Arquitectura de computadoras

Arquitectura, modo real, modo protegido, protección y multitarea

Julian

TECNICAS DIGITALES III

ARQUITECTURA PENTIUM

Contiene dos pipelines, U y V. El U para instrucciones enteras y de punto flotante, el V para enteras simples y de punto flotante FXCH. Capaces de ejecutar dos instrucciones enteras al mismo tiempo.

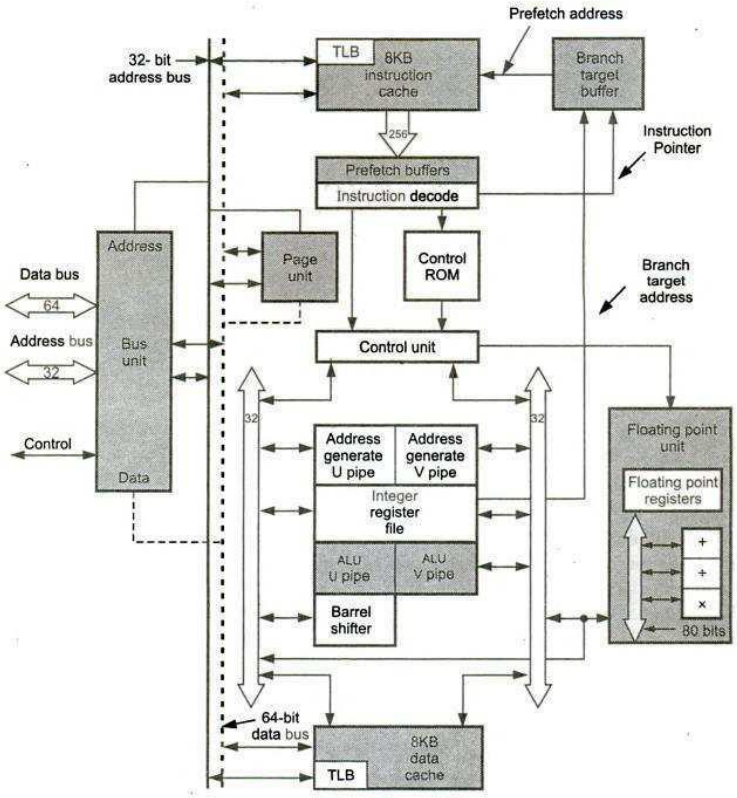


Fig. 1.2 Pentium architecture block diagram

- **Bus Unit:** bus de dirección de 32 bits y de datos de 64 bits. Capaz de realizar lecturas y escrituras en ráfagas de 32 bytes de memoria. Mediante el bus pipeling permite dos ciclos de bus simultáneos.
- **Caché de código:** 8kB, da acceso rápido a instrucciones más usadas. Permite dos accesos simultáneos desde el prefetcher.
- **Unidad de decodificación de instrucciones:** dos etapas D1 y D2.
- **Unidad de control:** interpreta la instrucción y entrada de microcódigo de D2. Maneja excepciones, breakpoints e interrupciones, controla pipelines enteros y secuencias de punto flotante.
- **ALUs:** provee 2 ALUs, una por pipeline. La del U puede completar y operar antes que la V.
- **Generadores de dirección:** uno por pipeline.
- **Caché de datos:** 8 kB, contiene copias de las solicitudes de datos más usadas por los pipelines. Permite acceso simultáneo desde ambos pipelines.
- **Unidad de paginación (modo protegido):** traduce dirección lineal a física. Puede dar soporte a ambos pipelines al mismo tiempos.

MODO REAL

Modo en el que inicia el procesador Pentium (arquitectura IA32) al encenderse. Mantiene así compatibilidad con 8086 admitiendo su misma estructura pero pudiendo acceder al registro de 32 bits del Pentium.

Modelo de programación en modo real

Solo las secciones sombreadas de la siguiente figura forman parte del modo real. Consiste de 8 registros de 16 bits y 8 de 32 bits. Accediendo al registro CR0 se puede entrar en modo protegido.

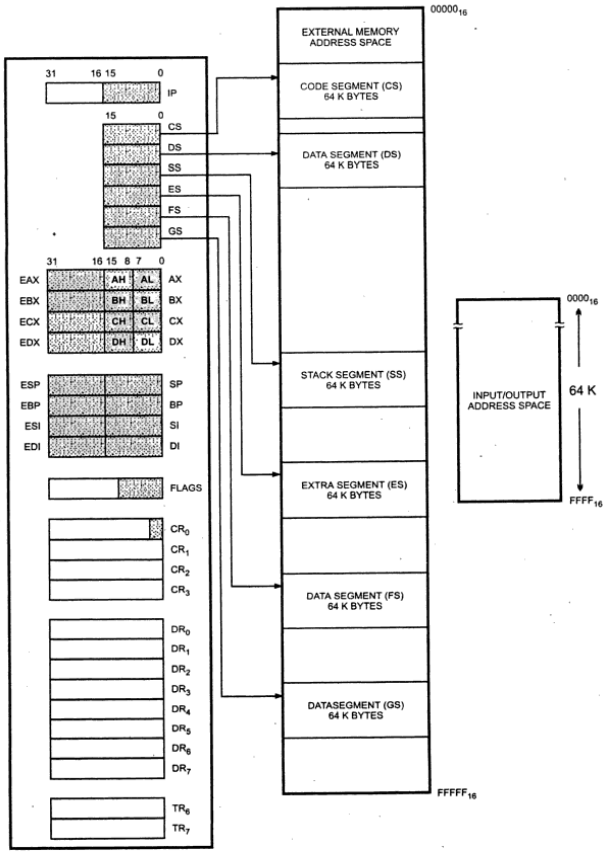


Fig. 1.6 Real mode programming model for Pentium processor

Direccionamiento de memoria en modo real

Memoria limitada a 1MB. Líneas de dirección A3 – A19 activas. El MB de memoria no puede estar activo simultáneamente, la memoria se particiona en segmentos de tamaño máximo de 64kB, cada cual representa una unidad de memoria direccionable independiente. Cada segmento consiste en ubicaciones de memoria consecutivas, con su propia dirección de inicio.

El registro de segmento posee las direcciones de inicio de los segmentos activos. Solo **6 de 16** segmentos pueden estar activos a la vez.

La paginación no está activa en modo real, por lo que las direcciones lineales son iguales a las físicas. Las físicas son generadas añadiendo el contenido de registro de segmento apropiado, desplazado a la izquierda 4 bits para agregar el offset (16 + 4 = 20 bits). En caso de acarreo al dirección pasa a ser de 21 bits.

