cuando se mueve la página.

3.3.4. Memoria virtual

La memoria virtual hace posible que los programas muy grandes, o los grupos de programas, se puedan ejecutar en las cantidades más pequeñas de memoria física sin que sea preciso escribir un código específico por parte del programador. Los sistemas de memoria virtual pueden estar basados en segmentos o en páginas.

En cualquier caso, la idea básica de la memoria virtual es la de explotar el coste mucho menor del almacenamiento en disco comparado con el de la memoria de semiconductores. Un sistema operativo de memoria virtual almacena todos los segmentos o todas las páginas en un área grande del disco, denominada con frecuencia área de intercambio (*swap*). La memoria física (“real”), mucho más pequeña, mantiene sólo las páginas o los segmentos usados más frecuentemente. Mientras sean usados con poca frecuencia los segmentos y las páginas almacenadas en el disco, un sistema de memoria virtual tendrá un rendimiento casi igual que otro con mucha más memoria, y con un coste menor. Las características arquitectónicas necesarias para soportar con eficacia la memoria virtual son:

- Un bit por cada segmento o página que diga al procesador (o a la unidad de manejo de memoria) si el segmento o la página está “presente” en la memoria o necesita ser intercambiada desde el disco.
- Un mecanismo de excepción o de interrupción, por el que un procesador pue-

da notificar al sistema operativo la necesidad de intercambiar un segmento o página no presente.

- Instrucciones reinicializables que permitan al procesador volver a intentar la ejecución de una instrucción después de que el sistema operativo ha cargado la página anteriormente ausente en la memoria, y de que la ha marcado como presente.

El 80386 tiene todas estas capacidades necesarias, además de otras que mejoran la eficacia de la gestión de la memoria virtual. Tanto los descriptors como las entradas de la tabla de páginas tienen un bit de presencia, y por consiguiente se pueden emplear como bases para el diseño de un sistema de memoria virtual. El intercambio de segmentos entre la memoria y el diseño es un planteamiento razonable cuando éstos son pequeños, como ocurre en arquitecturas de 16 bits. Sin embargo, cuando el tamaño de los segmentos puede ser muy grande, como puede ocurrir en el 80386, el intercambio de páginas es un tratamiento usualmente más efectivo, debido al tamaño fijo de las páginas. En un sistema basado en páginas, el sistema operativo asigna y libera memoria en unidades de tamaño de una página, llamadas *marcos de página*; una página intercambiada desde el disco encajará en cualquier marco disponible. Como la mayor parte de los sistemas de memoria virtual de 32 bits se basan en páginas, el resto de esta sección describirá el soporte de memoria virtual paginada del 80386.

En general, un sistema operativo de memoria virtual paginada transfiere páginas ausentes desde el disco a los marcos de página según se vayan pidiendo, esto es, el procesador notifica que una instrucción se refiere a una página ausente. Cuando el número de páginas libres disminuye, el sistema operativo también transfiere páginas desde los marcos de página al disco, tratando de eliminar las páginas que probablemente van a ser las menos referidas en un futuro cercano. Mediante el intercambio transparente de páginas entre los marcos de página y el disco, el sistema operativo da al *software* de la aplicación la ilusión de una memoria física tan grande como el área de in-

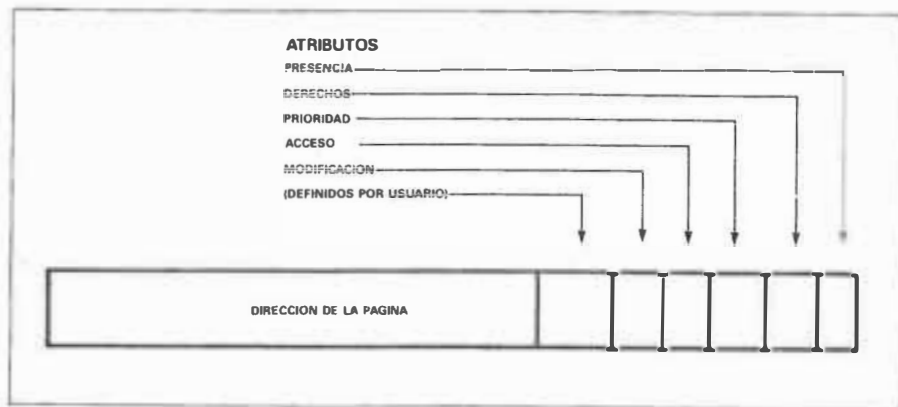


Figura 3.7. Campos principales de una entrada de una tabla de páginas

tercambio (*swap*) del disco. Los detalles de estas operaciones se describen a continuación.

Cuando en el transcurso de la traducción de una dirección lógica, el procesador produce una dirección lineal referida a una entrada de la tabla de páginas cuyo bit presente está a cero, el procesador causa una excepción denominada **falta de página**. Las excepciones se abordarán más tarde en este capítulo, pero la consecuencia básica de una falta de página es que el procesador llama a un procedimiento del sistema operativo que se conoce como **gestor de faltas de página**. Cuando se entra en el gestor de faltas de páginas, el registro de control 2 contiene la dirección lineal asociada con la página ausente. Desde esta dirección el gestor de faltas de páginas puede encontrar la entrada relevante a la tabla de páginas, traduciendo la dirección lineal justamente como lo hizo el procesador. Hay que hacer notar que todos los bits de una entrada ausente de la tabla de páginas, salvo el bit de presencia, son definidos por el usuario; estos bits proporcionan un lugar conveniente para que el sistema operativo almacene la dirección en el disco de la página ausente. Una vez determinada la dirección en el disco de la página ausente, el gestor de faltas de página puede asignar un marco de página y transferir la página del disco al marco. Después de actualizar el campo de dirección y el bit de presencia de la entrada de la tabla de páginas, el gestor de faltas de página simplemente retorna. Entonces el procesador revisa automáticamente la instrucción que falta, y el resultado es el mismo que si la página hubiese estado presente cuando la instrucción fue ejecutada por primera vez.

Otros campos de una entrada de la tabla de páginas del 80386 ayudan al sistema operativo a ejecutar con eficacia operaciones con la memoria virtual. Además de cargar las páginas cuando se vayan pidiendo, el sistema operativo debe mantener un suministro de marcos de página libres que puedan ser asignados por el gestor de faltas de página. Para aumentar este suministro, el sistema operativo debe decidir qué marcos van a estar libres. Antes de liberar un marco, el sistema operativo debe escribir también la página en el disco si ésta ha sido modificada desde su carga.

Para ayudar al sistema operativo en estas actividades, la arquitectura del 80386 proporciona un **bit de acceso** y un **bit de modificación** en cada entrada de la tabla de páginas; el procesador actualiza estos bits para todas las páginas presentes de forma automática. El 80386 pone a 1 el bit de acceso en el instante en que una página es leída o escrita y pone a 1 el bit de modificación cuando se escribe en la página. Mediante un examen periódico y poniendo los bits a 0, el sistema operativo puede identificar las páginas que no han sido usadas recientemente. Los marcos que contienen estas páginas son buenos candidatos para ser liberados, ya que es muy frecuente que las páginas que no han sido usadas recientemente tampoco lo sean en un futuro próximo. Cuando el sistema operativo ha seleccionado una página para que ésta abandone su marco de página, esta página no se escribirá en el disco a menos que el procesador haya puesto a 1 su bit de modificación.

Cada entrada de la tabla de páginas contiene también un campo de 3 bits que el sistema operativo usa como prefiere. Comúnmente este campo se emplea para marcar las páginas con condiciones especiales de estado, tales como "bloqueado para entrada/salida", etc.

3.4. Protección

El 80386 proporciona un conjunto de mecanismos de protección que los sistemas operativos pueden emplear de forma selectiva para cubrir sus necesidades. Una forma de protección ha sido estudiada ya: la separación de los espacios de direcciones de las tareas mediante tablas de descriptores de segmento y tablas de páginas. Esta separación evita de una forma eficaz que los códigos y los datos se interfieran unos a otros. Además de aislar las tareas entre sí, el 80386 proporciona utilidades para proteger el sistema operativo del código de la aplicación, para proteger cada parte del sistema operativo de las restantes y para proteger una tarea de algunos de sus propios errores. Además de hacer que los sistemas operativos sean más robustos, el sistema de protección del 80386 puede simplificar la

labor de depuración de errores atrapando y aislando los mismos a tareas concretas. Todas las utilidades de protección están implementadas en el propio *chip* de forma que el chequeo de la protección se pueda realizar sin penalizaciones en el rendimiento.

