Página principal del sitio de Darío Alpern | See Site in English

ELECTRÓNICA >> Microprocesadores Intel | Descargas
MATEMÁTICAS >> Calculadoras | Teoría de Números | Problemas
PROGRAMAS >> Assembler 80386 (descargas) | Java | Juegos
CONTACTO >> Personal | Comentarios | Libro de invitados | Enlaces

El microprocesador 80386

por Dario Alejandro Alpern

Introducción

El 80386 consiste en una unidad central de proceso (**CPU**), una unidad de manejo de memoria (**MMU**) y una unidad de interfaz con el bus (**BIU**).

La **CPU** está compuesta por la unidad de ejecución y la unidad de instrucciones. La unidad de ejecución contiene los ocho registros de 32 bits de propósito general que se utilizan para el cálculo de direcciones y operaciones con datos y un *barrel shifter* de 64 bits que se utiliza para acelerar las operaciones de desplazamiento, rotación, multiplicación y división. Al contrario de los microprocesadores previos, la lógica de división y multiplicación utiliza un algoritmo de 1 bit por ciclo de reloj. El algoritmo de multiplicación termina la iteración cuando los bits más significativos del multiplicador son todos ceros, lo que permite que las multiplicaciones típicas de 32 bits se realicen en menos de un microsegundo.

La unidad de instrucción decodifica los códigos de operación (*opcodes*) de las instrucciones que se encuentran en una cola de instrucciones (cuya longitud es de 16 bytes) y los almacena en la cola de instrucciones decodificadas (hay espacio para tres instrucciones).

El sistema de control de la unidad de ejecución es el encargado de decodificar las instrucciones que le envía la cola y enviarle las órdenes a la unidad aritmética y lógica según una tabla que tiene almacenada en ROM llamada CROM (Control Read Only Memory).

La unidad de manejo de memoria (**MMU**) consiste en una unidad de segmentación (similar a la del 80286) y una unidad de paginado (nuevo en este microprocesador). La segmentación permite el manejo del espacio de direcciones lógicas agregando un componente de direccionamiento extra, que permite que el código y los datos se puedan reubicar fácilmente. El mecanismo de paginado opera por debajo y es transparente al proceso de segmentación, para permitir el manejo del espacio de direcciones físicas. Cada segmento se divide en uno o más páginas de 4 kilobytes. Para implementar un sistema de memoria virtual (aquél donde el programa tiene un tamaño mayor que la memoria física y debe cargarse por partes (páginas) desde el disco rígido), el 80386 permite seguir ejecutando los programas después de haberse detectado fallos de segmentos o de páginas. Si una página determinada no se encuentra en memoria, el 80386 se lo indica al sistema operativo mediante la excepción 14, luego éste carga dicha página desde el disco y finalmente puede seguir ejecutando el programa, como si hubiera estado dicha página todo el tiempo. Como se puede observar, este proceso es transparente para la aplicación, por lo que el programador no debe preocuparse por cargar partes del código desde el disco ya que esto lo hace el sistema operativo con la ayuda del microprocesador.

La memoria se organiza en uno o más segmentos de longitud variable, con tamaño máximo de 4

gigabytes. Estos segmentos, como se vio en la explicación del 80286, tienen atributos asociados, que incluyen su ubicación, tamaño, tipo (pila, código o datos) y características de protección.

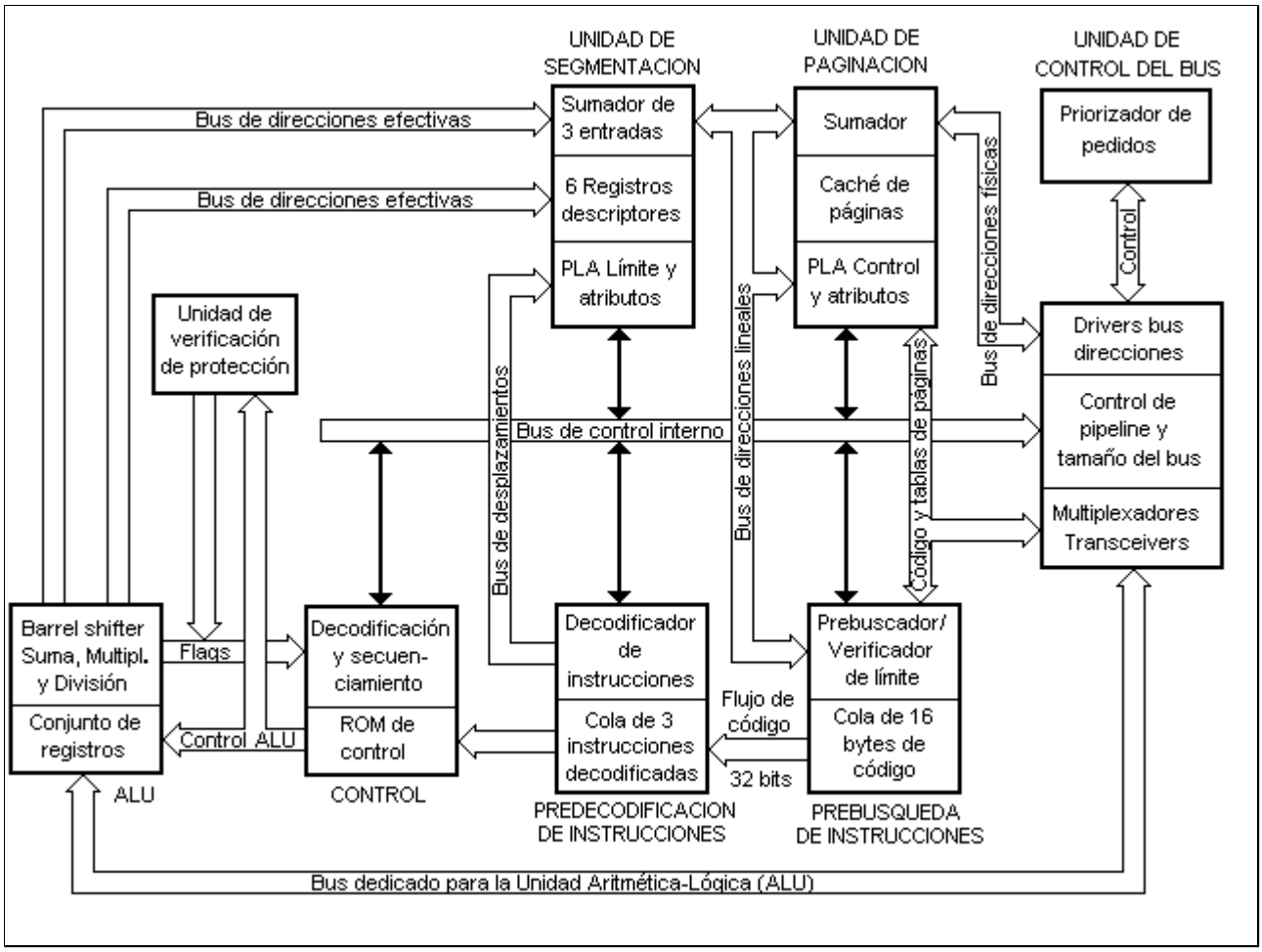
La unidad de segmentación provee cuatro niveles de protección para aislar y proteger aplicaciones y el sistema operativo. Este tipo de protección por hardware permite el diseño de sistemas con un alto grado de integridad.

El 80386 tiene dos modos de operación: modo de direccionamiento real (modo real), y modo de direccionamiento virtual protegido (modo protegido). En modo real el 80386 opera como un 8086 muy rápido, con extensiones de 32 bits si se desea. El modo real se requiere primariamente para preparar el procesador para que opere en modo protegido. El modo protegido provee el acceso al sofisticado manejo de memoria y paginado.

Dentro del modo protegido, el software puede realizar un cambio de tarea para entrar en tareas en modo 8086 virtual (*V86 mode*) (esto es nuevo con este microprocesador). Cada una de estas tareas se comporta como si fuera un 8086 el que lo está ejecutando, lo que permite ejecutar software de 8086 (un programa de aplicación o un sistema operativo). Las tareas en modo 8086 virtual pueden aislarse entre sí y del sistema operativo (que debe utilizar instrucciones del 80386), mediante el uso del paginado y el mapa de bits de permiso de entrada/salida (*I/O Permission Bitmap*).

Finalmente, para facilitar diseños de hardware de alto rendimiento, la interfaz con el bus del 80386 ofrece *pipelining* de direcciones, tamaño dinámico del ancho del bus de datos (puede tener 16 ó 32 bits según se desee en un determinado ciclo de bus) y señales de habilitación de bytes por cada byte del bus de datos. Hay más información sobre esto en la sección de hardware del 80386.