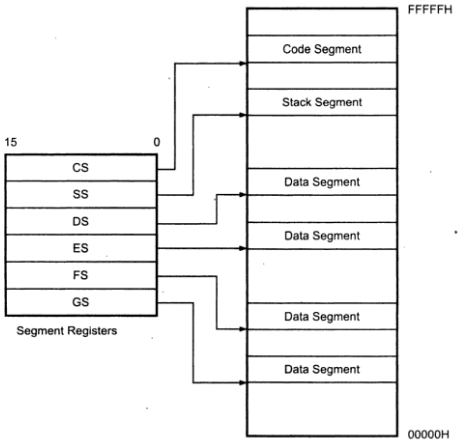


Fig. 1.7 Active segments of memory

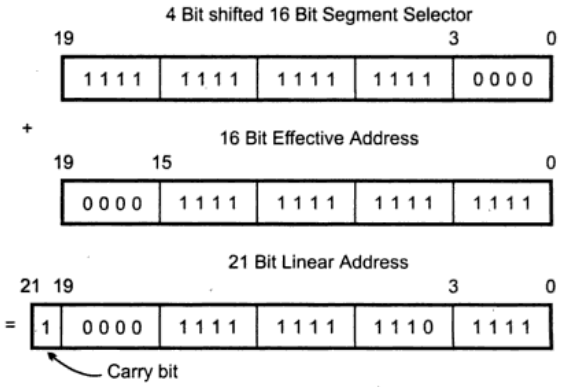


Fig. 1.8 Real address mode addressing

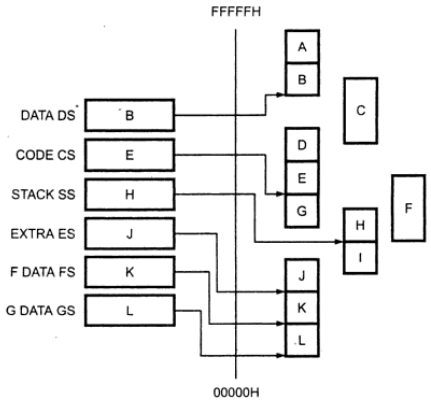


Fig. 1.9 Contiguous, adjacent, disjointed and overlapping segments

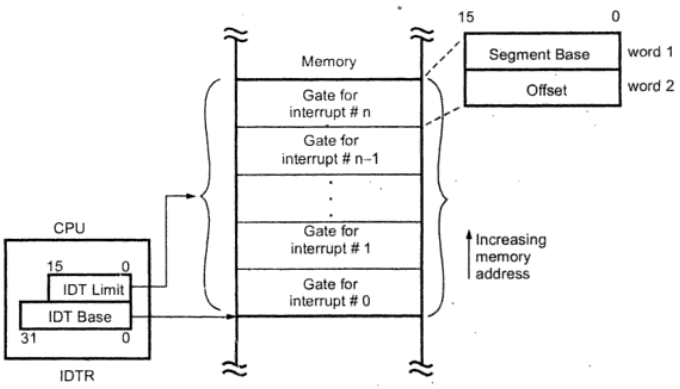
Los segmentos pueden ser leídos, escritos o ejecutados. Si la dirección efectiva está más allá del rango legal(0 a FFFFH) se lanza una excepción de protección general.

Todos los registros de segmento son accesibles a los programadores. Pueden usarse para establecer a los segmentos contiguos, adyacentes, inconexos o solapados.

Manejo de interrupciones y excepciones en modo real

Las ubicaciones de memoria desde 0 a 3FFH (400H ubicaciones) son dedicadas a la IDT. La IDT contiene punteros al inicio de las rutinas de servicio de interrupción. Cada puntero requiere 4 bytes de memoria, por lo que contiene 256 punteros. Los 2 bytes más significativos contienen la dirección base del segmento, los otros 2 el offset. Al ocurrir una interrupción, se multiplica el numero/tipo de interrupción por 4 generando un índice en la IDT.

La IDT es reubicable. Su dirección base se encuentra en el IDTR y al igual que su tamaño, puede modificarse mediante la instrucción LIDT.



Pipeling

Posee 2 instrucciones pipeling (U y V). Se trata de pipelines de 5 etapas independientes, las cuales son:

- PF Prefetch
- D1 Decodificación de instrucción
- D2 Generación de instrucción
- EX Ejecución, acceso a cache y ALU
- WB Writeback (respuesta)

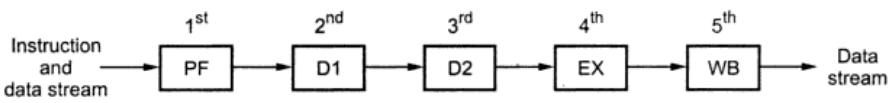


Fig. 1.13 Stages in U and V instruction pipelines

El pipeline U puede ejecutar cualquier instrucción, el V solo instrucciones simples (sin control de microcódigo[1] y normalmente de 1 solo ciclo de reloj).

1. **Etapa PF:** precarga de instrucciones a ambos pipelines.
2. **Etapa D1:** decoder, comprueba si el par de instrucciones actual puede ejecutarse a la vez.
3. **Etapa D2:** cálculo de direcciones de operandos residentes en memoria.
4. **Etapa EX:** lectura de operandos y ejecución de las operaciones de las ALUs.
5. **Etapa WB:** escritura del resultado de la instrucción y verificación de predicción de rama.

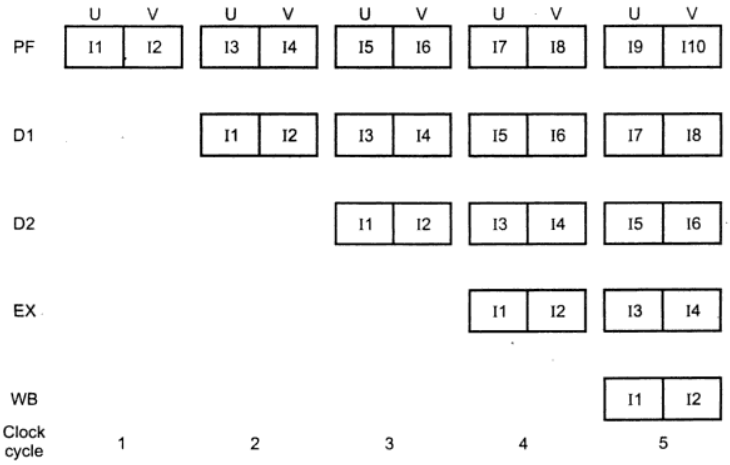


Fig. 1.14 Pipelined instruction execution

MODO PROTEGIDO

Se ingresa a este modo seteando el bit 0 del registro CR0. Aquí se desbloquean todas las capacidades del procesador.

Las características de este modo son:

- Más líneas de direccionamiento y memoria (4 GB), memoria virtual (64 TB).
- Gestor de memoria y protección asistida por hardware.
- Instrucciones especiales para multitasking.
- Paginación.

Los **registros de soporte** en este modo son:

- **GDTR:** 48 bits [32 direccionamiento, 16 límite GDT].
- **LDTR:** 16 bits [selección de LDTD].
- **IDT:** 48 bits [32 direccionamiento, 16 límite IDT].
- **TR:** 16bits [selección TSSR]

Las funciones de los siguientes registros son extendidas:

- Instruction Pointer pasa a ser de 32 bits (EIP).
- EFLAGs activos y con más bits.
- Registros de control $CR_0 - CR_4$ activos.

[1] Microcódigo: capa de instrucciones a nivel de hardware que implementan instrucciones de código de máquina de nivel superior.

Traslación de direcciones lógicas a físicas

Posee 3 espacios de direcciones diferentes: lógica o virtual, lineal y física. La **lógica** consta de un selector y offset, dicho selector es el contenido de un registro de segmento.

En **modo real**, la unidad de segmentación solo desplaza el selector a la izquierda 4 bits para añadir el offset y formar la dirección lineal.

En **modo protegido**, los selectores de segmento tienen una dirección lineal base asociada, la cual se almacena en el descriptor de segmento. Un selector apunta a un descriptor de segmento en la tabla de descriptores. La dirección base lineal del descriptor se agrega luego al offset de 32 bits para generar la dirección de 32 bits. Esto se conoce como **segmentación**.

Si la paginación no está habilitada, la dirección lineal es igual a la física. Si está habilitada, la paginación traslada el espacio de dirección lineal al espacio de dirección física.