3.4.1. Prioridades

Muchas de las utilidades de protección del 80386 se basan en la noción de una **jerarquía de prioridades**. En cualquier instante, una prioridad de una tarea es igual al nivel de **prioridad** del segmento de código en ejecución. En cada descriptor de segmento hay un campo que define el nivel de prioridad del segmento asociado; el campo puede tomar uno de los cuatro valores siguientes: 0, 1, 2 y 3. El nivel de prioridad 0 es el más prioritario y el nivel de prioridad 3 es el de prioridad menor.

La figura 3.8 muestra cómo pueden emplearse los niveles de prioridad para establecer diferentes criterios de protección. Un sistema desprotegido se puede implementar situando simplemente todos los procedimientos en un segmento (o en varios) cuyo nivel de prioridad sea 0. La distinción tradicional entre usuario y supervisor se puede implementar situando el código (aplicación) del usuario en un segmento con nivel de prioridad 3, y los procedimientos del supervisor en un segmento con nivel de prioridad 0. Un sistema operativo puede usar, si lo desea, los niveles 1 y 2. Por ejemplo, a los procedimientos del sistema operativo más importantes y que menos cambios sufran (denominados a veces el núcleo del sistema operativo) se les debe asignar el nivel de prioridad 0. El nivel de prioridad 1 debe emplearse para aquellos servicios menos críticos y que puedan sufrir modificaciones, o más frecuentemente, por ejemplo, los controladores de dispositivos. El nivel 2 podría reservarse para uso de fabricantes de equipos originales (OEM, del término inglés *Original Equipment Manufacturers*). Los OEM podrían asignar a su código el nivel 2, dejando el 3 para los usuarios. De esta forma, el *software* del OEM se protege de los



Figura 3.8. Uso de niveles de prioridad

usuarios, el sistema operativo se protege del OEM y de los usuarios, y el núcleo del sistema operativo del resto del *software*, incluida la parte del sistema operativo sujeta a cambios frecuentes.

Como se ampliará en sucesivas secciones, un nivel de prioridad de una tarea determina qué instrucciones puede ejecutar y a qué subconjunto de segmentos y/o páginas de su espacio de direcciones puede referirse. El procesador comprueba la consistencia entre el nivel de prioridad de una tarea y el del segmento o página objetivo de una instrucción. Cualquier intento por parte de una tarea de usar un segmento o página con un nivel de prioridad mayor hace que el procesador pare la ejecución de la instrucción y levante una excepción de protección general. (Las excepciones serán analizadas más tarde en el capítulo, al igual que las llamadas al sistema, que proporcionan un método controlado para que un procedimiento con menor nivel de prioridad pueda llamar a otro con prioridad mayor.)

3.4.2. Instrucciones prioritarias

El nivel de prioridad de una tarea define las instrucciones que puede ejecutar además de los segmentos y páginas a los que tiene acceso. El 80386 tiene un cierto número de instrucciones cuya ejecución debe estar estrechamente controlada para evitar fallos graves del sistema. Todas las instrucciones que cargan nuevos valores en los registros del sistema son ejemplos de **instrucciones prioritarias**. Sólo una tarea con nivel de prioridad 0 puede ejecutar instrucciones prioritarias.

3.4.3. Protección de segmento

Los descriptores en la LDT de una tarea y en la GDT definen el espacio lógico de direcciones de una tarea. Los segmentos definidos en estas tablas son teóricamente direccionables, ya que las tablas de descriptores proporcionan la información necesaria para calcular la dirección de un segmento. Sin embargo, un segmento direccionable puede no ser accesible a una operación particular debido a los chequeos de protección realizados por

el 80386. El 80386 comprueba cada referencia al segmento (bien generada por la ejecución de una instrucción o por la búsqueda de la instrucción) para verificar si la referencia es consistente con los atributos de protección del segmento, tal como se describe a continuación.

Prioridad Para acceder a un segmento, un programa debe tener al menos un nivel de prioridad igual al del segmento. Por ejemplo, un programa ejecutándose con nivel 3 puede hacer sólo referencias a segmentos cuyo nivel de prioridad sea 3, mientras que un programa en el nivel 0 puede tener acceso a todos los segmentos del espacio lógico de direcciones.

Límite Una referencia a un segmento debe estar comprendida dentro de los límites del segmento. Los límites del segmento permiten al procesador detectar errores comunes de programación tales como el desbordamiento de la pila, punteros y subíndices erróneos y llamadas y saltos a direcciones incorrectos. En los casos en que el sistema operativo puede determinar que una referencia externa a las fronteras de un segmento no es un error (el desbordamiento de la pila es un ejemplo en algunos sistemas), el sistema operativo puede ampliar el segmento (por ejemplo, añadiendo una página) y comenzar de nuevo la instrucción.

Tipo Cada descriptor contiene un campo de tipo que el procesador comprueba para que sea consistente con la instrucción que está ejecutando. Los segmentos ordinarios tienen un tipo de código o de datos, posibilitando que el procesador detecte los posibles intentos de escribir encima del

código; por ejemplo, los segmentos que las aplicaciones manipulan directamente son de tipo de código o de datos. Los descriptores del sistema tienen también tipos para que el procesador pueda comprobar cuándo se están intercambiando tareas; por ejemplo, que el segmento llamado por una instrucción *Jump TSS* sea de hecho un Segmento de Estado de Tarea (*Task State Segment*).

Derechos Un descriptor de segmento puede dotarse de derechos que restrinjan las operaciones permitidas sobre el segmento asociado. Los segmentos de códigos pueden marcarse sólo como ejecutables o bien como ejecutables y leeables, mientras que los segmentos de datos pueden marcarse como de sólo lectura o como de lectura y de escritura.

Todas las comprobaciones descritas anteriormente dependen de la integridad de los descriptores. Si una tarea que ejecuta su código de aplicación pudiera cambiar un descriptor, las comprobaciones no garantizarían nada. Por esta razón, un sistema operativo puede restringir el acceso a las tablas de descriptores únicamente a los códigos con nivel de prioridad 0.

Hay que hacer notar que en el caso de segmentos compartidos, los diferentes descriptores del mismo segmento (esto es, los "alias") pueden tener diferentes atributos de protección, permitiendo por ejemplo, que una tarea pueda leer y escribir un segmento, mientras que otra pueda sólo leerlo. Los "alias" permiten también al sistema operativo pasar por alto el sistema de protección cuando sea necesario, por ejemplo, para mover un segmento de código.

3.4.4. Protección de páginas

Los sistemas que no hagan un uso extensivo de los segmentos pueden en su lugar pro-

teger páginas. (La protección de páginas también se puede aplicar a secciones de segmentos grandes.) Al igual que un descriptor, cada entrada de una tabla de páginas tiene un conjunto de atributos de protección; el 80386 comprueba para cada una de las referencias a la página la conformidad con esos atributos.

Una entrada de una tabla de páginas puede estar marcada con uno de dos niveles posibles de prioridad, **usuario** o **supervisor**. El nivel de usuario corresponde al nivel de prioridad 3, mientras que las páginas de supervisor sólo pueden ser accedidas desde los niveles de prioridad 0, 1 ó 2. Una página de usuario puede también calificarse como de sólo lectura o de lectura y escritura.

El 80386 comprueba los atributos de protección de una página después de verificar que un acceso es consistente con los atributos del segmento. Así, la protección de una página es un modo conveniente para que el sistema operativo aplique protección adicional a determinadas partes de un segmento. Por ejemplo, un sistema operativo puede almacenar fácilmente los datos del sistema operativo relativos a una tarea, en un segmento de datos de una tarea convirtiendo las páginas que los contienen en páginas de supervisor.

3.5. Llamadas al sistema

La mayoría de los sistemas operativos organizan sus servicios en forma de una colección de procedimientos que las tareas pueden llamar. Un sistema operativo 80386 sin protección puede situar sus procedimientos y código de aplicación en un segmento de código de nivel 0 (o en más de uno); una tarea de aplicación puede entonces invocar un servicio del sistema operativo mediante una instrucción (llamada) *Call* corriente. Esta aproximación es más rápida pero depende de que las tareas de aplicación estén libres de errores y funcionen correctamente (como ocurre en los sistemas embebidos, por ejemplo). Nada puede evitar que una tarea ejecutándose en un nivel 0 haga una llamada a una dirección que no corresponde a un punto de entrada del sistema operativo; ni tampoco es posible impedir a una

tarea tal que corrompa los datos del propio sistema operativo. Para proteger el sistema operativo, los datos y el código de la aplicación pueden ponerse en segmentos con prioridad menor. Una tarea que se ejecuta en un nivel de prioridad dado no puede leer o escribir en un segmento o página de datos de un nivel de mayor prioridad, ni tampoco llamar directamente a un segmento de código de mayor nivel.