Diagrama en bloques del 80386



Versiones del 80386

- **80386**: En octubre de 1985 la empresa Intel lanzó el microprocesador **80386** original de 16 MHz, con una velocidad de ejecución de 6 millones de instrucciones por segundo y con 275.000 transistores. La primera empresa en realizar una computadora compatible com IBM PC AT basada en el 80386 fue Compaq con su Compaq Deskpro 386 al año siguiente.

- **386SX**: Para facilitar la transición entre las computadoras de 16 bits basadas en el 80286, apareció en junio de 1988 el **80386 SX** con bus de datos de 16 bits y 24 bits de direcciones (al igual que en el caso del 80286). Este microprocesador permitió el armado de computadoras en forma económica que pudieran correr programas de 32 bits. El 80386 original se le cambió de nombre: **80386 DX**.

- **386SL**: En 1990 Intel introdujo el miembro de alta integración de la familia 386: el **80386 SL** con varias características extras (25 MHz, frecuencia reducida ó 0 MHz, interfaz para caché opcional externo de 16, 32 ó 64 KB, soporte de LIM 4.0 (memoria expandida) por hardware, generación y verificación de paridad, ancho de bus de datos de 8 ó 16 bits) que lo hacen ideal para equipos portátiles.

Registros del 80386

El 80386 tiene registros de 32 bits en las siguientes categorías:

- Registros de propósito general.
- Registros de segmento.
- Puntero de instrucciones
- Indicadores.
- Registros de control (nuevos en el 80386).
- Registros de direcciones de sistema.
- Registros de depuración (*debug*) (nuevos en el 80386).
- Registros de test (nuevos en el 80386).

Todos los registros de los microprocesadores 8086, 80186 y 80286 son un subconjunto de los del 80386.

La siguiente figura muestra los registros de la arquitectura base del 80386, que incluye los registros de uso general, el puntero de instrucciones y el registro de indicadores. Los contenidos de estos registros y de los selectores del párrafo siguiente son específicos para cada tarea, así que se cargan automáticamente al ocurrir una operación de cambio de tarea. La arquitectura base también incluye seis segmentos direccionables directamente, cada uno de 4 gigabytes de tamaño máximo. Los segmentos se indican mediante valores de selectores puestos en los registros de segmento del 80386. Si se desea se pueden cargar diferentes selectores a medida que corre el programa.

Registros de la arquitectura base

Tipo	Registros	Bits 31-16	Bits 15-0	Descripción
Uso general	**EAX**	EAX_{31-16}	EAX_{15-0} = **AX**	Acumulador
	EBX	EBX_{31-16}	EBX_{15-0} = **BX**	Base
	ECX	ECX_{31-16}	ECX_{15-0} = **CX**	Contador
	EDX	EDX_{31-16}	EDX_{15-0} = **DX**	Datos
	ESI	ESI_{31-16}	ESI_{15-0} = **SI**	Indice Fuente
	EDI	EDI_{31-16}	EDI_{15-0} = **DI**	Indice Destino
	EBP	EBP_{31-16}	EBP_{15-0} = **BP**	Puntero Base
	ESP	ESP_{31-16}	ESP_{15-0} = **SP**	Puntero de Pila
De segmento	**CS**	No aplicable: estos registros son de 16 bits	**CS**	Segmento de código
	SS		**SS**	Segmento de pila
	DS		**DS**	Segmento de datos
	ES		**DS**	Segmentos de datos extra
	FS		**DS**	
	GS		**DS**	
Otros	**EIP**	EIP_{31-16}	EIP_{15-0} = **IP**	Puntero de instrucciones

	EFlags	$EFlags_{31-16}$	$EFlags_{15-0}$ = **Flags**	Indicadores

Registros de propósito general

Los ocho registros de propósito general de 32 bits mantienen datos y direcciones. Estos registros soportan operandos de 1, 8, 16, 32 y 64 bits y campos de bits de 1 a 32 bits. Soportan operandos de direcciones de 16 y de 32 bits. Los nombres simbólicos son: **EAX**, **EBX**, **ECX**, **EDX**, **ESI**, **EDI**, **EBP** y **ESP**. Los 16 bits menos significativos se pueden acceder separadamente. Esto se hace usando los nombres **AX**, **BX**, **CX**, **DX**, **SI**, **DI**, **BP** y **SP**, que se utilizan de la misma manera que en los procesadores previos. Al igual que en el 80286 y anteriores, **AX** se divide en **AH** y **AL**, **BX** se divide en **BH** y **BL**, **CX** se divide en **CH** y **CL** y **DX** se divide en **DH** y **DL**.

Los ocho **registros de uso general** de 32 bits se pueden usar para direccionamiento indirecto. Cualquiera de los ocho registros puede ser la base y cualquiera menos **ESP** puede ser el índice. El índice se puede multiplicar por 1, 2, 4 u 8.

Ejemplos de direccionamiento indirecto:

- *MOV ECX,[ESP]*
- *MOV AL, [EAX + EDI * 8]*
- *ADD CL, [EDX + EDX + 8245525h]*
- *INC DWORD PTR TABLA[EAX * 4]*

En modo real y en modo 8086 virtual, la suma de la base, el índice y el desplazamiento debe estar entre 0 y 65535 para que no se genere una excepción 13. El segmento por defecto es **SS** si se utiliza **EBP** o **ESP** como base, en caso contrario es **DS**. En el caso de usar direccionamiento de 16 bits, sólo se pueden usar las mismas combinaciones que para el 8088. No se pueden mezclar registros de 16 y de 32 bits para direccionamiento indirecto.

Puntero de instrucciones

El **puntero de instrucciones** es un registro de 32 bits llamado **EIP**, el cual mantiene el offset de la próxima instrucción a ejecutar. El offset siempre es relativo a la base del segmento de código (**CS**). Los 16 bits menos significativos de **EIP** conforman el *puntero de instrucciones de 16 bits* llamado **IP**, que se utiliza para direccionamiento de 16 bits.

Registro de indicadores

Es un registro de 32 bits llamado **EFlags**. Los bits definidos y campos de bits controlan ciertas operaciones e indican el estado del 80386. Los 16 bits menos significativos (bits 15-0) llevan el nombre de **Flags**, que es más útil cuando se ejecuta código de 8086 y 80286. La descripción de los indicadores es la siguiente:

- **VM** (*Virtual 8086 Mode*, bit 17): Este bit provee el modo 8086 virtual dentro del modo protegido. Si se pone a 1 cuando el 80386 está en modo protegido, entrará al modo 8086 virtual, manejando los segmentos como lo hace el 8086, pero generando una excepción 13 (Violación general de protección) en instrucciones privilegiadas (aquéllas que sólo se pueden ejecutar en modo real o en el anillo 0). El bit **VM** sólo puede ponerse a 1 en modo protegido mediante la instrucción IRET

(ejecutando en nivel de privilegio cero) y por cambios de tarea en cualquier anillo. El bit **VM** no cambia con la instrucción POPFD. La instrucción PUSHFD siempre pone un cero en el bit correspondiente en la pila, aunque el modo 8086 virtual esté activado. La imagen de **EFlags** puesta en la pila durante el procesamiento de una interrupción o salvada en un Task State Segment durante cambios de tarea, contendrá un uno en el bit 17 si el código interrumpido estaba ejecutando una tarea tipo 8086 virtual.

- **RF** (*Resume Flag*, bit 16): Este indicador se utiliza junto con los registros de depuración. Se verifica en las fronteras de instrucciones antes del procesamiento de los puntos de parada (*breakpoints*). Cuando **RF** vale 1, hace que se ignoren las *faltas* (hechos que ocasionan una excepción) de depuración. **RF** se pone automáticamente a cero luego de ejecutar correctamente cualquier instrucción (no se señalan faltas) excepto las instrucciones IRET y POPF y en los cambios de tarea causados por la ejecución de JMP, CALL e INT. Estas instrucciones ponen **RF** al valor especificado por la imagen almacenada en memoria. Por ejemplo, al final de la rutina de servicio de los puntos de parada, la instrucción IRET puede extraer de la pila una imagen de **EFlags** que tiene **RF** = 1 y resumir la ejecución del programa en la dirección del punto de parada sin generar otra falta de punto de parada en el mismo lugar.

- **NT** (*Nested Task*, bit 14): Este indicador se aplica al modo protegido. **NT** se pone a uno para indicar que la ejecución de la tarea está anidada dentro de otra tarea. Si está a uno, indica que el segmento de estado de la tarea (**TSS**) de la tarea anidada tiene un puntero válido al TSS de la tarea previa. Este bit se pone a cero o uno mediante transferencias de control a otras tareas. La instrucción IRET verifica el valor de **NT** para determinar si debe realizar un retorno dentro de la misma tarea o si debe hacer un cambio de tarea. Un POPF o IRET afectará el valor de este indicador de acuerdo a la imagen que estaba en la pila, en cualquier nivel de privilegio.