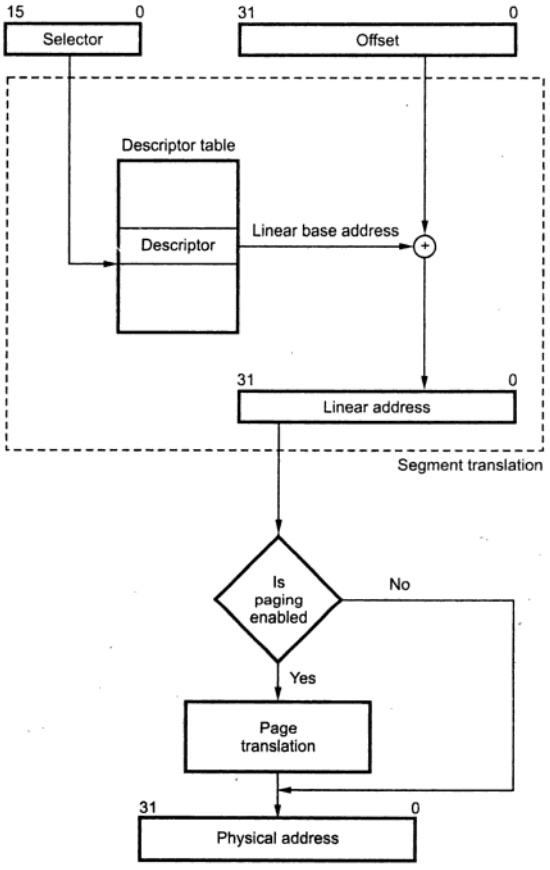


Fig. 4.2 Address translation overview

SEGMENTACION (LÓGICA → LINEAL)

El índice de 13 bits (bits 15 a 3) del selector se multiplica por 8 (descriptores de 8 bytes) y se usa como un puntero al descriptor en la tabla de descriptores. Este descriptor tiene base, límite y Access Right Byte. El procesador añade la dirección base desde el descriptor para la dirección efectiva o el offset para generar la dirección lineal.

El selector tiene 2 bits de nivel de privilegio (RPL – bits 0 y 1) que representan el nivel de privilegio de la sección de programa que solicita acceso al segmento. El descriptor de cada segmento contiene 2 bits que representan el nivel de privilegio de dicho segmento. Cuando un programa en ejecución intenta acceder a un segmento, la unidad administradora de memoria compara los niveles de privilegio del

selector y del descriptor. Si el selector tiene igual o mayor privilegio puede acceder al segmento, en caso contrario se niega el acceso y se lanza una interrupción de violación de nivel de privilegio.

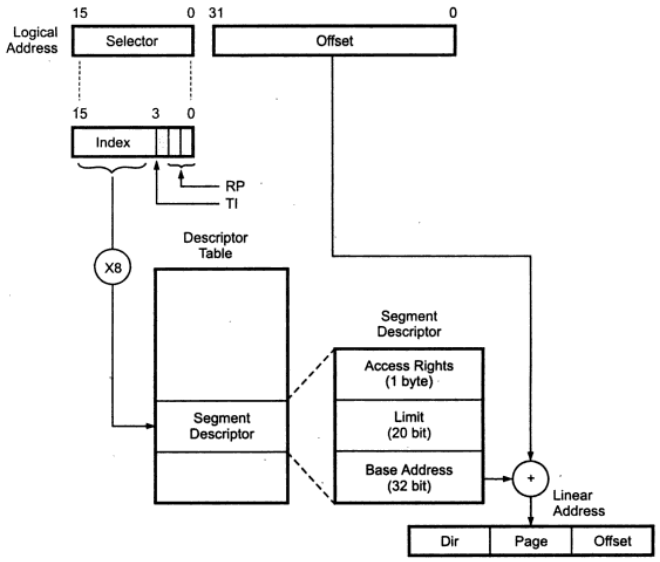


Fig. 4.3 Segment translation mechanism

Existen dos categorías de tablas de descriptores:

- **GDT:** propósito general, usadas por todos los programas para referenciar segmentos de memoria global.
- **LDT:** tareas individuales o grupos de tareas muy relacionadas.

El bit indicador de tabla TI en el selector indica qué tabla debe ser referida (bit 2, TI = 0 para GDT y TI = 1 para LDT).

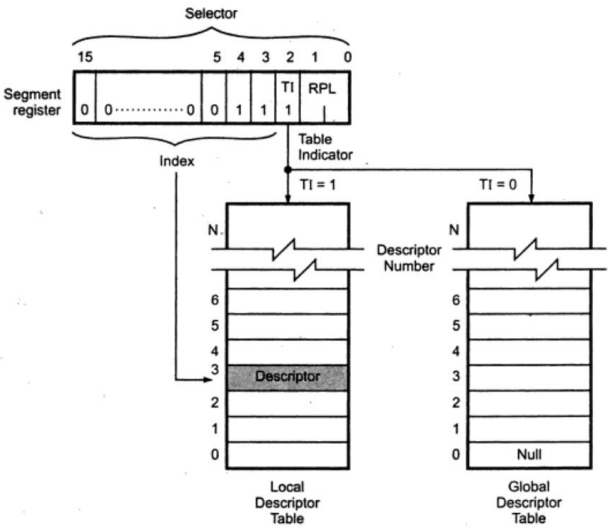


Fig. 4.4 Selector and descriptor tables

La primer entrada en la GDT es reservada (todos ceros) y se conoce como descriptor NULL.

Pentium tiene 6 registros de segmento:

- 1 para código actual (CS).
- 1 para pila actual (SS).
- 4 para segmentos de datos generales(DS, ES, FS, GS).

En resumen, los **registros de segmento** (selectores) seleccionan descriptores de segmento de la siguiente manera:

- 13 bits para seleccionar un descriptor.
- 1 bit para seleccionar una tabla de descriptores.
- 2 bits para chequear prioridad.

Descriptores de segmento y administración de memoria mediante segmentación

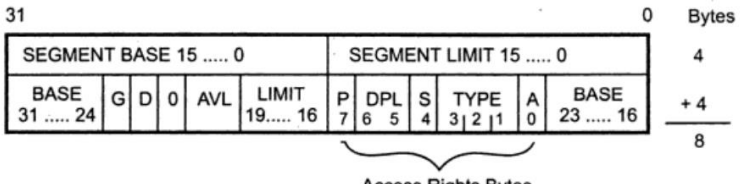
En modo protegido, la MMU usa el selector para acceder a un descriptor de segmento. Debe definirse un descriptor de segmento para cada segmento en memoria.

Los descriptores contienen los siguientes atributos:

- **Base:** 32 bits de dirección lineal base del segmento. Define la dirección dentro del espacio lineal de 4 GB.
- **Limit:** 20 bits de tamaño de *segmento*. Este valor se interpreta dependiendo del valor del bit G.
- **G (Granularity):** define la unidad de medida del limit
  - $G = 0 \rightarrow bytes$
  - $G = 1 \rightarrow 4kB$
- **D (Default Size):** define el tamaño de los operandos del segmento.
  - $D = 0 \rightarrow 16$ bits.
  - $D = 1 \rightarrow 32$ bits.
- **0 (Reservado por Intel):** compatibilidad a futuro.
- **AVL/U (User bit):** indefinido, disponible para usuario o SO.

ACCESS RIGHT BYTE

- **P (Present bit):** es 1 si el segmento está cargado en memoria.
- **DPL:** nivel de privilegio del espacio de memoria que define el descriptor.
- **S (System bit):** indica si un segmento es de sistema (0) o código (1).
- **Type**
- **A (Accessed bit):** seteado cada vez que se accede al segmento.



BASE	Base Address of the segment
LIMIT	The length of the segment
P	Present Bit : 1 = Present 0 = Not present
DPL	Descriptor privilege Level 0 - 3
S	Segment Descriptor : 0 = System Descriptor 1 = Code or Data Segment Descriptor
TYPE	Type of segment
A	Accessed Bit
G	Granularity Bit : 1 = Segment length is page granular 0 = Segment length is byte granular
D	Default Operation Size (recognised in code segment descriptors only) 1 = 32-bit segment 0 = 16-bit segment
0	Bit must be zero (0) for compatibility with future processors
AVL	Available field for user or OS

Note :
In a maximum - size segment (i.e. a segment with G = 1 and segment limit 19 ..... 0 = FFFFFH), the lowest 12 bits of the segment base should be zero. (i.e. segment base 11 ..... 000 = 000H).

Fig. 4.5 General Segment Descriptor Format

Descriptor de segmento	Un descriptor de segmento define
• Describe un segmento. • Creado para todo segmento en memoria. • Creado por el programador. • Determina dirección base, tamaño, tipo y DPL del segmento.	• Dirección base – 32 bits. • Límite – 20 bits. • Tipo – 4 bits. • Privilegio – 2 bits. • Si está en memoria – bit P . • Si ha sido accedido – bit A. • Granularidad – bit G. • Tamaño de operandos – bit D. • Bit reservado – 0. • AVL – 1 bit. • Tamaño por defecto – 1 bit.