Para permitir que una tarea que ejecuta un segmento de código con menor nivel de prioridad haga una llamada al sistema protegido, el sistema operativo debe definir uno o más puntos de entrada. En el 80386 estos puntos de entrada se llaman **puertas** o **gates** (véase la figura 3.9). Existen dos tipos de puertas que pueden usarse para implementar los puntos de entrada del sistema operativo, las **puertas trap** (*trap gates*) y las **puertas de llamada** (*call gates*). En general, las dos puertas son similares, pero la puerta de llamada hace que el interfaz del sistema operativo se comporte como lo haría un procedimiento ordinario. Usando las puertas de llamada, los compiladores y los programadores en lenguaje ensamblador pueden usar un solo conjunto de convenios para llamar a cualquier procedimiento, dejando que el 80386 se ocupe del procesamiento extra necesario para cambiar los niveles de prioridad.

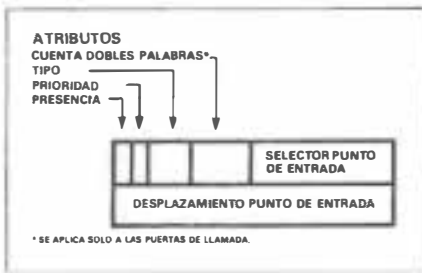


Figura 3.9 Campos principales de puertas

Como se muestra en la figura 3.9, una puerta contiene la dirección lógica de un punto de

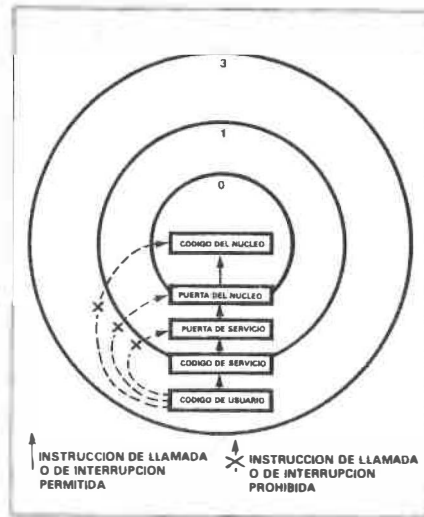


Figura 3.10. Las puertas como puntos de entrada protegidos

entrada y un conjunto de atributos. El atributo más importante es el nivel de prioridad de la puerta. Un nivel de prioridad de una puerta define los niveles de prioridad que pueden usar dicha puerta; para usar una puerta, un procedimiento que la llame debe tener al menos una prioridad igual a la de la puerta. La figura 3.10 muestra un ejemplo. En este sistema hipotético, el código de usuario tiene asignado un nivel de prioridad 3 mientras que el sistema operativo está dividido en dos niveles. El núcleo (*kernel*) del sistema operativo corre en el nivel de prioridad 0, y los procedimientos de servicio menos críticos del sistema operativo, en el 1. (El nivel de prioridad 2 no se emplea.) En este sistema el código del usuario puede llamar a los procedimientos de servicio del nivel 1, pero no al núcleo; los procedimientos de servicio pueden, no obstante, llamar al núcleo. De igual manera, el sistema operativo proporciona una puerta a los procedimientos de servicio; el nivel de prioridad de esta puerta es 3 para que el código del

usuario pueda llamarlos directamente. Asignando el nivel de prioridad 1 a la puerta del núcleo, el sistema operativo permite que los procedimientos de servicio puedan llamar al núcleo, pero niega el acceso al código del usuario, que está a un nivel menor. Así, un sistema operativo puede emplear puertas para definir de forma precisa sus puntos de entrada, incluyendo los niveles de prioridad que se requieren para usar un punto de entrada. En orden a conseguir que todos los procedimientos de servicio puedan ser llamados por todas las tareas, el sistema operativo coloca normalmente su puerta (o puertas) en la Tabla de Descriptor Global.

Para llamar directamente a una puerta *trap*, una tarea ejecuta una instrucción de Interrupción, y para llamar a una puerta de llamada, ejecuta una instrucción ordinaria de Llamada entre segmentos diferentes. Ambas instrucciones cambian el nivel de prioridad de la tarea, y cambian a la pila definida (en el TSS de la tarea) para el nivel de prioridad mayor. (El sistema operativo debe tener su propia pila, para garantizar que tenga suficiente espacio de pila para ejecutarse; no se puede confiar sin temor a equivocarse en que las tareas de aplicación hayan dejado espacio de pila suficiente.)

Antes de llamar a través de una puerta de llamada, una tarea puede introducir parámetros en su pila, como lo haría antes de llamar a cualquier otro procedimiento. El 80386 copia automáticamente los parámetros en una pila con mayor prioridad (el campo "cuenta de dobles palabras" en la puerta de llamada le dice al 80386 cuántas dobles palabras de parámetros deben copiarse). Los sistemas que llaman directamente a través de las puertas *trap* pueden pasar los parámetros en registros.

3.6. Interrupciones y excepciones

Los dispositivos generan interrupciones cuando precisan atención, mientras que las instrucciones pueden incurrir en excepciones cuando su ejecución encuentra una con-

dición especial, como una falta de página. Una interrupción o excepción típica requiere una invocación rápida a un gestor de *software* que responde a la interrupción o a la excepción. Cuando el gestor retorna, el 80386 reasume la ejecución de la cadena de instrucciones que fue interrumpida por la excepción. El 80386 trata las interrupciones y excepciones de una manera unificada, dada su similitud fundamental.

Cada fuente de interrupciones y cada tipo de excepción tienen un número de identificación en el rango 0-255; el 80386 usa este número para invocar al gestor asociado con la interrupción o excepción. En la tabla 3.1 se muestran los números de excepciones definidos por el 80386. Los números de interrupciones son definidos por el sistema operativo. Este inicializa un controlador de interrupciones programable 8259A para que cada fuente de interrupciones esté asociada con un número. Cuando tiene lugar una interrupción, el 8259A proporciona al 80386 el número de la interrupción. Las instrucciones de Interrupción facilitan sus números en sus operandos. Tenga en cuenta que para conseguir la compatibilidad con los productos de Intel actuales y futuros, los números 0-31 de las interrupciones y/o excepciones no deben usarse salvo tal y como se definen en la tabla 3.1. Todos los demás números se pueden usar libremente.

3.6.1. Tabla de Descriptores de Interrupciones

Al generar o al obtener un número de excepción o de interrupción, el 80386 usa éste como un índice a la **Tabla de Descriptores de Interrupciones**, o IDT (del término inglés, *Interrupt Descriptor Table*). La IDT puede estar localizada en cualquier lugar de la memoria; el sistema operativo inicializa la IDT y carga sus direcciones en el **Registro de Tabla de Descriptores de Interrupciones (IDTR)**. Como la GDT o una LDT, la IDT es un vector de descriptores, aunque las puertas son el único tipo de descriptores permitidos en la IDT. Hay una puerta para cada gestor de excepcio-

Tabla 3.1 Excepciones

ID	Descripción
0	Error de división
1	Excepción de depuración
3	Breakpoint del software
4	Desbordamiento
5	Chequea la frontera del array
6	Código de operación inválido
7	Coprocesador no presente
8	Falta doble
10	TSS inválido
11	Error de segmento
12	Desbordamiento de pila
13	Violación de la protección general
14	Falta de página
16	Error del coprocesador

nes o de interrupciones. (Funcionalmente, la IDT es similar a la “tabla de vectores de interrupción” proporcionada por la mayoría de las arquitecturas.)