- **IOPL** (*Input/Output Privilege Level*, bits 13-12): Este campo de dos bits se aplica al modo protegido. **IOPL** indica el CPL (*Current Privilege Level*) numéricamente máximo (esto es, con menor nivel de privilegio) permitido para realizar instrucciones de entrada/salida sin generar una excepción 13 (Violación general de protección) o consultar el mapa de bits de permiso de E/S (este mapa está ubicado en el segmento de estado de tarea (**TSS**) con el nuevo formato que provee el 80386). Además indica el máximo valor de CPL que permite el cambio del indicador **IF** (indicador de habilitación del pin **INTR**) cuando se ejecuta POPF e IRET. Estas dos últimas instrucciones sólo pueden alterar **IOPL** cuando CPL = 0. Los cambios de tarea siempre alteran el campo **IOPL**, cuando la nueva imagen de los indicadores se carga desde el TSS de la nueva tarea.

- **OF** (*Overflow flag*, bit 11): Si vale 1, hubo un desborde en una operación aritmética con signo, esto es, un dígito significativo se perdió debido a que tamaño del resultado es mayor que el tamaño del destino.

- **DF** (*Direction Flag*, bit 10): Define si **ESI** y/o **EDI** se autoincrementan (**DF** = 0) o autodecrementan (**DF** = 1) durante instrucciones de cadena.

- **IF** (***INTR*** *enable Flag*, bit 9): Si vale 1, permite el reconocimiento de interrupciones externas señaladas en el pin **INTR**. Cuando vale cero, las interrupciones externas señaladas en el pin **INTR** no se reconocen. **IOPL** indica el máximo valor de CPL que permite la alteración del indicador **IF** cuando se ponen nuevos valores en **EFlags** desde la pila.

- **TF** (*Trap enable Flag*, bit 8): Controla la generación de la excepción 1 cuando se ejecuta código paso a paso. Cuando **TF** = 1, el 80386 genera una excepción 1 después que se ejecuta la

instrucción en curso. Cuando **TF** = 0, la excepción 1 sólo puede ocurrir como resultado de las direcciones de punto de parada cargadas en los registros de depuración (**DR0**-**DR3**).

- **SF** (*Sign Flag*, bit 7): Se pone a 1 si el bit más significativo del resultado de una operación aritmética o lógica vale 1 y se pone a cero en caso contrario. Para operaciones de 8, 16, 32 bits, el indicador **SF** refleja el estado de los bits 7, 15 y 31 respectivamente.

- **ZF** (*Zero Flag*, bit 6): Se pone a 1 si todos los bits del resultado valen cero, en caso contrario se pone a cero.

- **AF** (*Auxiliary carry Flag*, bit 4): Se usa para simplificar la adición y sustracción de cantidades BCD empaquetados. **AF** se pone a 1 si hubo un préstamo o acarreo del bit 3 al 4. De otra manera el indicador se pone a cero.

- **PF** (*Parity Flag*, bit 2): se pone a uno si los ocho bits menos significativos del resultado tienen un número par de unos (paridad par), y se pone a cero en caso de paridad impar (cantidad impar de unos). El indicador **PF** es función de los ocho bits menos significativos del resultado, independientemente del tamaño de las operaciones.

- **CF** (*Carry Flag*, bit 0): Se pone a uno si hubo arrastre (suma) o préstamo (resta) del bit más significativo del resultado.

Los bits 5 y 3 siempre valen cero, mientras que el bit 1 siempre vale uno.

Registros de segmento del 80386

Son seis registros de 16 bits que mantienen valores de selectores de segmentos identificando los segmentos que se pueden direccionar. En modo protegido, cada segmento puede tener entre un byte y el espacio total de direccionamiento (4 gigabytes). En modo real, el tamaño del segmento siempre es 64 KB.

Los seis segmentos direccionables en cualquier momento se definen mediante los registros de segmento **CS**, **DS**, **ES**, **FS**, **GS**, **SS**. El selector en **CS** indica el segmento de código actual, el selector en **SS** indica el segmento de pila actual y los selectores en los otros registros indican los segmentos actuales de datos.

Registros descriptores de segmento: Estos registros no son visibles para el programador, pero es muy útil conocer su contenido. Dentro del 80386, un **registro descriptor** (invisible para el programador) está asociado con cada registro de segmento (visible para el programador). Cada descriptor mantiene una dirección base de 32 bits, un límite (tamaño) de 32 bits y otros atributos del segmento.

Cuando un selector se carga en un registro de segmento, el **registro descriptor** asociado se cambia automáticamente con la información correcta. En modo real, sólo la dirección base se cambia (desplazando el valor del selector cuatro bits hacia la izquierda), ya que el límite y los otros atributos son fijos. En modo protegido, la dirección base, el límite y los otros atributos se cargan con el contenido de una tabla usando el selector como índice.

Siempre que ocurre una referencia a memoria, se utiliza automáticamente el **registro descriptor de segmento** asociado con el segmento que se está usando. La dirección base de 32 bits se convierte en uno de los componentes para calcular la dirección, el límite de 32 bits se usa para verificar si una referencia no supera dicho límite (no se referencia fuera del segmento) y los atributos se verifican para determinar

si hubo alguna violación de protección u otro tipo.

Registros de control

El 80386 tiene tres registros de control de 32 bits, llamados **CR0**, **CR2** y **CR3**, para mantener el estado de la máquina de naturaleza global (no el específico de una tarea determinada). Estos registros, junto con los registros de direcciones del sistema, mantienen el estado de la máquina que afecta a todas las tareas en el sistema. Para acceder los registros de control, se utiliza la instrucción **MOV**.

CR0 (Registro de control de la máquina): Contiene seis bits definidos para propósitos de control y estado. Los 16 bits menos significativos de **CR0** también se conocen con el nombre de *palabra de estado de la máquina* (**MSW**), para la compatibilidad con el modo protegido del 80286. Las instrucciones LMSW y SMSW se toman como casos particulares de carga y almacenamiento de **CR0** donde sólo se opera con los 16 bits menos significativos de **CR0**. Para lograr la compatibilidad con sistemas operativos del 80286 la instrucción LMSW opera en forma idéntica que en el 80286 (ignora los nuevos bits definidos en **CR0**). Los bits definidos de **CR0** son los siguientes:

- **PG** (*Paging Enable*, bit 31 de **CR0**): Este bit se pone a uno para habilitar la unidad de paginado que posee el chip. Se pone a cero para deshabilitarlo. El paginado sólo funciona en modo protegido, por lo que si **PG** = 1, deberá ser PE = 1. Nuevo en 80386.

- **ET** (*Processor Extension Type*, bit 4 de **CR0**): Indica el tipo de coprocesador (80287, 80387) según se detecte por el nivel del pin **/ERROR** al activar el pin **RESET**. El bit **ET** puede ponerse a cero o a uno cargando **CR0** bajo control del programa. Si **ET** = 1, se usa el protocolo de 32 bits del 80387, mientras que si **ET** = 0, se usa el protocolo de 16 bits del 80287. Para lograr la compatibilidad con el 80286, la instrucción LMSW no altera este bit. Sin embargo, al ejecutar SMSW, se podrá leer en el bit 4 el valor de este indicador. Nuevo en 80386.

- **TS** (*Task Switched*, bit 3 de **CR0**): Se pone a uno cuando se realiza una operación de cambio de tarea. Si **TS** = 1, un código de operación **ESC** del coprocesador causará una excepción 7 (Coprocesador no disponible). El manejador de la excepción típicamente salva el contexto del 80287/80387 que pertenece a la tarea previa, carga el estado del 80287/80387 con los valores de la nueva tarea y pone a cero el indicador **TS** antes de retornar a la instrucción que causó la excepción.

- **EM** (*Emulate Coprocessor*, bit 2 de **CR0**): Si este bit vale uno, hará que todos los códigos de operación de coprocesador causen una excepción 7 (Coprocesador no disponible). Si vale cero, permite que esas instrucciones se ejecuten en el 80287/80387 (éste es el valor del indicador después del RESET). El código de operación WAIT no se ve afectado por el valor del indicador.

- **MP** (*Monitor Coprocessor*, bit 1 de **CR0**): Este indicador se usa junto con TS para determinar si la instrucción WAIT generará una excepción 7 cuando TS = 1. Si MP = TS = 1, al ejecutar WAIT se genera la excepción. Nótese que TS se pone a uno cada vez que se realiza un cambio de tarea.

- **PE** (*Protection Enable*, bit 0 de **CR0**): Se pone a uno para habilitar el modo protegido. Si vale cero, se opera otra vez en modo real. **PE** se puede poner a uno cargando **MSW** o **CR0**, pero puede ser puesto a cero sólo mediante una carga en **CR0**. Poner el bit **PE** a cero es apenas una parte de una secuencia de instrucciones necesarias para la transición de modo protegido a modo real. Nótese que para la compatibilidad estricta con el 80286, **PE** no puede ponerse a cero con la instrucción LMSW.

CR2 (Dirección lineal de falta de página): Mantiene la dirección lineal de 32 bits que causó la última falta de página detectada. El código de error puesto en la pila del manejador de la falta de página cuando se la invoca provee información adicional sobre la falta de página.