TIPOS DE DESCRIPTORES DE SEGMENTO

Existen dos tipos principales, de sistema y non – system. Estos se diferencian en el descriptor según el bit S del Access Right Byte. Dentro de estas 2 categorías, se definen 5 tipos de descriptores más, los cuales se identifican según el valor del campo Type del Access Right Byte.

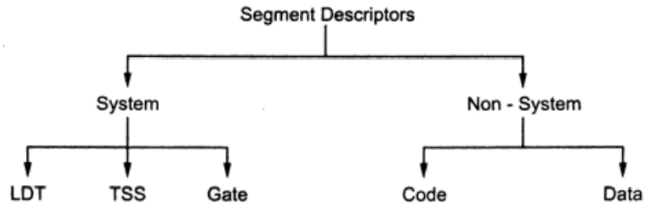
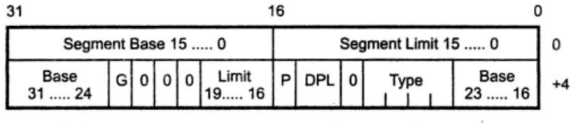


Fig. 4.6 Types of segment descriptors

- **Segmentos de sistema:** dan información del SO, tablas de sistema, tareas y gates.
  - **LDT Descriptors (S = 0, Type = 2):** presente solo en la GDT, contiene información sobre las tablas locales de descriptores. El DPL de este descriptor es ignorado ya que solo puede accederse desde PL0.
  - **TSS Descriptor (S = 0, Type = 1, 3, 9, B):** al cambiar de una tarea1 a una tarea2 se almacena en este descriptor todo lo necesario para regresar a la tarea1 (registros, variables y dirección de instrucciones), denominado contexto de tarea. En el descriptor de TSS se almacena localización, tamaño y PL del TSS.
  - **Gate Descriptors (S = 0, Type = 4 – 7, C, F):** tipo especial de descriptor. Permite al procesador realizar chequeos de protección automáticos. Existen 4 tipos de gates:
    - Call Gate: para cambiar niveles de privilegio.
    - Task Gate: para realizar cambios de tarea.
    - Interrupt y Trap Gate: para rutinas de servicio de interrupción.



Type	Defines	Type	Defines
0	Invalid	8	Invalid
1	Available 80286 TSS	9	Available Intel Pentium processor TSS
2	LDT	A	Undefined (Intel Reserved)
3	Busy 80286 TSS	B	Busy Intel Pentium processor TSS
4	80286 Call Gate	C	Intel Pentium processor Call Gate
5	Task Gate (for 80286 or Intel Pentium processor Task)	D	Undefined (Intel Reserved)
6	80286 Interrupt Gate	E	Intel Pentium processor Interrupt Gate
7	80286 Trap Gate	F	Intel Pentium Processor Trap Gate

Note :
In a maximum - size segment (i.e. a segment with G = 1 and segment limit 19 ..... 0 = FFFFFH), the lowest 12 bits of the segment base should be zero. (i.e. segment base 11 ..... 000 = 000H).

- **Segmentos Non – System:** usados para acceder a segmentos de dato y código.

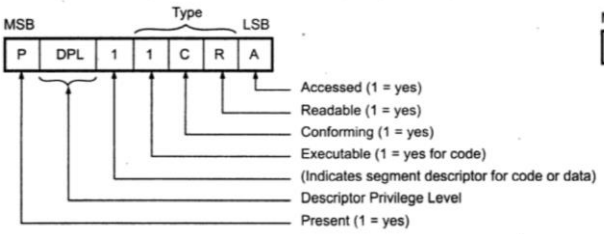


Fig. 4.9 (a) Code segment descriptor access right byte configuration

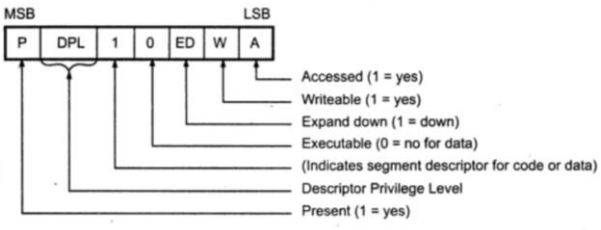


Fig. 4.9 (b) Data segment descriptor access right byte configuration

TABLAS DE DESCRIPTORES

Ubicadas en ubicaciones de memoria contiguas, agrupan los descriptores de segmento. Su longitud máxima es de 64 kB y como cada descriptor ocupa 8 bytes puede tener hasta 8192 descriptores.

Los 3 tipos de tablas existentes son:

- Global GDT: es única, contiene la mayoría de los segmentos usados y puede contener descriptores de sistema especiales.
- Local LDT: es opcional, extiende el alcance de la GDT y es usada por tareas individuales o grupos de tareas muy relacionadas.
- De interrupción IDT: es única, contiene los descriptores que definen rutinas de servicio de interrupciones.

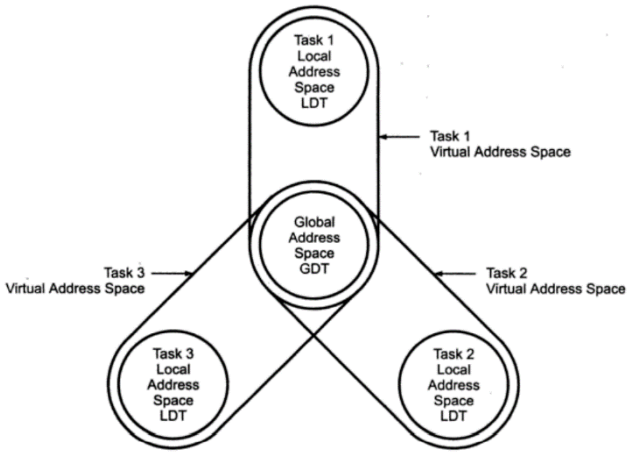


Fig. 4.12 Memory area shared by different tasks

Cada tabla tiene un registro asociado, los cuales contienen 32 bits de dirección lineal base y límite de tabla. Este límite indica cuantos descriptores tiene.

Global Descriptor Table Register GDTR

Define una GDT en memoria física. Contiene 48 bits, los 2 bytes menos significativos indican el límite y los demás la dirección base.

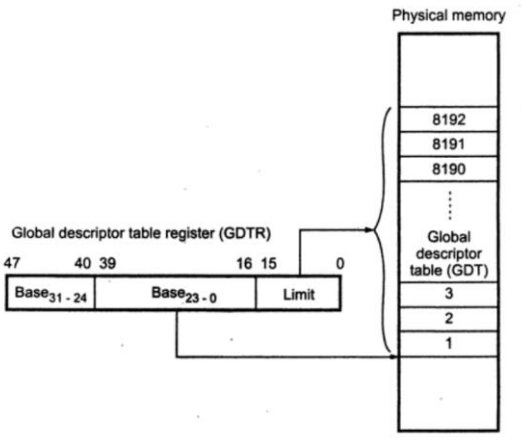


Fig. 4.13 GDTR and GDT

Interrupt Descriptor Table Register IDTR

Misma disposición que GDTR, por lo que podría tener 64536 bytes, pero como Pentium solo soporta 256 interrupciones, el tamaño no debe ser mayor a dicho valor.

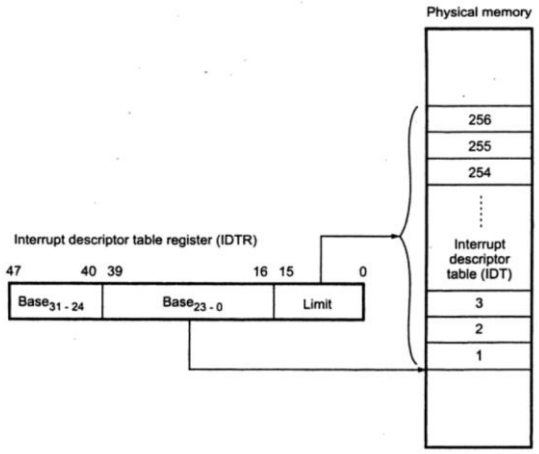


Fig. 4.14 IDTR and IDT

Local Descriptor Table Register LDTR

Es de 16 bits. No indica límite ni dirección base, solo la dirección del descriptor LDT almacenado en la GDT. Contiene un selector que apunta a un descriptor en la GDT, este descriptor define el inicio de la tabla con los 32 bits de dirección lineal base y el tamaño de la tabla con los 16 de límite.