Un gestor de interrupción o de excepción del 80386 puede estar implementado como un procedimiento o como una tarea; las cualidades de estas dos alternativas se analizarán brevemente. El 80386 llama a un gestor basado en procedimientos más o menos como si ejecutara una llamada al sistema a través de puertas. Para llamar a un gestor basado en tareas, el 80386 ejecuta un cambio de tarea. Una puerta tipo IIT de un gestor dice al procesador cómo invocar al gestor (véase tabla 3.2). Como se ha mencionado, las puertas trap y de interrupciones son funcionalmente similares a las puertas de llamada, excepto en que no tienen la posibilidad de copia de parámetros, y también hacen que el 80386 guarde el registro de flags en la pila del gestor. Se diferencian sólo en el estado del Flag de Habilitación de Interrupciones (Interrupt Enable Flag, o IF) en el momento de la entrada al ges-

tor; cuando se entra a un gestor de interrupciones éstas están inhabilitadas, mientras que a un gestor de trap, que se usa típicamente para manejar excepciones, se entra con las interrupciones tal como estaban. Cuando entra a un gestor basado en tareas, el 80386 carga el registro de flags con los valores guardados en el TSS de la tarea, permitiendo que el gestor funcione con interrupciones habilitadas o inhabilitadas.

Los gestores basados en procedimientos son apropiados para rutinas que se deberían ejecutar en el contexto (esto es, usar el espacio de direcciones y los valores de los registros) de la tarea que es interrumpida o que incurre en la excepción. A 16 MHz, la secuencia de invocación requiere 3.6 microsegundos. Como cualquier otro, un procedimiento de excepción o interrupción tiene acceso a todos los recursos de la tarea en ejecución —sus datos y su código, sus registros y su pila—. Así es como debería ser para la mayoría de las excepciones, ya que una tarea produce una excepción y es posible que se requiera el acceso a los datos para resolverla. Por ejemplo, un gestor de faltas de página necesita las tablas de páginas de la tarea en ejecución para encontrar la dirección de disco de la página no presente.

En el caso ideal, las interrupciones deberían ser manejadas por tareas y no por procedimientos, debido a que una interrupción, por regla general, no está relacionada con la tarea que lo interrumpe. Además, un gestor de interrupciones debería tener sus propios recursos (por ejemplo, su propia pila) en lugar de “heredar” los de la tarea que se está ejecu-

Tabla 3.2 Características de la puerta de excepción y de la puerta de interrupción

Tipo de puerta	Gestor	Interrupciones
Interrupción	Procedimiento	Inhabilitadas
Trap	Procedimiento	Habilitadas
Tarea	Tarea	(Flag IF del gestor)

tando cuando la interrupción tiene lugar, cualquiera que sea. Por otra parte, un intercambio de tarea consume un tiempo mayor que una llamada a un procedimiento (17 frente a 3.6 microsegundos) debido a que el procesador salva y recupera sus registros cuando conmuta tareas. Los sistemas que son extremadamente sensibles a interrupciones pueden manipular las interrupciones mediante procedimientos.

3.6.2. Excepciones y registros de depuración

Como la mayoría de los procesadores, el 80386 tiene una instrucción de breakpoint que puede usarse para llamar a un depurador cuando se ejecuta. El soporte principal de depuración del 80386 toma la forma de los registros de depuración que se muestran en la figura 3.11. Los registros de depuración soportan tanto breakpoints de instrucciones como breakpoints de datos. Los breakpoints de datos son una importante innovación que puede evitar horas en el tiempo de depuración, determinando con precisión, por ejem-

plo, cuándo una estructura de datos se está sobrescribiendo. Los registros de breakpoints eliminan también las complicaciones que se requieren para escribir una instrucción de breakpoint en un código que está protegido contra escritura o compartido por múltiples tareas.

Un depurador para el 80386 se implementa como el gestor para la excepción número 1. Se puede orientar al procesador para que llame al depurador después de cada instrucción (poniendo a 1 el flag TF, el Flag de Trap de ejecución paso a paso, cuando se produzcan determinados intercambios de tareas, o cuando tenga lugar la condición de breakpoint definida en uno de los registros de depuración. Inspeccionando el Registro de Estado de Depuración, el gestor de excepción de depuración puede determinar cuál fue la causa de haber sido llamado. El depurador puede volver a recargar los registros de depuración con los valores correspondientes a la nueva tarea.

El 80386 puede controlar a lo sumo cuatro condiciones de breakpoint de forma simultánea, llamando al gestor de excepción de depuración cuando tiene lugar alguna de estas condiciones. Cada condición de breakpoint se define por el contenido de un registro de depuración; estos registros pueden ser cargados o almacenados con formas prioritarias de la instrucción Move. Una condición de breakpoint se compone de una dirección lineal de 32 bits, un campo con una longitud de 2 bits y un campo de acceso; los últimos dos términos se especifican en los campos de DR7, el Registro de Control de Depuración. La dirección y longitud de una condición de breakpoint forman un rango de dirección que el procesador comprueba en cada referencia a memoria. El campo de acceso define el tipo de acceso para el que el procesador va a señalar la excepción 1. Se pueden especificar tres tipos de acceso:

1. Instrucción en la dirección ejecutada.
2. Escritura de datos en un rango de direcciones.
3. Lectura o escritura de datos en un rango de direcciones.



Figura 3.11 Registros de depuración

Compatibilidad de la arquitectura

3.7. Entrada/Salida

Un sistema basado en el 80386 puede mapear dispositivos de entrada/salida (I/O) en el espacio de memoria del procesador o en un espacio I/O separado. Los dispositivos de I/O mapeados en memoria pueden ser leídos o escritos usando instrucciones de referencia a memoria tales como *Move*, *Or*, etc. Los dispositivos mapeados en memoria se pueden proteger con los mecanismos estándar del 80386 de protección de páginas y segmentos.

Además de su espacio de direcciones de memoria, el 80386 tiene un espacio de direcciones de I/O de 64 kilobytes. Los dispositivos mapeados en este espacio se manipulan con las instrucciones *Input* (Entrada), *Output* (Salida), *Input String* (Entrada de cadenas) y *Output String* (Salida de cadenas). Las dos primeras instrucciones transfieren un byte, palabra o doble palabra a/o desde el registro *EAX*, mientras que las dos últimas transfieren una cadena o *string* de bytes, palabras o dobles palabras a/o desde memoria.

Las instrucciones de I/O del 80386 son sensibles al nivel de prioridad. En el registro de *flags* hay un campo llamado *Nivel de prioridad de I/O* (IOPL), que define el nivel de prioridad mínimo en el que la tarea en ejecución puede ejecutar instrucciones de I/O. (El IOPL es cargado desde el TSS de manera que

puedan existir diferentes IOPL.) Por ejemplo, si el IOPL de una tarea es 1, la tarea no puede ejecutar la instrucción de I/O excepto cuando se ejecuta con un nivel de prioridad 1 ó 0. El mecanismo IOPL soporta sistemas operativos con múltiples niveles de protección en los que, por ejemplo, los procedimientos correspondientes a la parte esencial del sistema operativo y que son estables y críticos corren en el nivel 0, mientras que los procedimientos de I/O sujetos a cambios y modificaciones están en el nivel 1; en este caso el sistema operativo sólo tiene que poner a 1 el IOPL cuando crea una tarea. Como el IOPL es específico de una tarea, a las tareas de confianza se les puede permitir ejecutar instrucciones de I/O mientras está ejecutándose el código de la aplicación, permitiéndoles, por ejemplo, manejar directamente dispositivos especiales, para los que no hay disponible ningún controlador propio del sistema operativo.

Para llevar a cabo accesos I/O directos a memoria (DMA, del inglés *Direct Memory Access*), un sistema operativo pasa una dirección física al controlador DMA y debe garantizar que el segmento (o segmentos) y/o la página (o páginas) objetivo no se muevan durante la transferencia. Una forma de marcar páginas "bloqueadas para I/O" es usar uno de los tres bits de la tabla de páginas definidos por el usuario.

El 80386 es compatible a nivel de código objeto con el 80286 y con el 8086. Aunque sea posible usar el 80386 simplemente como un rápido 80286 o un muy rápido 8086, sus facilidades de compatibilidad son sustancialmente más versátiles. El 80386 puede ejecutar de forma concurrente programas del 80386 y del 80286, y usando el Modo Virtual 86 del 80386, los programas existentes del 8086 también pueden ejecutarse de forma concurrente. Así pues, con el 80386 es posible construir sistemas en los que se puede ejecutar de forma concurrente *software* escrito para las tres generaciones de la familia 86 de microprocesadores de Intel.