Un cambio de tareas a través de un TSS que cambie el valor de **CR3**, o una carga explícita de **CR3** con cualquier valor, invalidará todas las entradas en la tabla de páginas que se encuentran en el caché de la unidad de paginación. Si el valor de **CR3** no cambia durante el cambio de tareas se considerarán válidos los valores almacenados en el caché.

Registros de direcciones del sistema

Cuatro registros especiales se definen en el modelo de protección del 80286/80386 para referenciar tablas o segmentos. Estos últimos son:

- GDT (Tabla de descriptores globales).
- IDT (Tabla de descriptores de interrupción).
- LDT (Tabla de descriptores locales).
- TSS (Segmento de estado de la tarea).

Las direcciones de estas tablas y segmentos se almacenan en registros especiales, llamados **GDTR**, **IDTR**, **LDTR** y **TR** respectivamente.

GDTR e **IDTR**: Estos registros mantienen la dirección lineal base de 32 bits y el límite de 16 bits de **GDT** e **IDT**, respectivamente. Los segmentos **GDT** e **IDT**, como son globales para todas las tareas en el sistema, se definen mediante direcciones lineales de 32 bits (sujeto a traducción de página si el paginado está habilitado mediante el bit PG) y límite de 16 bits.

LDTR y **TR**: Estos registros mantienen los selectores de 16 bits para el descriptor de LDT y de TSS, respectivamente. Los segmentos LDT y TSS, como son específicos para cada tarea, se definen mediante valores de selector almacenado en los registros de segmento del sistema. Nótese que un registro descriptor del segmento (invisible para el programador) está asociado con cada registro de segmento del sistema.

Registros de depuración

Al igual que en los procesadores anteriores, el 80386 tiene algunas características que simplifica el proceso de depuración de programas. Las dos características compartidas con los microprocesadores anteriores son:

1) El código de operación de punto de parada INT 3 (**0CCh**).

2) La capacidad de ejecución paso a paso que provee el indicador TF.

Lo nuevo en el 80386 son los **registros de depuración**. Los seis registros de depuración de 32 bits accesibles al programador, proveen soporte para depuración (*debugging*) por hardware. Los registros DR0-DR3 especifican los cuatro puntos de parada (*breakpoints*). Como los puntos de parada se indican mediante registros en el interior del chip, un punto de parada de ejecución de instrucciones se puede ubicar en memoria ROM o en código compartido por varias tareas, lo que no es posible utilizando el código de operación INT 3. El registro de control DR7 se utiliza para poner y habilitar los puntos de parada y el registro de estado DR6 indica el estado actual de los puntos de parada. Después del *reset*, los

puntos de parada están deshabilitados. Los puntos de parada que ocurren debido a los registros de depuración generan una excepción 1.

Registros de direcciones lineales de puntos de parada (DR0 - DR3): Se pueden especificar hasta cuatro direcciones de puntos de parada escribiendo en los registros **DR0** a **DR3**. Las direcciones especificadas son direcciones lineales de 32 bits. El hardware del 80386 continuamente compara las direcciones lineales de los registros con las direcciones lineales que genera el software que se está ejecutando (una dirección lineal es el resultado de computar la dirección efectiva y sumarle la dirección base de 32 bits del segmento). Nótese que si la unidad de paginación del 80386 no está habilitada (mediante el bit PG del registro CR0), la dirección lineal coincide con la física (la que sale por el bus de direcciones), mientras que si está habilitada, la dirección lineal se traduce en la física mediante dicha unidad.

Registro de control de depuración (DR7): La definición de los bits de este registro es la siguiente:

Campo	**LEN3**		**RW3**		**LEN2**		**RW2**		**LEN1**		**RW1**		**LEN0**		**RW0**	
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

Campo	**Indef.**		**GD**	**Indef.**			**GE**	**LE**	**G3**	**L3**	**G2**	**L2**	**G1**	**L1**	**G0**	**L0**
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

- Campo **LENi** (*Campo de especificación de longitud del punto de parada*): Por cada punto de parada existe un campo de dos bits **LEN**. Este campo especifica la longitud del campo que produce el punto de parada. En el caso de punto de parada por acceso de datos se puede elegir 1, 2 ó 4 bytes, mientras que los de lectura de instrucciones deben tener una longitud de un byte (**LENi** = 00). La codificación del campo es como sigue:

  LENi = 00 (1 byte): Los 32 bits de DRi se utilizan para especificar el punto de parada.
  LENi = 01 (2 bytes): Se utilizan los bits 31-1 de DRi.
  LENi = 10: Indefinido. No debe utilizarse.
  LENi = 11 (4 bytes): Se utilizan los bits 31-2 de DRi.

- Campo **RWi** (*Bits calificadores de acceso a memoria*): Existe un campo **RW** de dos bits por cada uno de los puntos de parada. Este campo especifica el tipo de uso de memoria que produce el punto de parada:

  RWi = 00: Ejecución de instrucciones solamente.
  RWi = 01: Escritura de datos solamente.
  RWi = 10: Indefinido. No debe utilizarse.
  RWi = 11: Lectura y/o escritura de datos solamente.

  Los puntos de parada (excepción 1) debido a la ejecución de instrucciones ocurren ANTES de la ejecución, mientras que los puntos de parada debido a acceso a datos ocurren DESPUES del acceso.

  Uso de LENi y RWi para poner un punto de parada debido a acceso a datos: Esto se realiza cargando la dirección lineal del dato en **DRi** (i=0-3). RWi puede valer 01 (sólo parar en escritura) o 11 (parar en lectura/escritura). LENi puede ser igual a 00 (un byte), 01 (2 bytes) ó 11 (4 bytes).

Si un acceso de datos cae parcial o totalmente dentro del campo definido por **DRi** y LENi y el punto de parada está habilitado, se producirá la excepción 1.

Uso de LENi y RWi para poner un punto de parada debido a la ejecución de una instrucción: Debe cargarse DRi con la dirección del comienzo (del primer prefijo si hay alguno). RWi y LENi deben valer 00. Si se está por ejecutar la instrucción que comienza en la dirección del punto de parada y dicho punto de parada está habilitado, se producirá una excepción 1 antes de que ocurra la ejecución de la instrucción.

- Campo **GD** (*Detección de acceso de registros de depuración*): Los registros de depuración sólo se pueden acceder en modo real o en el nivel de privilegio cero en modo protegido (no se pueden acceder en modo 8086 virtual). El bit **GD**, cuando vale uno, provee una protección extra contra cualquier acceso a estos registros aun en modo real o en el anillo 0 del modo protegido. Esta protección adicional sirve para garantizar que un depurador por software (Codeview, Turbo Debugger, etc.) pueda tener el control total de los recursos provistos por los registros de depuración cuando sea necesario. El bit **GD**, cuando vale 1, causa una excepción 1 si una instrucción trata de leer o escribir algún registro de depuración. El bit **GD** se pone automáticamente a cero al ocurrir dicha excepción, permitiendo al manejador el control de los registros de depuración.

- Bits **GE** y **LE** (*Punto de parada exactamente cuando ocurre el acceso a datos, global y local*): Si **GE** o **LE** valen uno, cualquier punto de parada debido a acceso a datos se indicará exactamente después de completar la instrucción que causó la transferencia. Esto ocurre forzando la unidad de ejecución del 80386 que espere que termine la transferencia de datos antes de comenzar la siguiente instrucción. Si **GE** = **LE** = 0, la parada ocurrirá varias instrucciones después de la transferencia que lo debería haber causado. Cuando el 80386 cambia de tarea, el bit **LE** se pone a cero. Esto permite que las otras tareas se ejecuten a máxima velocidad. El bit **LE** debe ponerse a uno bajo control del software. El bit **GE** no se altera cuando ocurre un cambio de tarea. Los puntos de parada debido a la ejecución de instrucciones siempre ocurren exactamente, independientemente del valor de **GE** y **LE**.

- Bits **Gi** y **Li** (*Habilitación del punto de parada, global y local*): Si cualquiera de estos bits vale uno, se habilita el punto de parada asociado. Si el 80386 detecta la condición de parada ocurre una excepción 1. Cuando se realiza un cambio de tareas, todos los bits **Li** se ponen a cero, para evitar excepciones espúreas en la nueva tarea. Estos bits deben ponerse a uno bajo control del software. Los bits **Gi** no se afectan durante un cambio de tarea. Los bits **Gi** soportan puntos de parada que están activos en todas las tareas que se ejecutan en el sistema.

Registro de estado de depuración (DR6): Este registro permite que el manejador de la excepción 1 determine fácilmente por qué fue llamado. El manejador puede ser invocado como resultado de uno de los siguientes eventos:

1. Punto de parada debido a DR0.
2. Punto de parada debido a DR1.
3. Punto de parada debido a DR2.
4. Punto de parada debido a DR3.
5. Acceso a un registro de depuración cuando GD = 1.
6. Ejecución paso a paso (si TF = 1).
7. Cambio de tareas.