La GDT puede tener varios descriptores LDT. Para poner en servicio uno en particular, debe cargarse el registro LDTR con el selector correspondiente.

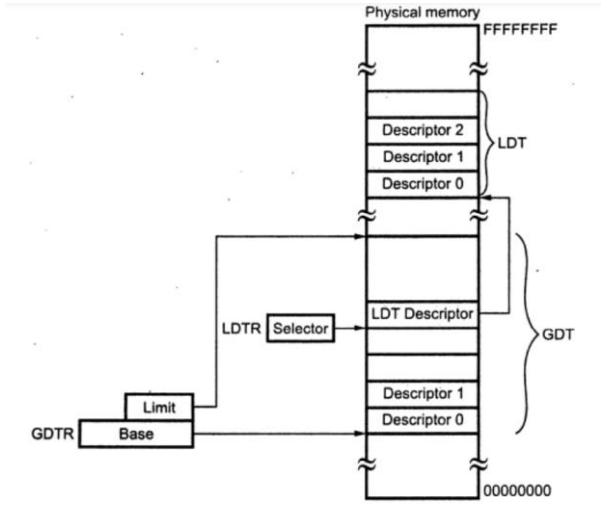


Fig. 4.15 Global and local descriptor tables

Para cargar valores en los registros se usan las instrucciones privilegiadas LGDT, LIDT y LLDT de 48 bits.

MÁS SOBRE REGISTROS DE SEGMENTO

La parte visible para el programador de los registros de segmento se usa como selector para los descriptores en una tabla de descriptores. La parte oculta es conocida como registro de caché de descriptor de segmento.

Esta caché es manipulada por el procesador. Una vez que los descriptores estén almacenados en caché, pueden realizarse referencias a ellos sin sobrecargar el descriptor.

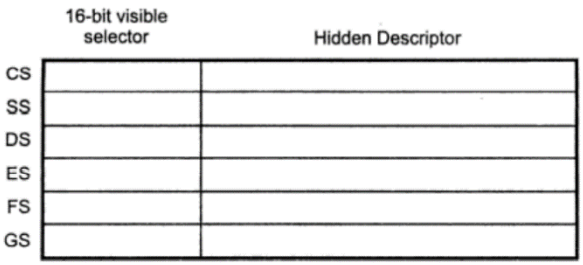


Fig. 4.16 Segment register and segment descriptor cache

Paginación (lineal → física)

Transforma la dirección lineal generada por la segmentación en dirección física. Es opcional y está habilitada solo si el bit PG de CR0 está seteado. Imprescindible si el SO va a implementar tareas virtuales de 8086, protección a memoria o memoria virtual orientada a página.

Al activarse, organiza el espacio físico en 1.048.496 páginas de 4 KB (4 TB).

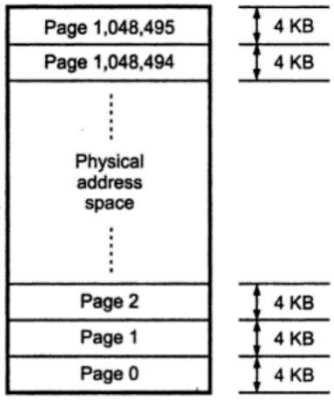


Fig. 4.18 Paged organization of the physical address space

REGISTROS DE SOPORTE Y TABLAS

Existen 3 componentes en la paginación Pentium:

- Directorio de páginas PD.
- Tabla de páginas PT (ubicadas en memoria especial).
- Marco de página (Page Frame).

Los PD y PT están hechos de descriptores de 32 bits. Cada uno con 1024 descriptores, así cada directorio o tabla tiene una longitud de 4 kB. Un marco de página es una unidad de 4 kB de direcciones contiguas en memoria física.

Al activarse la paginación, el procesador divide la dirección lineal en 3 campos: 2 de 10 bits y 1 de 12. Los 10 bits más significativos (Directory) se usan como índice en la PD, los siguientes 10 (Page) son un índice en la PT y los últimos 12 (Offset) seleccionan 1 de 4096 bytes de memoria desde el marco de página determinado por la PT.

La dirección física del PD es almacenada en el registro de control CR3.

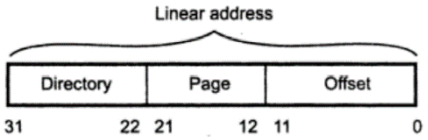


Fig. 4.19 Linear address format

Los descriptores del PD y el PT son conocidos como "entradas" (Entry), por lo que se denominan PDE y PTE respectivamente.

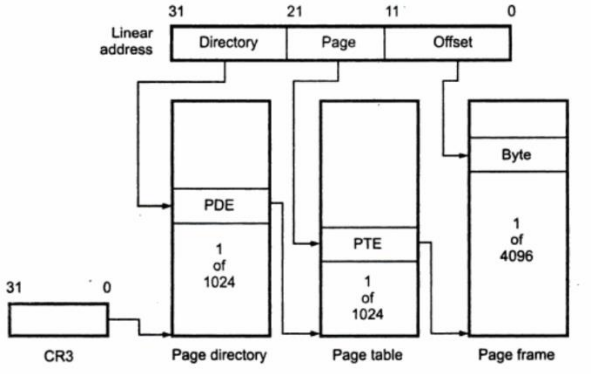


Fig. 4.20 Linear to physical address translation

DESCRIPTOR PDE

- **Page table address:** dirección física de inicio de la base de una tabla de página. Especifica los 20 bits más significativos de la dirección, ignorando los restantes 12, así localiza todas las PT en saltos de 4 kB ($2^{12} = 4kB$).
- **User/Avail:** disponible para usuario.
- **Accessed bit:** seteado si el PDE está en uso, solo puede borrarse manualmente.
- **Bits $\frac{U}{S}$ (User/$\overline{Supervisor}$) y $\frac{R}{W}$ (Read/$\overline{Write}$):** usados para protección a nivel de página.
- **P (Present bit):** indica si una PTE puede ser usada en la traducción de dirección (1) o si está ausente (0).

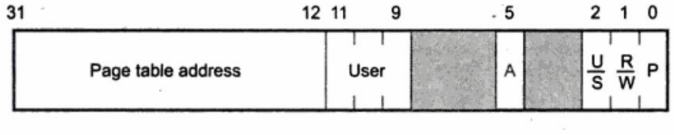


Fig. 4.21 Page directory entry

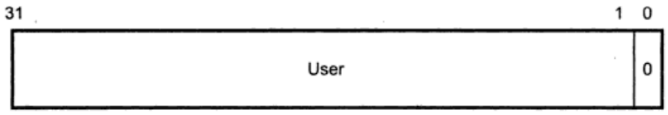


Fig. 4.22 Not present page descriptor

DESCRIPTOR PTE

- **Page frame address:** da el inicio de la memoria física de un marco de página de 4 KB.
- **User/Avail bits:** mismo funcionamiento que en el PDE.
- **Accessed bit:** seteado por el procesador si el PDE es usado. Puede hacerse seguimiento de páginas más usadas testeando y limpiando este bit.
- **Bits $\frac{U}{S}$ (User/$\overline{Supervisor}$) y $\frac{R}{W}$ (Read/$\overline{Write}$):** protección a nivel de página.
- **Present:** indica si una PTE puede ser usada.