4.1. Compatibilidad con el 80286

La arquitectura del 80286 es un subconjunto seleccionado de la arquitectura del 80386. Debido a que el 80386 reconoce todas las ins-

trucciones del 80286, además de los registros, descriptores, etc., un sistema operativo basado en el 80286 y los programas de aplicación pueden ser portados a un *hardware* basado en el 80386 sin cambiar un solo bit.

El llevarlo directamente de uno a otro sistema, como se describe anteriormente, es la forma más rápida de poner a ejecutar el *software* existente del 80286 sobre un sistema basado en el 80386. De forma alternativa, un sistema operativo del 80386 puede ser diseñado para soportar aplicaciones existentes para el 80286, mientras que al mismo tiempo permite que las nuevas aplicaciones usen los recursos de la arquitectura del 80386 (por ejemplo, los segmentos de gran tamaño y los parámetros de 32 bits). En un diseño híbrido tal, las nuevas aplicaciones llaman al sistema operativo directamente, a través de los parámetros de 32 bits. Las llamadas de las aplicaciones antiguas, que están en el formato de 16 bits del 80286, son interceptadas y convertidas al formato de 32 bits y entonces se pasan al sistema operativo.

4.2. Modos Real y Virtual 86

El 80386 puede ejecutar el código objeto del 8086 en uno de dos modos, **Modo Real** o **Modo Virtual 86**. El 80386 entra en Modo Real cuando se inicializa mediante un *reset*. En Modo Real, el procesador proporciona una rápida ejecución en un entorno sin protección como el del 8086. La mayoría de los sistemas operativos cambiarán del Modo Real al Modo Protegido después de la inicialización, pero también es posible ejecutar el *software* del 8086 en Modo Real. La principal diferencia entre el Modo Real del 80386 y un 8086 real es la velocidad: los programas del 8086 que dependen en gran medida de la velocidad (por ejemplo, aquellos que usan bucles para temporización) pueden necesitar algunas pequeñas modificaciones para ejecutarse de forma adecuada sobre el Modo Real del 80386, mucho más rápido. La gran mayoría de los programas del 8086, sin embargo, correrán sin dificultades, tal como lo harían sobre un Modo Real del 80286.

El Modo Virtual 86 establece un entorno de ejecución del 8086 dentro del entorno multitarea protegido del 80386. Donde el Modo Real gobierna todas las cosas que hace el procesador, el Modo Virtual 86 se puede aplicar para seleccionar tareas del 80386. Cuando se ejecuta una tarea en Modo Virtual 86, el procesador se comporta como un 8086, pero cuando conmuta a una tarea normal, el procesador opera como un 80386 (el cual, por supuesto, puede interpretar tanto programas del 80286 como del 80386). Así, el Modo Virtual 86 capacita a un sistema operativo para soportar la ejecución de forma concurrente de programas del 8086, 80286 y 80386.

El capítulo 3 describía cómo un segmento de estado de tarea de una tarea representa el estado de su procesador virtual. El *flag VM86* en el registro de *flags*, que se carga desde el TSS, define el procesador virtual de una tarea en ejecución como un 8086 o un 80386. Cuando el 80386 carga sus registros desde un TSS cuyo *flag VM86* está puesto, el procesador entra en Modo Virtual 86. Cuando, en el cambio de tarea posterior, el procesador carga los valores de los registros desde un TSS

cuyo *flag VM86* está a cero, abandona el Modo Virtual 86. Así, siguiendo un criterio de tarea a tarea, el procesador va emulando a un 80386 o a un 8086, de acuerdo con el valor del *flag VM86*. El 80386 abandona también el Modo Virtual 86 cuando señala una excepción o cuando es interrumpido, haciendo que la totalidad de los recursos de la arquitectura estén disponibles a los controladores de excepción y de interrupción. Cuando se vuelve desde un controlador al que se ha invocado en el Modo Virtual 86, el 80386 vuelve a entrar automáticamente en el Modo Virtual 86. Debido a que el espacio de direcciones de un 8086 es de un megabyte, la dirección lógica generada por una tarea Modo Virtual 86 cae en el primer megabyte del espacio lineal de direcciones. Las múltiples tareas Modo Virtual 86 podrían interferir unas con otras, ya que todas deberían compartir el megabyte bajo el espacio lineal de direcciones. Un sistema operativo puede usar el paginado del 80386 para colocar de nuevo los espacios lineales de direcciones de las tareas Modo Virtual 86 en diferentes áreas del espacio físico de direcciones. Usando el paginado de esta forma no sólo se evitan las interferencias entre las tareas del Modo Virtual 86, sino que se capacita a un sistema operativo de memoria virtual para intercambiar las páginas de las tareas del Modo Virtual 86 como si se trataran de tareas del 80386.

Una tarea Modo Virtual 86 puede ejecutar un programa escrito para ser ejecutado sobre un ordenador personal monotarea. Tal programa puede contener instrucciones que son potencialmente peligrosas si se ejecutan en un entorno multitarea. Por ejemplo, si se permite que una tarea Modo Virtual 86 ejecute una instrucción de puesta a 0 del *flag* de interrupciones, inhabilitando de esta forma las interrupciones, podría llevar a una caída del sistema. Para evitar tales disrupciones, el 80386 señala una excepción cuando una tarea Modo Virtual 86 trata de ejecutar una instrucción I/O o una instrucción relativa a una interrupción.

Evitando la ejecución de tales instrucciones se puede proteger al resto del sistema operativo de una tarea Modo Virtual 86, pero no se

satisface la necesidad de la tarea Modo Virtual 86 de ejecutar las instrucciones. La solución es simular las instrucciones sensibles en un procedimiento del sistema operativo llamado **monitor de máquina virtual**. Cuando se invoca a un manipulador de excepciones, éste puede inspeccionar el *flag VM86* en la imagen de los *flags* que se mantienen en la pila para ver si la fuente de la excepción es una tarea Modo Virtual 86; en ese caso, el manipulador de excepciones puede llamar al monitor de máquina virtual, que puede simular la instrucción y retornar a la tarea Modo Virtual 86. Nótese que un monitor de máquina virtual simula sólo unas cuantas instrucciones del 80386 y que tanto las instrucciones simuladas como aquellas que el 80386 ejecuta directamente se benefician de las prestaciones más altas del 80386 comparado con el 8086.

Trabajando juntos, el 80386 y un monitor de máquina virtual implementan el repertorio completo de instrucciones del 80386, y el paginado puede proporcionar a cada tarea en Modo Virtual 86 su propio espacio de direcciones protegido. De cualquier forma, la mayoría de los programas del 8086 necesitan

recursos adicionales proporcionados por un sistema operativo y por *hardware* periférico. Un ejemplo del primer tipo de recursos es un sistema de ficheros; un ejemplo del segundo es un controlador de vídeo de mapa de bits, manejado directamente por un programa de aplicación. Puede que estos recursos no existan de la misma forma en el sistema basado en un 80386 que en el sistema para el que los programas del 8086 fueron diseñados. Para simplificar el trabajo de proporcionar estos recursos en un entorno diferente, el 80386 puede interrumpir las referencias al sistema operativo y a los periféricos hechas por tareas Modo Virtual 86.

Por ejemplo, la mayoría de los sistemas operativos usan la instrucción de Interrupción para implementar las llamadas al sistema operativo. El 80386 indica una excepción cuando una tarea Modo Virtual 86 intenta ejecutar una instrucción de Interrupción. Entonces el monitor de máquina virtual puede traducir la llamada al sistema operativo 8086 a una llamada al sistema operativo del 80386, tal como se muestra en la figura 4.1. Si el IOPL (nivel de prioridad I/O) de una tarea Modo

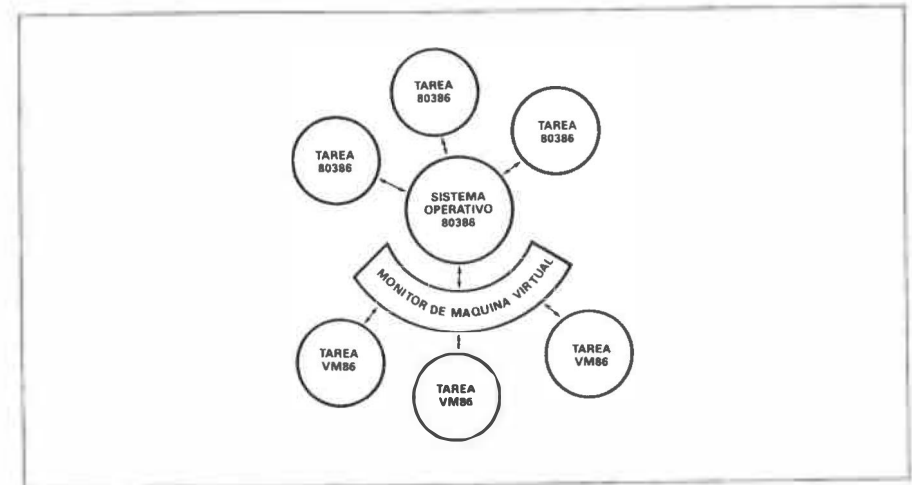


Figura 4.1. "Atrapamiento" de las llamadas al sistema en Modo Virtual 86

Virtual 86 se pone a un valor menor que 3, el 80386 atrapar  cualquier instrucci n de entrada/salida que el programa ejecute. La facilidad de paginado del 80386 puede usarse para redirigir las referencias a perif ricos mapeados en memoria a otras direcciones, si fue-

ra necesario. Tales referencias pueden ser "atrapadas" tambi n marcando las p ginas correspondientes como de s lo lectura (para atrapar las escrituras), o como ausentes (para atrapar tanto las lecturas como las escrituras).