El registro DR6 contiene indicadores para cada uno de los eventos arriba mencionados. Los otros bits son indefinidos. Estos indicadores se ponen a uno por hardware pero nunca puestos a cero por hardware, por lo que el manejador de la excepción 1 deberá poner **DR6** a cero.

Condición	7	6	5	4	3	2	1
Indicador	BT	BS	BD	B3	B2	B1	B0
Bit	15	14	13	3	2	1	0

Registros de test

Se utilizan dos registros para verificar el funcionamiento del **RAM/CAM** (*Content Addressable Memory*) en el buffer de conversión por búsqueda (**TLB**) de la unidad de paginado del 80386. **TR6** es el registro de comando del test, mientras que **TR7** es el registro de datos que contiene el dato proveniente del **TLB**. El **TLB** guarda las entradas de tabla de página de uso más reciente en un caché que se incluye en el chip, para reducir los accesos a las tablas de páginas basadas en RAM.

Acceso a registros

Hay algunas diferencias en el acceso de registros en modo real y protegido, que se indican a continuación. Escr = Escritura del registro, Lect = Lectura del registro.

Registro	**Modo Real**		**Modo Protegido**		**Modo virtual 8086**	
	Escr	**Lect**	**Escr**	**Lect**	**Escr**	**Lect**
Registros generales	Sí	Sí	Sí	Sí	Sí	Sí
Registros de segmento	Sí	Sí	Sí	Sí	Sí	Sí
Indicadores	Sí	Sí	Sí	Sí	IOPL	IOPL
Registros de control	Sí	Sí	CPL=0	CPL=0	No	Sí
GDTR	Sí	Sí	CPL=0	Sí	No	Sí
IDTR	Sí	Sí	CPL=0	Sí	No	Sí
LDTR	No	No	CPL=0	Sí	No	No
TR	No	No	CPL=0	Sí	No	No
Registros de depuración	Sí	Sí	CPL=0	CPL=0	No	No
Registros de test	Sí	Sí	CPL=0	CPL=0	No	No

CPL=0: Los registros se pueden acceder sólo si el nivel de privilegio actual es cero.

IOPL: Las instrucciones PUSHF y POPF son sensibles al indicador IOPL en modo 8086 virtual.

Compatibilidad

Algunos bits de ciertos registros del 80386 no están definidos. Cuando se obtienen estos bits, deben tratarse como completamente indefinidos. Esto es esencial para la compatibilidad con procesadores

futuros. Para ello deben seguirse las siguientes recomendaciones:

1. No depender de los estados de cualquiera de los bits no definidos. Estos deben ser enmascarados (mediante la instrucción AND con el bit a enmascarar a cero) cuando se utilizan los registros.

2. No depender de los estados de cualquiera de los bits no definidos cuando se los almacena en memoria u otro registro.

3. No depender de la habilidad que tiene el procesador de retener información escrita en bits marcados como indefinidos.

4. Cuando se cargan registros siempre se deben poner los bits indefinidos a cero.

5. Los registros que se almacenaron previamente pueden ser recargados sin necesidad de enmascarar los bits indefinidos.

Los programas que no cumplen con estas indicaciones, pueden llegar a no funcionar en 80486, Pentium y siguientes procesadores, donde los bits no definidos del 80386 poseen algún significado en los otros procesadores.

Modo protegido en el 80386

Está basado en el modo protegido del 80286, por lo que se recomienda primero leer dicha información. Aquí se mostrarán las diferencias entre ambos microprocesadores.

Descriptores de segmento

Debido a la mayor cantidad de funciones del 80386, estos descriptores tienen más campos que los descriptores para el 80286.

El formato general de un descriptor en el 80386 es:

- **Byte 0**: Límite del segmento (bits 7-0).
- **Byte 1**: Límite del segmento (bits 15-8).
- **Byte 2**: Dirección base del segmento (bits 7-0).
- **Byte 3**: Dirección base del segmento (bits 15-8).
- **Byte 4**: Dirección base del segmento (bits 23-16).
- **Byte 5**: Derechos de acceso del segmento.
- **Byte 6**:
  - Bit 7: ***Granularidad*** (**G**): Si vale cero, el límite es el indicado por el campo límite (bits 19-0), mientras que si vale uno, a dicho campo se le agregarán 12 bits a uno a la derecha para formar el límite (de esta manera el límite se multiplica por 4096, pudiendo llegar hasta 4GB como límite máximo).
  - Bit 6: ***Default*** (para segmento de código)/***Big*** (para segmento de pila)(**D**/**B**): Para los descriptores de segmentos de código si este bit está a uno, el segmento será de 32 bits (se utiliza **EIP** como offset del puntero de instrucciones, mientras que si vale cero el segmento será de 16 bits (utiliza **IP** como offset del puntero de instrucciones). Para segmento de pila si el bit vale uno, el procesador usará el registro **ESP** como puntero de pila, en caso contrario usará **SP**.

    - Bits 5 y 4: Deben valer cero.
    - Bits 3 a 0: Límite del segmento (bits 19-16).
- **Byte 7**: Dirección base del segmento (bits 31-24).

El **byte de derechos de acceso** es el que define qué clase de descriptor es. El bit 4 (S) indica si el segmento es de código o datos (S = 1), o si es del sistema (S = 0). Veremos el primer caso.

- Bit 7: **Presente (P)**. Si P = 1 el segmento existe en memoria física, mientras que si P = 0 el segmento no está en memoria. Un intento de acceder un segmento que no está **presente** cargando un registro de segmento (CS, DS, ES, SS, FS o GS) con un selector que apunte a un descriptor que indique que el segmento no está **presente** generará una excepción 11 (en el caso de SS se generará una excepción 12 indicando que la pila no está **presente**). El manejador de la interrupción 11 deberá leer el segmento del disco rígido. Esto sirve para implementar memoria virtual.

- Bits 6 y 5: **Nivel de privilegio del descriptor (DPL)**: Atributo de privilegio del segmento utilizado en tests de privilegio.

- Bit 4: **Bit descriptor del segmento (S)**: Como se explicó más arriba, este bit vale 1 para descriptores de código y datos.

- Bit 3: **Ejecutable (E)**: Determina si el segmento es de código (E = 1), o de datos (E = 0).
  Si el bit 3 vale cero:
    - Bit 2: **Dirección de expansión (ED)**: Si E = 0, el segmento se expande hacia arriba, con lo que los offsets deben ser menores o iguales que el límite. Si E = 1, el segmento se expande hacia abajo, con lo que los offsets deben ser mayores que el límite. Si no ocurre esto se genera una excepción 13 (Fallo general de protección) (en el caso de la pila se genera una excepción 12).
    - Bit 1: **Habilitación de escritura (W)**: Si W = 0 no se puede escribir sobre el segmento, mientras que si W = 1 se puede realizar la escritura.

  Si el bit 3 vale uno:
    - Bit 2: **Conforme (C)**: Si C = 1, el segmento de código sólo puede ser ejecutado si CPL es mayor que DPL y CPL no cambia. Los segmentos conformes sirven para rutinas del sistema operativo que no requieran protección, tales como rutinas matemáticas, por ejemplo.
    - Bit 1: **Habilitación de lectura (R)**: Si R = 0, el segmento de código no se puede leer, mientras que si R = 1 sí. No se puede escribir sobre el segmento de código.
- Bit 0: **Accedido (A)**: Si A = 0 el segmento no fue accedido, mientras que si A = 1 el selector se ha cargado en un registro de segmento o utilizado por instrucciones de test de selectores. Este bit es puesto a uno por el microprocesador.

Si se lee o escribe en un segmento donde no está permitido o se intenta ejecutar en un segmento de datos se genera una excepción 13 (Violación general de protección). Si bien no se puede escribir sobre un segmento de código, éstos se pueden inicializar o modificar mediante un **alias**. Los **alias** son segmentos de datos con permiso de escritura (E = 0, W = 1) cuyo rango de direcciones coincide con el segmento de código.

Los segmentos de código cuyo bit C vale 1, pueden ejecutarse y compartirse por programas con diferentes niveles de privilegio (ver la sección sobre protección, más adelante).

A continuación se verá el formato del **byte de derechos de acceso** para descriptores de segmentos del sistema:

- Bit 7: **Presente (P)**: Igual que antes.
- Bits 6 y 5: **Nivel de privilegio del descriptor (DPL)**: Igual que antes.
- Bit 4: **Bit descriptor del segmento (S)**: Como se explicó más arriba, este bit vale 0 para descriptores del sistema.