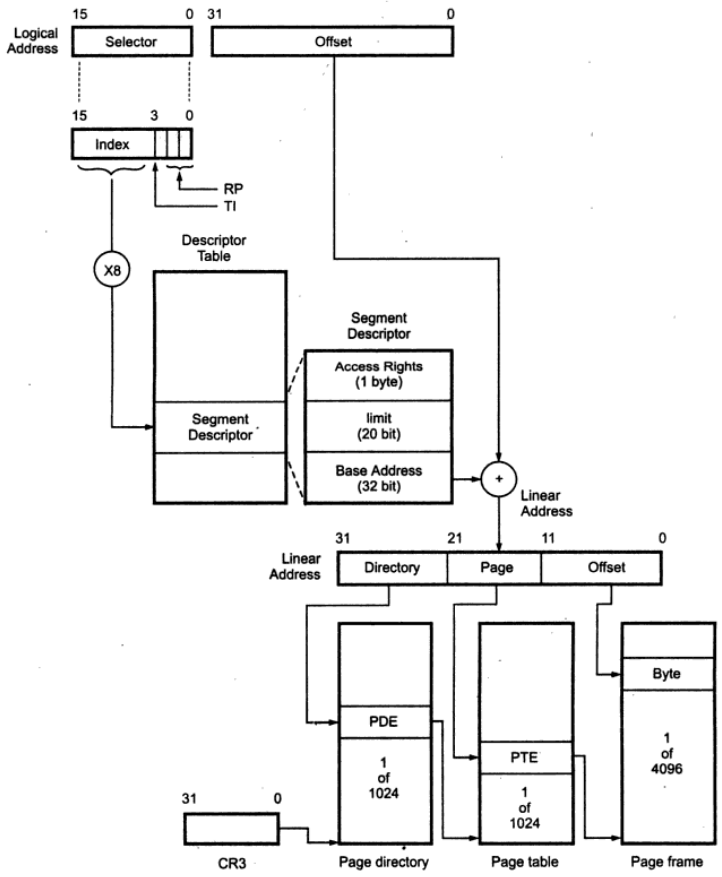


Fig. 4.24 Protected mode address translation

Buffer de traslación o cache de traslación de página TLB

El rendimiento del sistema de memoria virtual puede verse degradado debido al acceso a dos niveles de tablas para cada acceso a memoria (PDE y PTE). Pentium almacena las entradas a las PT más usadas en un caché, llamado TLB que contiene hasta 32 entradas de la PTE. Como las paginas son de 4 kB, da una cobertura de 128 kB de direcciones de memoria. Ahorra dos referencias a memoria. Para multitareas de sistemas comunes tiene una tasa de acierto del 98%.

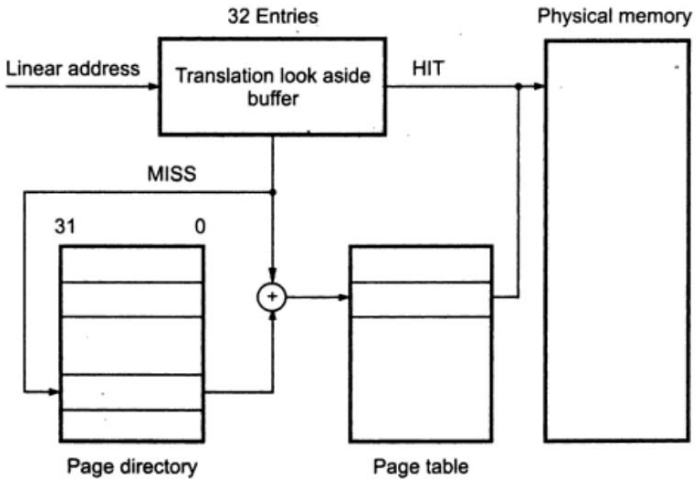


Fig. 4.25 Translation lookaside buffer

Operación de paginación

Desde la unidad de segmentación, el mecanismo de paginación recibe una dirección lineal de 32 bits. Los 20 bits más significativos se comparan con las 32 entradas del TLB. En caso de coincidencia, la dirección física de 32 bits es calculada con la ayuda del buffer y se sitúa en el bus de direcciones.

En caso contrario, se lee el PDE apropiado. Si $P = 1$, significa que la PT está en memoria, el procesador lee el PTE y setea el bit Access. Si $P = 1$ en el PTE, significa que la página está en memoria, actualiza los bits Access y Dirty del PTE y recupera el operando. Almacena los 20 bits de dirección lineal leídos desde la PT en el TLB para futuros accesos.

Si $P = 0$ en el PDE o en el PTE, se genera una excepción de fallo de página (14). La misma también se genera si se violan los atributos de protección de página.

Para actualizar el TLB se examinan las 32 entradas y se elimina los PTE menos usados para colocar el nuevo PTE (LRU).

Es necesario vaciar todo el cache siempre que las PT sean modificadas de la siguiente manera:

- Recargar CR3 con la instrucción MOV
    - MOV CR3, EAX
- Realizar cambio de tarea a un TSS con una imagen CR3 diferente a la actual.

PROTECCIÓN

En sistemas multiusuario o multitarea, pueden producirse problemas si más de 1 usuario o proceso cambia el contenido de una posición de memoria compartida a la vez. Esta posición se denomina

sección crítica y debe estar protegida contra accesos extraños hasta que se complete la operación. Otro tipo de protección se requiere además en el código del SO.

Protección a nivel de segmentos

Al intentar acceder a un segmento se comprueba:

- **Validez:** si el selector intenta acceder a una posición de memoria fuera del límite de la tabla de descriptores o si el selector apunta a un selector no válido se genera un excepción. El campo "limit" evita que los programas se dirijan fuera de los segmentos. El valor de este campo se interpreta dependiendo del valor del bit G. Para segmentos de datos, también se comprueba el bit de dirección de expansión ED y el bit "Big" B. Se genera una excepción general cuando:
  - Acceso a byte en dirección > limit.
  - Acceso a palabra en dirección ≥ limit.
  - Acceso a Dword[2] en dirección ≥ limit – 2.
- **Tipo:** si el selector apunta a un descriptor de tipo adecuado para ser cargado en la cache de registro de segmento especificada. El campo "type" indica si el segmento es de escritura (W), lectura (R), conforme (C), con acceso (A) y expandido hacia abajo (E). Por ejemplo, un descriptor de un segmento de datos de solo lectura no puede ser cargado en un registro de SS, porque la pila debe ser capaz de ser escrita.

Si se cumplen estas condiciones, los campos límite, base y Access Right Byte son copiados en la parte oculta de registro de segmento. Entonces se comprueba el bit P del Access Right Byte para verificar que el segmento se encuentre presente.

Protección a nivel de privilegios

Posee 4 niveles optimizados para SO multitarea para aislar y proteger programas de usuario entre sí y del SO. Se trata de niveles de privilegio de 0 a 3, donde el 0 indica el mayor privilegio.

El núcleo del SO tiene el mayor privilegio (PL0), los servicios del sistema, como la BIOS, PL1, los controladores de dispositivos PL2 y los programas de aplicación PL3.

Pentium asigna estos niveles de privilegio de descriptores y selectores en sus respectivos campos, del siguiente modo:

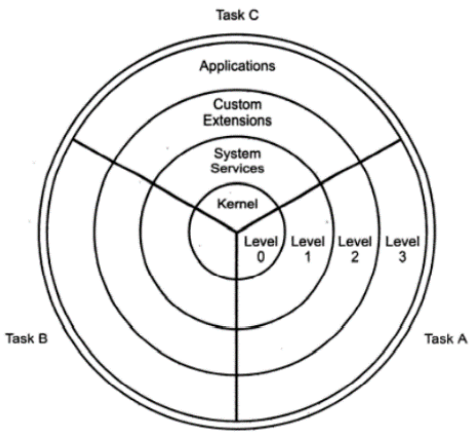


Fig. 4.26 Assignment of privilege levels

- Descriptores → **DPL**
- Selectores → **RPL** (PL del procedimiento que origina al selector)
- Segmentos en ejecución → **CPL** (PL del segmento en ejecución cuyo descriptor se encuentra en la cache interna, es decir, en la parte oculta del registro de segmento).