5

Implementaci n hardware

La arquitectura del 80386, ya descrita en los cap tulos anteriores, se implementa sobre 275 000 transistores usando la tecnolog a CHMOS III de Intel. Este cap tulo mira brevemente en el interior del *chip* 80386, y examina con m s detalle las se ales que el 80386 y otros componentes usan para comunicarse.

5.1. Dise o interno

La figura 5.1 presenta una visi n de conjunto de las unidades funcionales que componen el 80386. Estas seis unidades est n dispuestas en un *pipeline*, lo que les permite operar en paralelo sobre instrucciones diferentes, o sobre partes distintas de la misma instrucci n. La **unidad de bus** realiza las transacciones para las otras unidades. Cuando ninguna otra unidad necesita el *bus*, la **unidad de preb squeda** lee la siguiente doble palabra de la secuencia de instrucciones residente en la memoria en la cola de preb squeda. De este modo, la mayor a de las b squedas de c digo son

ejecutadas en paralelo con la ejecuci n aprovechando ciclos de *bus* no utilizados. La **unidad de decodificaci n** descompone cada c digo de operaci n, convirti ndolo en un puntero al microc digo que implementa la instrucci n. La **unidad de ejecuci n** ejecuta las microinstrucciones. La unidad de ejecuci n puede sumar dos registros de 32 bits en 2 pulsos de reloj. El *hardware* multiplicador/divisor realiza multiplicaciones de 32 bits entre 9 y 41 pulsos de reloj, dependiendo del n mero de cifras significativas, y divisiones de 32 bits entre 38 y 42 pulsos, dependiendo de si los operandos tienen o no signo. Las instrucciones Desplazamiento, Rotaci n y las que operan sobre un solo bit, son auxiliadas por un desplazador de tambor que puede desplazar hasta 64 bits en un solo pulso de reloj. En las combinaciones t picas de instrucciones, que incluyen llamadas y saltos, el 80386 ejecuta las instrucciones a una velocidad promedio de 4.4 pulsos de reloj cada una.

La conexi n en *pipeline* de las unidades de b squeda, decodificaci n y ejecuci n de la ins-

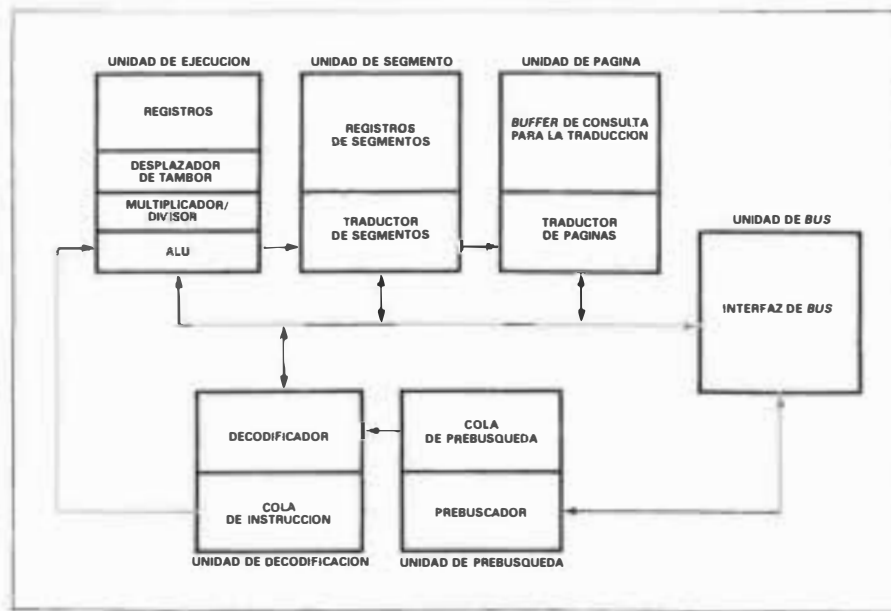


Figura 5.1 Unidades funcionales

trucción, en el propio *chip*, no es inusual en los microprocesadores modernos. Por otra parte, la colocación de la unidad de manejo de memoria (MMU) en *pipeline* sí que lo es. La incorporación de la MMU en el *chip* aumenta la velocidad de traducción de direcciones reduciendo el retraso en la propagación de la señal (la mayoría de las MMU incorporadas fuera del *chip* introducen al menos un estado de espera), y explotando las señales con los semiciclos de reloj que están disponibles en el *chip* (la entrada de reloj del 80386 es dos veces la frecuencia del *chip*). La unidad MMU se compone de unidades de página y de segmento, como se muestra en la figura 5.1.

La unidad de segmento traduce las direcciones lógicas a direcciones lineales, y comprueba cada acceso para que esté en consonancia con los atributos de protección del seg-

mento. Para la mayoría de las instrucciones, la unidad de segmento obtiene los datos de traducción y protección del segmento a partir de los registros de descriptors y de segmentos en el propio *chip* del 80386. La **unidad de página** es habilitada o inhabilitada por el *software* del sistema operativo. Cuando se inhabilita, las direcciones lineales producidas por la unidad de segmento pasan inalteradas a través de la unidad de página. Cuando el paginado está habilitado, la unidad de página traduce las direcciones lineales a direcciones físicas, y verifica que los accesos son consistentes con los atributos de la página. La unidad de página incluye un **buffer de consulta para la traducción** (TLB) de 32 entradas que recoge en memoria la información de traducción de las páginas usadas más recientemente. Usando el TLB, la unidad de página

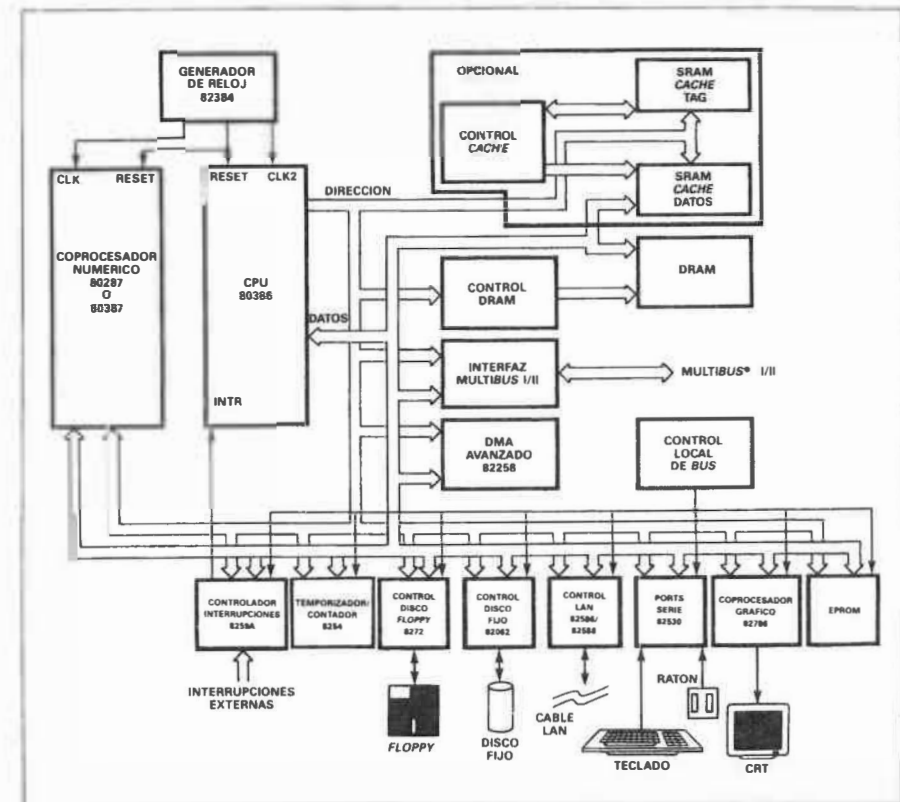


Figura 5.2 Diagrama de bloques de un sistema representativo

puede traducir la mayoría de los accesos a páginas (normalmente el 98-99 por 100) sin consultar las tablas de páginas con base en memoria. Cuando es necesario, la unidad de página inicia los ciclos de *bus* que se requieren para retornar una entrada del TLB antigua a su tabla de páginas y para cargar el hueco vacante del TLB con la entrada de la tabla de páginas referida por la instrucción en curso.