- Bits 3-0: **Tipo de descriptor**: En el 80386 están disponibles los siguientes:

0000: Inválido	1000: Inválido
0001: TSS tipo 286 disponible	1001: TSS tipo 386 disponible
0010: LDT	1010: Inválido
0011: TSS tipo 286 ocupado	1011: TSS tipo 386 ocupado
0100: Compuerta de llamada tipo 286	1100: Compuerta de llamada tipo 386
0101: Compuerta de tarea tipo 286 ó 386	1101: Inválido
0110: Compuerta de interrupción tipo 286	1110: Compuerta de interrupción tipo 386
0111: Compuerta de trampa tipo 286	1111: Compuerta de trampa tipo 386

Vea la información sobre TSS tipo 286, LDT y compuertas en las secciones correspondientes del microprocesador 80286.

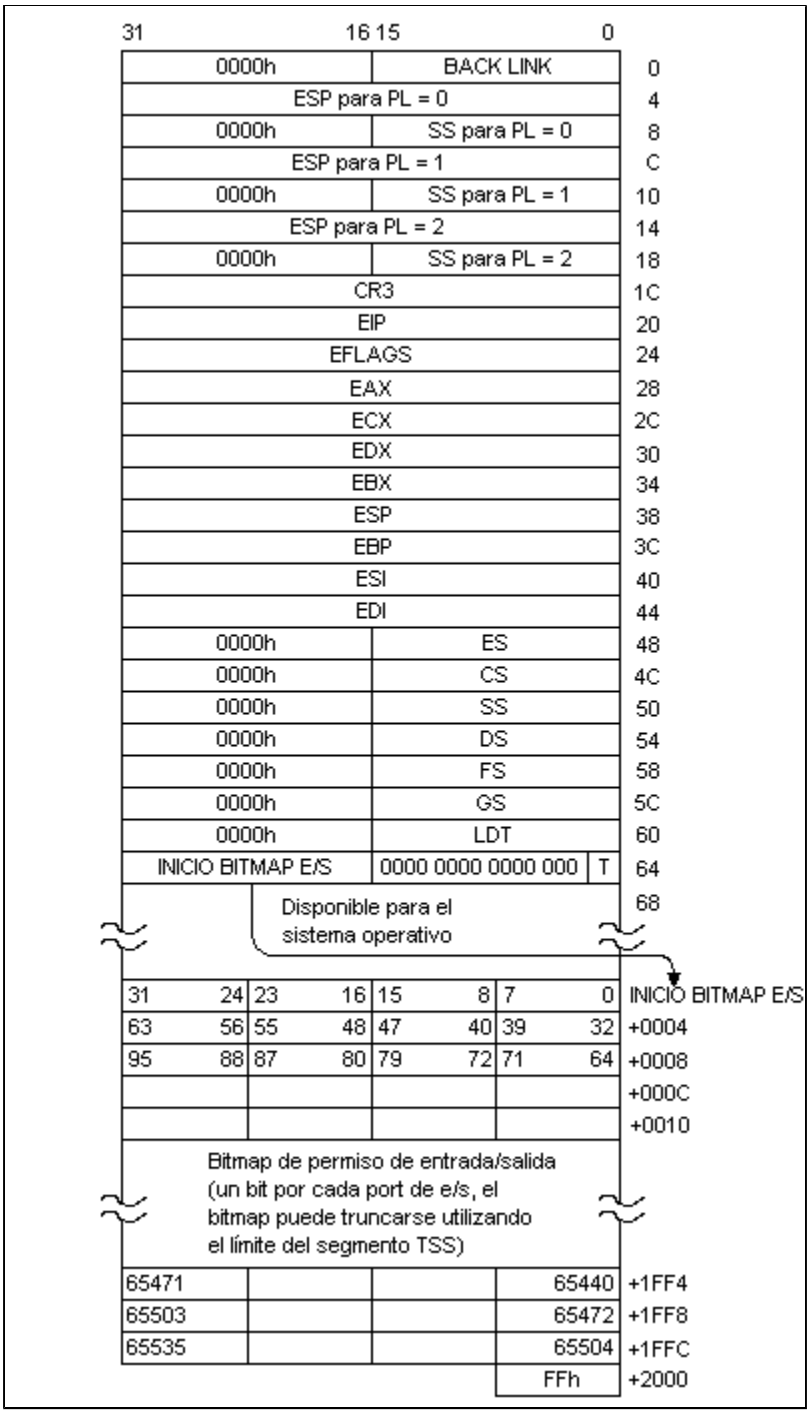
El formato de las compuertas tipo 386 es diferente de las del tipo 286, por lo que se muestra a continuación:

- Byte 0: Offset (bits 7-0).
- Byte 1: Offset (bits 15-8).
- Byte 2: Selector (bits 7-0).
- Byte 3: Selector (bits 15-0).
- Byte 4: Bits 4-0: Cantidad de palabras, bits 7-5: Cero.
- Byte 5: Derechos de acceso.
- Byte 6: Offset (bits 23-16).
- Byte 7: Offset (bits 31-24).

Segmento de estado de la tarea

El descriptor **TSS** apunta a un segmento que contiene el estado de la ejecución del 80386 mientras que un descriptor de compuerta de tarea contiene un selector de **TSS**.

En el 80386 hay dos tipos de **TSS**: tipo 286 y tipo 386. El primero es idéntico al TSS del microprocesador 80286, mientras que la del 80386 tiene el siguiente formato:



Mapa de bits de permisos de entrada/salida

Aparte de poder almacenar los registros nuevos del 80386, el segmento de estado de la tarea tiene este nuevo campo, como se puede apreciar en la parte inferior de la tabla.

Cuando el procesador debe acceder a un puerto de entrada/salida (usando una instrucción **OUT**, **IN**, **OUTS** o **INS**) en modo protegido, el procesador primero verifica si **CPL** <= **IOPL**. Si esto ocurre, la instrucción se ejecuta. En caso contrario si la tarea en ejecución está asociada a una TSS tipo 286 el 80386 lanza una excepción 13. Si está asociada con una TSS tipo 386, el procesador consulta esta tabla para saber si debe ejecutar la instrucción o si debe lanzar una excepción 13.

El mapa de bits de permisos de entrada/salida se puede ver como una cadena de bits cuya longitud puede ser entre cero y 65536 bits. Cada bit corresponde a un puerto (ver la figura que está más arriba). Si el bit está a cero, la instrucción de E/S se puede ejecutar. De esta manera se pueden proteger zonas de espacio de E/S en forma selectiva.

El inicio del mapa de bits está indicado por el puntero que está en el offset 66h del TSS. El mapa de bits puede ser truncado ajustando el límite del segmento TSS. Todos los puertos de E/S que no tengan una entrada en el mapa de bits debido a lo anterior, no se podrán acceder (es como si estuviera el bit correspondiente del mapa a "1").

Al final del mapa de bits debe haber un byte a FFh. Este byte debe estar dentro del límite del TSS.

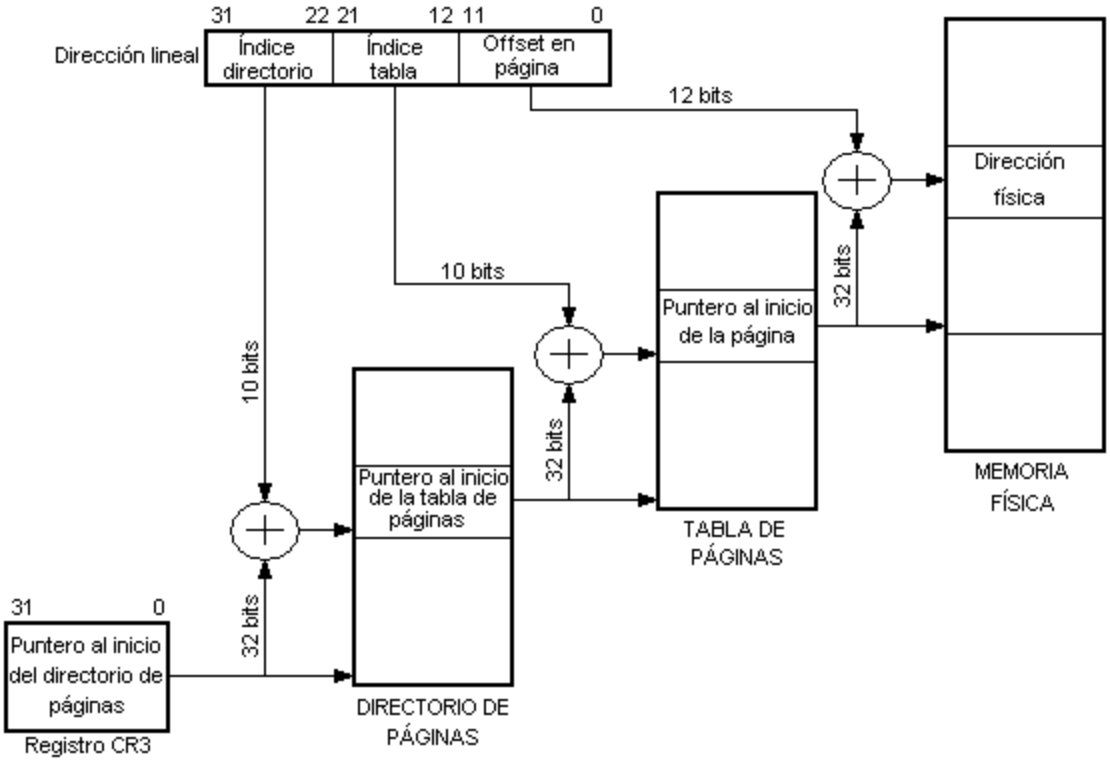
Paginación en el 80386

La paginación es un tipo de manejo de memoria útil para sistemas operativos multitarea que manejan memoria virtual. A diferencia de la segmentación que modulariza programas y datos en segmentos de longitud variable, la paginación divide los programas en varias páginas de tamaño fijo. Estas páginas no tienen niguna relación con la estructura lógica del programa.