RESTRICCION DE ACCESO A DATOS

Para el acceso a segmentos de datos, se realizan 3 comprobaciones de PL:

- CPL (PL segmento en ejecución).
- RPL (PL del programa que genera el selector).

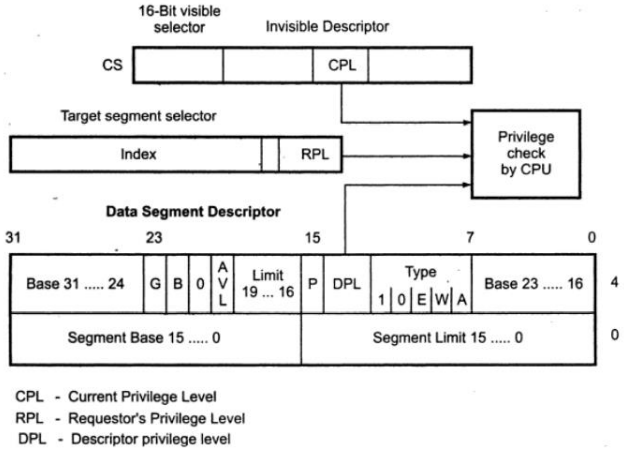
[2] Dword (Double Word) es una palabra de 16 bits.

- DPL (PL del descriptor de segmento de destino).

El programa (es decir, el selector) solo puede cargar un registro de segmento de datos si numéricamente se cumple:

$$DPL \geq \max(CPL, RPL)$$

Es decir, si el PL del segmento en ejecución o del selector es mayor o igual (menor numéricamente) que el PL del segmento de destino. En otras palabras, un proceso solo puede acceder a datos que están en el mismo nivel de privilegio o en un nivel inferior.

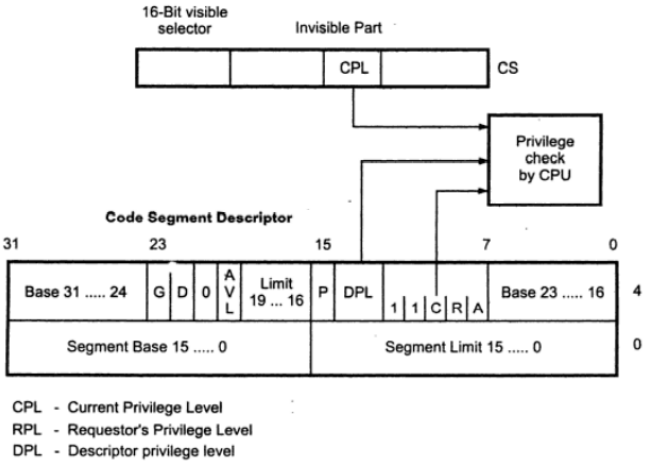


Nº.	DPL	CPL	RPL	Access
1	2	0	1	Válido
2	3	1	2	Válido
3	1	1	0	Válido
4	1	**2**	0	No válido
5	2	2	**3**	No válido

ACCESO A DATOS EN SEGMENTOS DE CÓDIGO

Existen 3 formas de leer datos de segmentos de código:

- Cargar un registro de segmento de datos con un selector de un segmento no conforme, de lectura y ejecutable. (Debe cumplirse la condición de privilegio)
- Cargar con un selector conforme, de lectura y ejecutable (siempre válido, PL del segmento conforme igual al del RPL).
- Usar prefijo de anulación de CS, para leer un segmento de lectura y ejecutable cuyo selector está cargado en el registro CS (siempre válido porque el DPL por definición es igual al CPL).



Transferencia de control entre niveles de privilegio

Existen 2 formas de acceder a un procedimiento en un segmento con un PL mayor:

- Si dicho segmento es un segmento de código conforme.
- Usando un Call Gate.

SEGMENTO DE CÓDIGO CONFORME

Posee el bit 2 del Access Right Byte seteado. No tiene nivel de privilegio propio sino que copia el PL del código que hace CALLs o JMPs a ellos.

Al transferir el control a este tipo de segmento, el RPL del registro CS no se modifica para que coincida con el DPL del segmento.

Si bien no tienen PL, tiene una restricción respecto a cuando pueden usarse:

N.º	CPL	DPL del segmento de código conforme	Acceso
1	3	2	Válido
2	2	0	Válido
3	1	1	Válido
4	**1**	**2**	No válido
5	**2**	**3**	No válido

Nunca puede transferirse el control a un segmento cuyo DPL es mayor (menos privilegiado) que el CPL, así se permite que el control regrese al segmento original.

CALL GATE

Un Call Gate es un tipo especial de descriptor, el cual no define ningún espacio de memoria, no poseen campos de dirección base o limite. Aunque no es del todo un descriptor, conviene almacenarlo en tablas de descriptores.

Actúa como una capa de interface entre segmentos de código en diferentes niveles de privilegio. Es el único mecanismo que permite llamar a un proceso ubicado en cualquier segmento (conforme o no conforme) que tiene un nivel de privilegio superior.

No se permite hacer JMPs a Call Gates y debe tenerse en cuenta que la instrucción CALL debe referirse al Call Gate no al segmento de código de destino. El Call gate define el segmento de código y el offset exacto donde el control es transferido. Los usuarios no pueden especificar el offset deseado en sus programas por seguridad.

Los descriptores Call gate son almacenados en la GDT o en un LDT, al igual que los segmentos y otros descriptores. Cuando un programa hace un CALL a un procedimiento en otro segmento, el selector de Call gate de dicho segmento es ubicado en la parte visible del registro CS y el descriptor Call gate en la parte no visible.

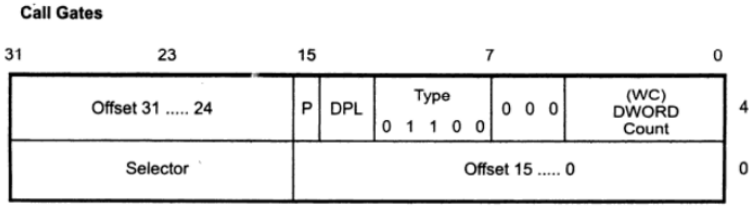


Fig. 4.29 Format of Pentium processor call gate

El descriptor Call gate contiene 2 cosas importantes:

- Selector que apunta al descriptor de segmento donde el procedimiento está actualmente cargado.
- Offset del procedimiento llamado en su segmento.

Si el CALL es válido, el selector del Call gate es colocado en la parte visible del registro CS y el correspondiente descriptor de segmento es cargado en la porción no visible del registro CS. El procesador luego usa la dirección base del descriptor de segmento y el offset del descriptor Call gate para calcular la dirección física del procedimiento llamado.

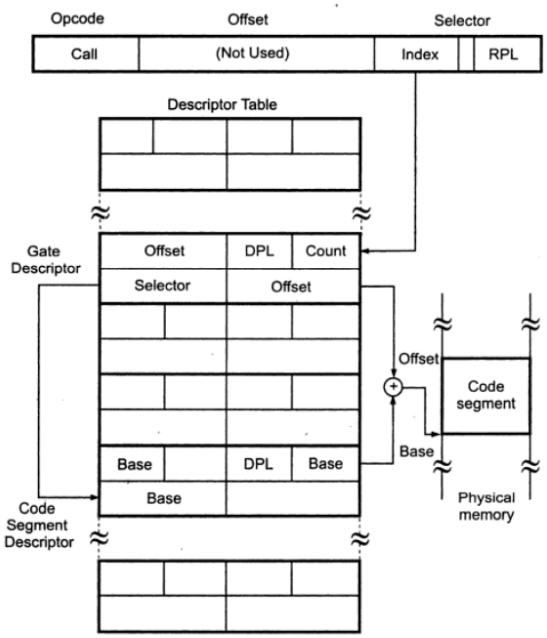


Fig. 4.30 Indirect transfer via call gate

En resumen, los Call Gates:

- Se definen como descriptores de segmento.
- No definen espacio de memoria.
- Se ubican en tablas de descriptores.
- Son el único medio para alterar el CPL.
- Definen puntos de entrada a otros niveles de privilegio.
- Se invocan desde instrucciones CALL.