5.2. Interfaz externo

La figura 5.2 es un diagrama de bloques que muestra el 80386 en un sistema representativo, una estación de trabajo para ingeniería.

La figura 5.3 presenta el interfaz externo del 80386 con más detalle, agrupando las patillas en grupos relacionados funcionalmente. Las

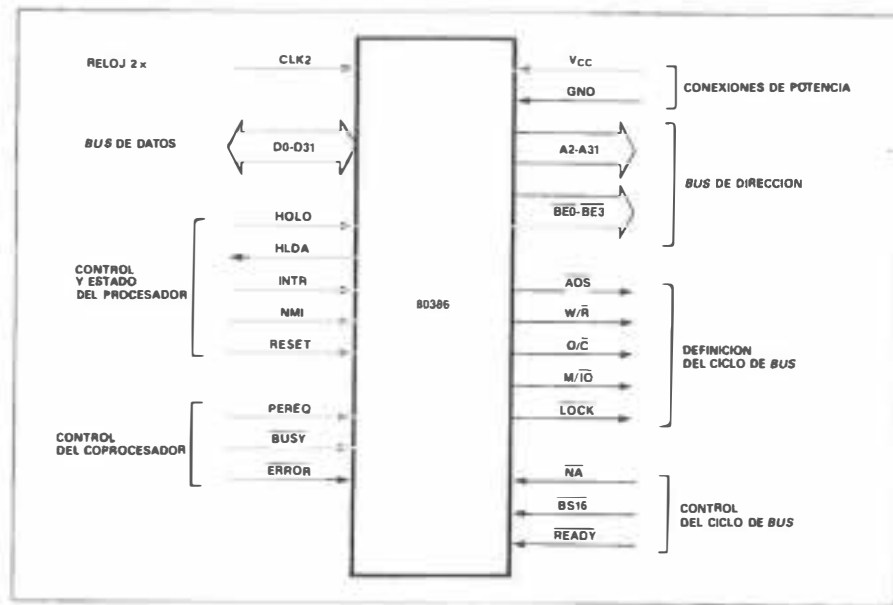


Figura 5.3. Entradas y salidas funcionales

memoria de 32 bits y eliminan la necesidad de un *hardware* de decodificación de byte (véase figura 5.4). Cuando es necesario, por ejemplo, para conectar a un *bus* de sistema que requiere los bits de dirección de orden más bajo, A0 y A1 pueden ser generados a partir de BE0-BE3 con cuatro puertas.

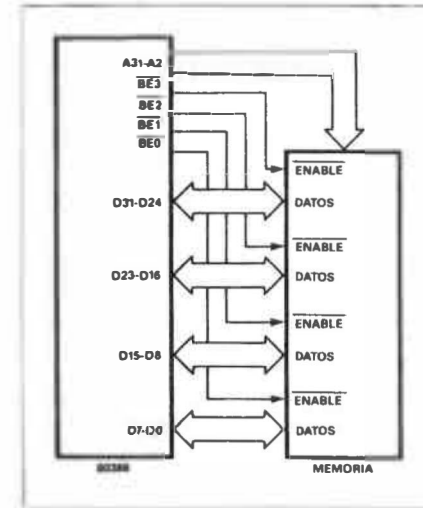


Figura 5.4. Uso de los byte enable

Los operandos en la memoria del 80386 no necesitan ser alineados, pero el rendimiento es mejor cuando caen en zonas divisibles exactamente por su tamaño en bytes. Esto es, las palabras están mejor localizadas en direcciones que son divisibles por 2, y las dobles palabras, en aquellas divisibles por 4. (Los términos mayores que 32 bits, tales como los números de punto flotante de doble precisión, deberían estar alineados también en direcciones divisibles por 4 para un rendimiento mejor.) El 80386 ejecuta automáticamente múltiples ciclos de *bus* para transferir operandos no alineados; por ejemplo, un entero de doble palabra almacenado en una dirección par

no divisible por 4 se transfiere a dos ciclos de *bus* de 16 bits.

5.2.3. Definición de ciclo de *bus*

El 80386 avisa al *hardware* externo de que un ciclo de *bus* normal está comenzando, manteniendo el *ADS* (*Address Status* o estado de la dirección). Al mismo tiempo, el procesador define el tipo de ciclo de *bus* con las señales *W/R*, *D/C* y *M/I/O*. Estas señales distinguen entre escritura frente a lectura, datos frente a código y memoria frente a accesos I/O, respectivamente.

El 80386 proporciona la señal *LOCK* (*Bus Lock* o bloqueo del *bus*) para los diseños multimaestros y multiprocesadores. La señal dice a los otros maestros o "amos" del *bus* que el procesador está ejecutando una operación de ciclo de *bus* múltiple que no debe ser interrumpida. El 80386 mantiene a 1 automáticamente la señal del *LOCK* cuando actualiza las tablas de páginas y de descriptors de segmento, durante los ciclos de *bus* de reconocimiento de interrupciones, y cuando ejecuta la instrucción Intercambio. Esta instrucción proporciona la operación indivisible de "chequear y poner a 1" (*test and set*), que constituye el elemento básico crucial para la implementación de semáforos para la gestión de memoria compartida. Los programadores en lenguaje ensamblador pueden bloquear el *bus* durante la ejecución de otras instrucciones precediendo a tales instrucciones con prefijos *Lock*.

5.2.4. Control del ciclo de *bus*

Bajo la dirección de *hardware* externo, el 80386 puede ejecutar dos tipos de ciclos de *bus*, en *no-pipeline* y en *pipeline*. El primero está diseñado para proporcionar accesos de dos ciclos de reloj a la *caché* de alta velocidad y a las memorias locales de cualquier tamaño. (La efectividad de una memoria *caché* depende de su tamaño en relación con las características de referencia de la aplicación.) El segundo proporciona a los sistemas de memo-

siguientes secciones describen las señales asociadas a estas patillas.

5.2.1. Reloj

Las primeras versiones del 80386 funcionan a 12.5 ó 16 MHz y son controladas por una señal de reloj (CLK2) que es dos veces la frecuencia del chip. Un generador de reloj 82384 proporciona la señal CLK2, que el 80386 divide en dos para obtener su reloj interno.

5.2.2. Buses de datos y de dirección

El 80386 tiene *buses* de 32 bits separados de direcciones y de datos. Para que exista com-

patibilidad con el *hardware* actual y los controladores o *drivers* de dispositivos, el ancho efectivo del *bus* de datos debe ser cambiado dinámicamente entre 16 y 32 bits. Este tema será analizado en secciones posteriores.

El conjunto de instrucciones del 80386 soporta transferencias de 8, 16 y 32 bits. El *bus* de direcciones está organizado para especificar directamente los bytes de datos que están activos durante un ciclo de *bus* dado. Los 30 bits de mayor orden de cada dirección se envían por las patillas A2-A31. Las patillas BE0-BE3 (*Byte Enable* o habilitación de byte) indican que los bytes del *bus* de datos son relevantes en la transferencia en curso. La BE0 corresponde a D0-D7, la BE1 corresponde a D8-D15, etc. Estas patillas de *byte enable* se corresponden directamente con el modo en que se organizan la mayoría de los subsistemas de

cuando esté disponible. La gestión en *pipeline* de las direcciones puede proporcionar al *hardware* externo un margen de tres pulsos de reloj para el acceso a los datos, conservando una anchura de banda de dos ciclos de reloj para el procesador.