Como la mayoría de los programas utilizan referencias cercanas (esto se llama efecto de *localidad*), sólo es necesaria una pequeña cantidad de páginas presentes en la memoria por cada tarea.

Mecanismo de paginación

El 80386 utiliza dos niveles de tablas para traducir las direcciones lineales (que vienen de la unidad de segmentación) en una dirección física. Los tres componentes del mecanismo de paginado son: el directorio de páginas, las tablas de páginas y las páginas mismas. Cada uno de estos elementos ocupa 4KB en la memoria física. Un tamaño uniforme para todos los elementos simplifica el manejo de memoria, ya que no existe fragmentación. La siguiente figura muestra cómo funciona el mecanismo de paginación:



Registros de control usados para la paginación

El registro **CR2** es el que mantiene la dirección lineal de 32 bits que causó el último fallo de página detectado por el microprocesador.

El registro **CR3** contiene la dirección física inicial del directorio de páginas. Los doce bits menos significativos del registro siempre están a cero para que siempre el directorio de páginas esté dentro de una página determinada. La operación de carga mediante la instrucción *MOV CR3, reg* o bien un cambio de tareas que implique un **cambio** de valor del registro CR3 hace que se eliminen las entradas del caché de la tabla de páginas.

Directorio de páginas

La longitud es de 4KB y permite hasta 1024 entradas. Estas entradas se seleccionan mediante los bits 31-22 de la dirección lineal (que viene de la unidad de segmentación). Cada entrada contiene la dirección e información adicional del siguiente nivel de tabla, la tabla de páginas, como se muestra a continuación:

31 - 12	**11 - 9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Dirección tabla de páginas (31-12)	Libre	0	0	D	A	0	0	U/S	R/W	P

Más abajo se explica el significado de estos campos.

Tabla de páginas

Las tablas de páginas tienen una longitud de 4KB y permiten hasta 1024 entradas. Estas entradas se seleccionan mediante los bits 21-12 de la dirección lineal. Cada entrada contiene la dirección inicial e información estadística de la página. Los 20 bits más significativos de la dirección de la página se concatenan con los 12 bits menos significativos de la dirección lineal para formar la dirección física. Las tareas pueden compartir tablas de páginas. Además el sistema operativo puede enviar una o más tablas al disco (si la memoria RAM está llena).

El formato de una entrada de la tabla de páginas es:

31 - 12	**11 - 9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Dirección física de la página (31-12)	Libre	0	0	D	A	0	0	U/S	R/W	P

Más abajo se explica el significado de estos campos.

Entradas del directorio y las tablas de páginas

Los doce bits menos significativos contienen información estadística acerca de las tablas de páginas y las páginas respectivamente. El bit 0 (Presente) indica si la entrada se puede utilizar para traducir de dirección lineal a física. Si P = 0 no se puede, mientras que si P = 1 sí. Cuando P = 0 los otros 31 bits quedan libres para que los use el software. Por ejemplo, estos bits podrían utilizarse para indicar dónde se encuentra la página en el disco.

El bit 5 (Accedido), es puesto a uno por el 80386 en ambos tipos de entradas cuando ocurre un acceso de lectura o escritura en una dirección que esté dentro de una página cubierta por estas entradas. El bit 6 (Dirty), es puesto a uno por el microprocesador cuando ocurre un acceso de escritura. Los tres bits marcados como libre los puede utilizar el software.

Los bits 2 (Usuario/Supervisor) y 1 (Lectura/Escritura) se utilizan para proteger páginas individuales, como se muestra en el siguiente apartado.

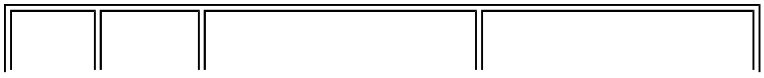
Protección a nivel de página

Para la paginación existen dos niveles de protección: usuario que corresponde al nivel de privilegio 3 y supervisor que corresponde a los otros niveles: 0, 1 y 2. Los programas que se ejecutan en estos niveles no se ven afectados por este esquema de protección.

Los bits U/S y R/W se utilizan para proveer protección Usuario/Supervisor y Lectura/Escritura para páginas individuales o para todas las páginas cubiertas por una tabla de páginas. Esto se logra tomando los bits U/S y R/W más restrictivos (numéricamente inferior) entre el directorio de páginas y la tabla de páginas que corresponda a la página en cuestión.

Por ejemplo, si los bits U/S y R/W para la entrada del directorio de páginas valen 10, mientras que los correspondientes a la tabla de páginas valen 01, los derechos de acceso para la página será 01.

La siguiente tabla muestra la protección que dan estos bits:



U/S	R/W	Acceso permitido nivel 3	Acceso permitido niveles 0, 1, 2
0	0	Ninguno	Lectura/Escritura
0	1	Ninguna	Lectura/Escritura
1	0	Sólo lectura	Lectura/Escritura
1	1	Lectura/Escritura	Lectura/Escritura

Caché de entradas de tablas

Con el esquema visto hasta ahora, el procesador debe realizar dos accesos a tablas por cada referencia en memoria. Esto afectaría notablemente el rendimiento del procesador (por cada acceso indicado por el software habría que hacer tres). Para resolver este problema, el 80386 mantiene un caché con las páginas accedidas más recientemente. Este caché es el buffer de conversión por búsqueda (*TLB = Translation Lookaside Buffer*). El TLB es un caché asociativo de cuatro vías que almacena 32 entradas de tablas de páginas. Como cada página tiene una longitud de 4KB, esto alcanza a 128KB. Para muchos sistemas multitarea, el TLB tendrá un porcentaje de éxito (*hit*) del 98%. Esto significa que el procesador deberá acceder a las dos tablas el 2% del tiempo.

Operación

El hardware de paginación opera de la siguiente manera. La unidad de paginación recibe una dirección lineal de 32 bits procedente de la unidad de segmentación. Los 20 bits más significativos son comparados con las 32 entradas del TLB para determinar si la entrada de la tabla de páginas está en el caché. Si está (*cache hit*), entonces se calcula la dirección física de 32 bits y se la coloca en el bus de direcciones.

Si la entrada de la tabla de páginas no se encuentra en el TLB (*cache miss*), el 80386 leerá la entrada del directorio de páginas que corresponda. Si P=1 (la tabla de páginas está en memoria), entonces el 80386 leerá la entrada que corresponda de la tabla de páginas y pondrá a uno el bit Accedido de la entrada del directorio de páginas. Si P=1 en la entrada de la tabla de páginas indicando que la página se encuentra en memoria, el 80386 actualizará los bits Accedido y Dirty según corresponda y luego accederá a la memoria. Los 20 bits más significativos de la dirección lineal se almacenarán en el TLB para futuras referencias. Si P=0 para cualquiera de las dos tablas, entonces el procesador generará una excepción 14 (Fallo de Página).

El procesador también generará una excepción 14, si la referencia a memoria viola los atributos de protección de página (bits U/S y R/W) (por ejemplo, si el programa trata de escribir a una página que es de sólo lectura). En el registro CR2 se almacenará la dirección lineal que causó el fallo de página. Como la excepción 14 se clasifica como un fallo (es recuperable), CS:EIP apuntará a la instrucción que causó el fallo de página.

En la pila se pondrá un valor de 16 bits que sirve para que el sistema operativo sepa por qué ocurrió la excepción. El formato de esta palabra es:

- Bits 15-3: Indefinido.
- Bit 2: Vale 1 si el procesador estaba ejecutando en modo usuario. Vale 0 si el procesador estaba ejecutando en modo supervisor. Nótese que un acceso a una tabla de descriptores siempre se considera modo supervisor, por más que el programa se esté ejecutando en el nivel 3.

- Bit 1: Vale 1 si el procesador estaba por realizar una escritura. Vale 0 si estaba por realizar una lectura.
- Bit 0: Vale 1 si hubo una violación de protección en la página. Vale 0 si la página no estaba presente.