La gestión en *pipeline* de las direcciones se aprovecha mejor en los sistemas de memoria que pueden responder a accesos en paralelo a bancos alternos (*interleave*). Poniendo a 1 la señal *Next Address* (siguiente dirección), el *hardware* externo puede pedir al 80386 que envíe la siguiente dirección de ciclo de *bus* tan pronto como el procesador pueda disponer de ella, en lugar de esperar a la señal *READY* (véase figura 5.6).

5.2.5. Dimensionamiento dinámico del *bus*

Además de controlar la temporización de las definiciones de ciclo de *bus*, el subsistema de memoria (e I/O) puede también controlar dinámicamente el tamaño efectivo del *bus* dinámico. El dimensionamiento dinámico del *bus* permite:

1. Las combinaciones arbitrarias de subsistema de memoria de 16 y 32 bits; el *software* puede realizar transferencias de 32 bits sin preocuparse de si está accediendo a una memoria de 16 ó 32 bits.
2. La conexión sencilla a buses de 16 bits, tales como el *bus* MULTIBUS I.
3. La compatibilidad con los periféricos de 16 bits (y sus controladores), cuyos registros suelen estar situados en direcciones de 16 bits y no de 32.

Poniendo a 1 la señal **tamaño de *bus* 16** (BS16), el *hardware* externo puede informar al procesador para que realice la transferencia actual únicamente por los 16 bits menos significativos del *bus* de datos. Si la señal BS16 está a 1 y el acceso es de 32 bits, el procesador empleará automáticamente dos ciclos de *bus* (véase figura 5.7). El 80386 muestrea la señal BS16 al final del ciclo de *bus*, permi-

tiendo que el *hardware* externo la ponga a 1 sólo para las direcciones relevantes de memoria e I/O.

5.2.6. Control y estado del procesador

Cualquier otro maestro del *bus* (un procesador o un periférico inteligente, como un controlador DMA) puede solicitar el uso del *bus* local del 80386 poniendo a 1 la señal **HOLD**. El procesador garantiza el *bus* poniendo a 1 **HLDA** (**Hold Acknowledge**, o *hold* reconocido) al final del ciclo de *bus* actual; a continuación suspenderá el próximo ciclo de *bus* hasta que **HOLD** vuelva a 0. Cuando el 80386 entrega el control del *bus* a otro maestro, activa la señal **HLDA** y pone en alta impedancia el resto de sus patillas, aislándose eléctricamente del sistema.

Las interrupciones del 80386 se clasifican en enmascarables y no enmascarables; el primer grupo entra por la patilla **INTR** (**Interrupt Request**, o petición de interrupción) del procesador, mientras que el segundo lo hace por la patilla **NMI** (**Non-Maskable Interrupt**, o interrupción no enmascarable). El *software* del sistema operativo puede hacer que el 80386 ignore la patilla **INTR** poniendo a 0 el *flag* de habilitación de interrupciones (*Interrupt Enable*). El procesador siempre muestrea la patilla **NMI**; la mayoría de los sistemas usan esta patilla para informar al procesador cuándo tiene lugar un fallo de red o un error fatal del sistema.

Las peticiones de interrupción enmascarable suelen estar conectadas a **INTR** a través de uno o más Controladores de Interrupciones Programables (PIC) 8259A. Cada 8259A puede controlar hasta 8 fuentes de interrupción; se pueden conectar varios 8259A en cascada para proporcionar hasta 64 fuentes de interrupción diferentes. El sistema operativo inicializa cada 8259A con un número identificador (*vector*) para cada fuente de interrupciones que el PIC controla. El 8259A envía este número al 80386 en respuesta al ciclo de *bus* de reconocimiento de la interrupción del procesador. Este usa dicho número para in-

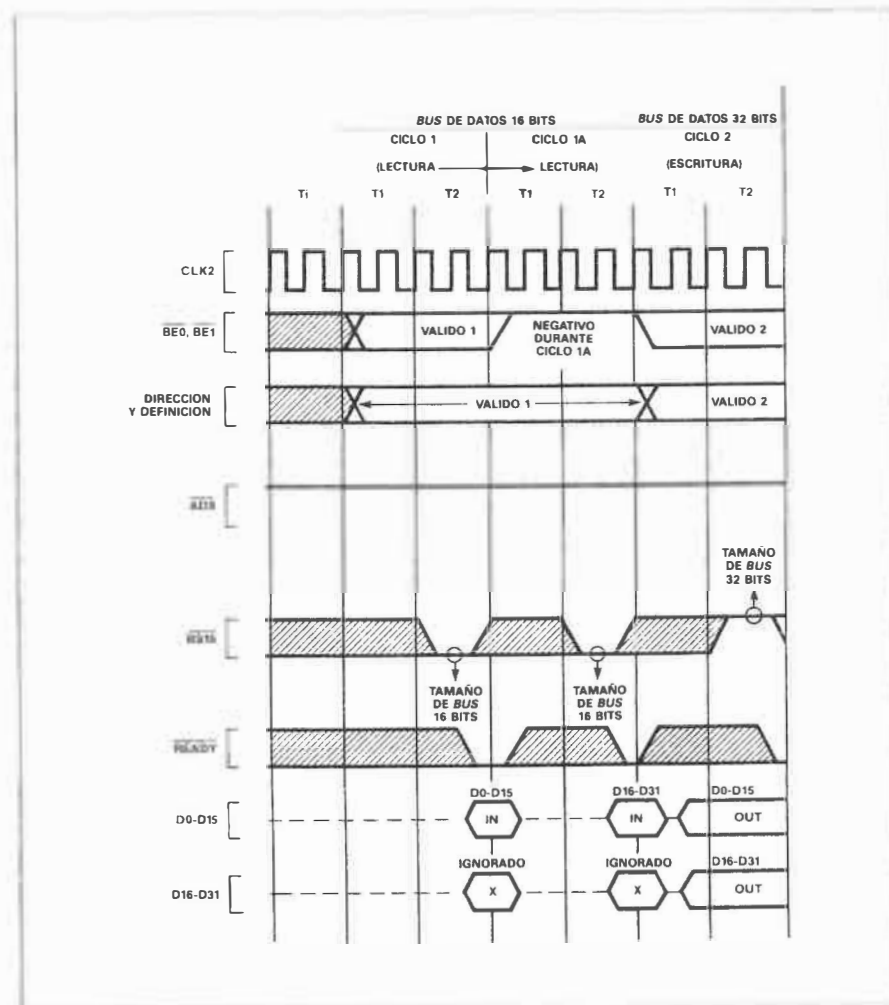


Figura 5.7 Accesos mixtos de 16 y 32 bits

vocar al gestor asignado para responder a dicha interrupción.

La puesta a 1 de la señal **RESET** pone al

procesador en un estado inicial predefinido (en Modo Real con las interrupciones inhabilitadas) y le hace iniciar la búsqueda de

la instrucción situada en la dirección física FFFFFFF0H.

5.2.7. Control del coprocesador

El 80386 le pasa instrucciones y operandos al coprocesador numérico 80287 u 80387 ejecutando ciclos de *bus* de I/O accediendo a direcciones reservadas por encima del espacio de I/O normal de 64 kbytes. Un coprocesador numérico se selecciona poniendo a 1 la línea de direcciones A31 y a 0 la línea M/I $\bar{O}$. El 80386 usa protocolos de comunicaciones diferentes para cada coprocesador, pasándole al 80287 paquetes de 16 bits y de 32 bits al 80387. El 80386 puede detectar cuándo se inicializa mediante un RESET si hay presente un 80387; la

detección de la presencia de un 80287 puede realizarla el *software* de inicialización del sistema.

El coprocesador pone a 1 la señal **BUSY** mientras se está ejecutando una instrucción. El 80386 no le pasará la siguiente instrucción numérica hasta que dicha señal se vuelva a poner a 0. Es posible sincronizar al 80386 con el coprocesador mediante *software* utilizando la instrucción **WAIT**, que suspende al 80386 hasta que la señal **BUSY** vuelva a 0. El coprocesador pone a 1 la señal **ERROR** cuando encuentra una excepción que debiera ser tratada por el *software* del sistema operativo; a su vez el 80386 invoca al gestor de excepciones numéricas señalando una excepción 7. La pata **PEREQ** se utiliza para implementar el protocolo 80386-coprocesador.