Modo virtual 8086

El 80386 permite la ejecución de programas para el 8086 tanto en modo real como en modo virtual 8086. De los dos métodos, el modo virtual es el que ofrece al diseñador del sistema la mayor flexibilidad. El modo virtual permite la ejecución de programas para el 8086 manteniendo el mecanismo de protección del 80386. En particular, esto permite la ejecución simultánea de sistemas operativos y aplicaciones para el 8086, y un sistema operativo 80386 corriendo aplicaciones escritas para el 80286 y el 80386. El escenario más común consiste en correr una o más aplicaciones de DOS simultáneamente mientras se corren programas escritos para Windows, todo al mismo tiempo.

Una de las mayores diferencias entre los modos real y protegido es cómo se interpretan los selectores de segmentos. Cuando el procesador ejecuta en modo virtual los registros de segmento se usan de la misma manera que en modo real. El contenido del registro de segmento se desplaza hacia la izquierda cuatro bits y luego se suma al offset para formar la dirección lineal.

El 80386 permite al sistema operativo especificar cuales son los programas que utilizan el mecanismo de direccionamiento del 8086, y los programas que utilizan el direccionamiento de modo protegido, según la tarea que pertenezcan (una tarea determinada corre en modo virtual o en modo protegido). Mediante el uso del paginado el espacio de direcciones de un megabyte del modo virtual se puede mapear en cualquier lugar dentro del espacio de direccionamiento de 4GB. Como en modo real, las direcciones efectivas (offsets) que superen los 64KB causarán una excepción 13.

El hardware de paginación permite que las direcciones lineales de 20 bits producidos por el programa que corre en modo virtual se dividan en 1MB/4(KB/página) = 256 páginas. Cada una de las páginas se pueden ubicar en cualquier lugar dentro del espacio de direcciones físicas de 4GB del 80386. Además, como el registro **CR3** (el registro que indica la base del directorio de páginas) se carga mediante un cambio de tareas, cada tarea que corre en modo virtual puede usar un método exclusivo para realizar la correspondencia entre las páginas y las direcciones físicas. Finalmente, el hardware de paginado permite compartir el código del sistema operativo que corre en 8086 entre múltiples aplicaciones que corren en dicho microprocesador.

Todos los programas que corren en el modo virtual 8086 se ejecutan en el nivel de privilegio 3, el nivel de menor privilegio, por lo que estos programas se deben sujetar a todas las verificaciones de protección que ocurren en modo protegido. En el modo real, los programas corren en el nivel de privilegio cero, el nivel de mayor privilegio.

Cuando una tarea corre en modo virtual, todas las interrupciones y excepciones realizan un cambio de nivel de privilegio hacia el nivel cero, donde corre el sistema operativo 80386.Dicho sistema operativo puede determinar si la interrupción vino de una tarea corriendo en modo virtual o en modo protegido examinando el bit **VM** de la imagen de los indicadores en la pila (el sistema operativo no debe leer directamente el bit **VM** puesto que se pone a cero automáticamente al entrar a la rutina de atención de interrupción).

Para entrar al modo virtual 8086, habrá que ejecutar una instrucción **IRET** cuando CPL = 0 y cuya imagen de los indicadores en la pila tenga el bit **VM** a uno. Otra posibilidad consiste en ejecutar un

cambio de tarea hacia una tarea cuyo TSS tipo 386 tenga el bit **VM** a uno en la imagen de los indicadores.

Vectores de interrupción predefinidos

Las siguientes son las excepciones que puede generar el microprocesador, por lo que, en modo real, deberán estar los vectores correspondientes (en la zona baja de memoria) que apunten a los manejadores, mientras que en modo protegido, deberán estar los descriptores de compuerta correspondientes en la tabla de descriptores de interrupción (**IDT**).

Clase de excepción	Tipo	Instrucción que la causa	El manejador retorna a dicha instrucción
Error de división	0	DIV, IDIV	SÍ
Excepción de depuración	1	Cualquier instrucción	
Interrupción NMI	2	INT 2 o NMI	NO
Interrupción de un byte	3	INT 3	NO
Sobrepasamiento	4	INTO	NO
Fuera de rango	5	BOUND	SÍ
Código inválido	6	Instrucción ilegal	SÍ
El dispositivo no existe	7	ESC, WAIT	SÍ
Doble falta	8	Cualquier instrucción que pueda generar una excepción	
TSS inválido	10	JMP, CALL, IRET, INT	SÍ
Segmento no presente	11	Carga de registro de segmento	SÍ
Falta de pila	12	Referencia a la pila	SÍ
Violación de protección	13	Referencia a memoria	SÍ
Falta de página	14	Acceso a memoria	SÍ
Error del coprocesador	16	ESC, WAIT	SÍ
Reservado	17-32		
Interrupción de 2 bytes	0-255	INT n	NO

Nuevas instrucciones del 80386

Aparte de las instrucciones del 8088, y las nuevas del 80186 y del 80286, el 80386 tiene las siguientes

nuevas instrucciones:

BSF *dest, src* (Bit Scan Forward): Busca el primer bit puesto a 1 del operando fuente *src* (comenzando por el bit cero hasta el bit n-1). Si lo encuentra pone el indicador ZF a 1 y carga el destino *dest* con el índice a dicho bit. En caso contrario pone ZF a 0.

BSR *dest, src* (Bit Scan Reverse): Busca el primer bit puesto a 1 del operando fuente *src* (comenzando por el bit n-1 hasta el bit cero). Si lo encuentra pone el indicador ZF a 1 y carga el destino *dest* con el índice a dicho bit. En caso contrario pone ZF a 0.

BT *dest, src* (Bit Test): El bit del destino *dest* indexado por el valor fuente se copia en el indicador CF.

BTC *dest, src* (Bit Test with Complement): El bit del destino *dest* indexado por el valor fuente se copia en el indicador CF y luego se complementa dicho bit.

BTR *dest, src* (Bit Test with Reset): El bit del destino *dest* indexado por el valor fuente *src* se copia en el indicador CF y luego pone dicho bit a cero.

BTS *dest, src* (Bit Test with Set): El bit del destino *dest* indexado por el valor fuente *src* se copia en el indicador CF y luego pone dicho bit a uno.

CDQ (Convert Doubleword to Quadword): Convierte el número signado de 4 bytes en **EAX** en un número signado de 8 bytes en **EDX:EAX** copiando el bit más significativo (de signo) de **EAX** en todos los bits de **EDX**.

CWDE (Convert Word to Extended Doubleword): Convierte una palabra signada en el registro **AX** en una doble palabra signada en **EAX** copiando el bit de signo (bit 15) de **AX** en los bits 31-16 de **EAX**.

JECXZ *label* (Jump on ECX Zero): Salta a la etiqueta *label* si el registro **ECX** vale cero.

LFS, **LGS**, **LSS** *dest, src* (Load pointer using **FS**, **GS**, **SS**): Carga un puntero de 32 bits de la memoria *src* al registro de uso general destino *dest* y **FS**, **GS** o **SS**. El offset se ubica en el registro destino y el segmento en **FS**, **GS** o **SS**. Para usar esta instrucción la palabra en la dirección indicada por *src* debe contener el offset, y la palabra siguiente debe contener el segmento. Esto simplifica la carga de punteros lejanos (*far*) de la pila y de la tabla de vectores de interrupción.

MOVSX *dest, src* (Move with Sign eXtend): Copia el valor del operando fuente *src* al registro destino *dest* (que tiene el doble de bits que el operando fuente) extendiendo los bits del resultado con el bit de signo. Se utiliza para aritmética signada (con números positivos y negativos).

MOVZX *dest, src* (Move with Zero eXtend): Copia el valor del operando fuente *src* al registro destino *dest* (que tiene el doble de bits que el operando fuente) extendiendo los bits del resultado con ceros. Se utiliza para aritmética no signada (sin números negativos).

POPAD (Pop All Doubleword Registers): Retira los ocho registros de uso general de 32 bits de la pila en el siguiente orden: EDI, ESI, EBP, ESP, EBX, EDX, ECX, EAX. El valor de ESP retirado de la pila se descarta.

POPFD (Pop Doubleword Flags): Retira de la pila los indicadores completos (32 bits).

PUSHAD (Push All Doubleword Registers): Pone los ocho registros de uso general de 32 bits en la pila en el siguiente orden: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI.

PUSHFD (Push Doubleword Flags): Pone los 32 bits del registro de indicadores en la pila.

SETcc *dest* (Set Byte on Condition cc): Pone el byte del destino *dest* a uno si se cumple la condición, en caso contrario lo pone a cero. Las condiciones son las mismas que para los saltos condicionales (se utilizan las mismas letras que van después de la "J").

SHLD *dest, src, count* (Shift Left Double precision): Desplaza *dest* a la izquierda *count* veces y las posiciones abiertas se llenan con los bits más significativos de *src*.

SHRD *dest, src, count* (Shift Left Double precision): Desplaza *dest* a la derecha *count* veces y las posiciones abiertas se llenan con los bits menos significativos de *src*.