Durante este proceso, la validez de la transferencia de control se chequea usando 4 niveles de privilegio diferentes:

1. CPL
2. RPL del selector usado para especificar el Call gate.
3. DPL del descriptor de gate.
4. DPL del descriptor del segmento ejecutable de destino.

Para que la transferencia de control sea válida, debe cumplirse numéricamente para la instrucción CALL:

$$DPL_{destino} \leq \max(RPL, CPL) \leq DPL_{gate}$$

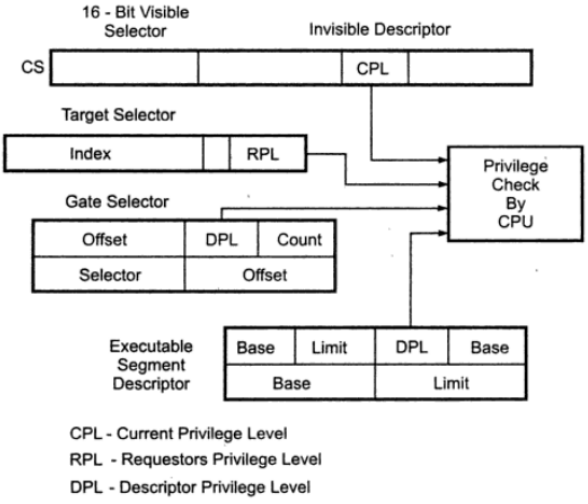


Fig. 4.31 (a) Privilege check via call gate

Para usar un Call gate, debe cumplirse numéricamente:

- $DPL_{call\ gate} \geq CPL$
- $DPL_{call\ gate} \geq RPL_{selector\ gate}$
- $DPL_{call\ gate} \geq DPL_{segmento\ codigo\ destino}$
- $DPL_{segmento\ codigo\ destino} \leq CPL$

Mediante los Call gates se accede al procedimiento indirectamente a través del descriptor Call gate. Este acceso indirecto tiene 2 grandes ventajas:

1. Permite otro nivel de comprobación de privilegios (compara $DPL_{call\ gate}$ y CPL).
2. Los programas de usuario no pueden accidentalmente entrar a segmentos más privilegiados. Para ingresar deben ingresar el offset especifico contenido en los descriptores Call gate.

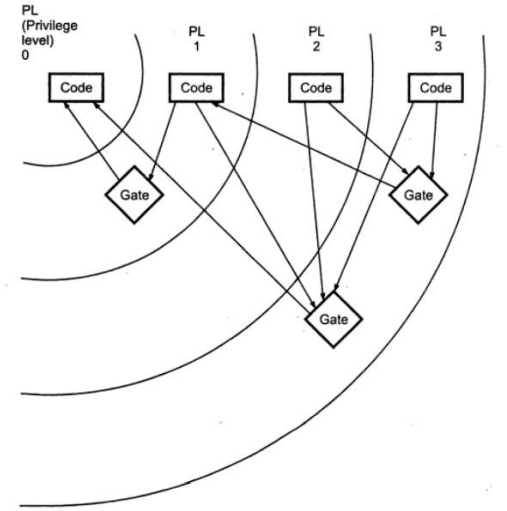


Fig. 4.31 (b) Some valid accesses to higher privileged levels using call gates

CAMBIO DE PILAS

EL cambio de PL modifica el dominio de direcciones del programa. El procesador cambia además las pilas en caso de cambio de PL. Si el Call gate provoca un cambio de PL, el segmento de pila y el puntero son guardados y una nueva pila correspondiente al nuevo PL interno es usada. Cuando el control vuelve al código de nivel exterior, se reestablece el uso de la pila original.

En presencia de un Call gate válido, el procesador usa una nueva pila, tomando el selector y el puntero de esta del TSS. Si el usuario está llamando a un procedimiento con PL1, el nuevo selector y puntero de pila se tomarán desde SS1 y ESP1, respectivamente.

El antiguo selector y puntero de pila son pusheados a esta nueva pila. Luego, el procesador busca el número de entradas de Dwords a ser pusheadas desde la antigua pila a la nueva en el campo WC del descriptor Call gate. El campo WC (contador de palabras) define el número de parámetros de paso a la nueva pila. El nuevo selector CS y el offset EIP son pusheados a la nueva pila.

Finalmente, CS es cargado desde el campo selector del descriptor Call gate, EIP es cargado desde el campo offset y la ejecución comienza en la nueva dirección.

Protección a nivel de página

Implica dos tipos de protecciones: restricción de dominio direccionable y comprobación de tipos.

- Restricción de dominio direccionable:
  - $\frac{U}{S} = 0 \rightarrow$ nivel supervisor: SO y otro software de sistema y datos relacionados. Todas las paginas son direccionables.
  - $\frac{U}{S} = 1 \rightarrow$ nivel usuario: solo las páginas pertenecientes a nivel usuario son direccionables.
- Comprobación de tipos: se definen 2 tipos de acceso:
  - $\frac{R}{W} = 0 \rightarrow$ acceso de solo lectura.
  - $\frac{R}{W} = 1 \rightarrow$ acceso de solo escritura.

Al ejecutarse a nivel de supervisor, todas las páginas se asignan con acceso de lectura/escritura, mientras que en nivel usuario esto depende del bit $\frac{R}{W}$ en los campos de PDE y PTE. Además, en nivel usuario no puede accederse a páginas de nivel supervisor.

Instrucciones privilegiadas

Afectan al mecanismo de segmentación y protección, alteran la bandera de interrupción o realizan I/O periféricos. Se dividen en:

- **Instrucciones privilegiadas**: afectan a la estructura de datos del sistema. Deben ejecutarse cuando CPL = 0, de otro modo se genera una excepción de protección general.
- **Instrucciones IOPL – sensibles:** el campo IOPL en el registro FLAG define el derecho a usar instrucciones I/O relacionadas. Para ejecutarlas, el CPL del procedimiento debe ser igual o menor que el numero representado por los bits IOPL.

Paginación bajo demanda – Entorno virtual

El hardware de paginación posee 3 capacidades principales:

1. Traslación de direcciones.
2. Protección a nivel de páginas.
3. Paginación bajo demanda.

Esta última permite crear entornos virtuales para los programas. Ni el código de programa ni el programador necesitan saber cuánta memoria RAM está realmente disponible en el sistema o donde está ubicada. Si el programa hace referencia a algo que no está en memoria principal, el procesador

llamará a un manejador de fallo de página. Usando esta rutina, recupera los datos deseados de un almacenamiento secundario (como un disco) y los sitúa en la memoria. El contenido previo de la memoria es intercambiado con los datos del disco, de esta forma es posible crear la impresión de un sistema con gran cantidad de memoria principal (memoria virtual). Su tamaño real y ubicación nunca son conocidos por el programa o programador, pero todo se ejecuta como se desea.

Paso a modo protegido

Se necesita al menos una GDT y una IDT definidas en el sistema. La IDT debe tener al menos 256 bytes y la GDT al menos un segmento de código y uno de datos.

Los siguientes pasos logran el cambio de modo real a protegido:

1. Preparar GDT con un descriptor NULL en su primer entrada, un descriptor de segmento de código uno de datos y uno de pila.
2. Inicializar la IDT para que contenga puertas de interrupción válidas para al menos 32 tipos de interrupciones.
3. Cargar la dirección base y el límite de GDT en GDTR con la instrucción LGDT.
4. Establecer la bandera PE en CR0 usando la instrucción MOV CR0 o LMSW.
5. Ejecutar un FAR JUMP para cargar el registro CS y vaciar la cola de decodificación de instrucciones.
6. Cargar todos los registros de segmento de datos con los valores iniciales de selector.

Las siguientes figuras muestran las tablas y descriptores necesarios para el modo protegido simple. En este modo el sistema tiene un solo segmento de código y un solo segmento de datos/pila de 4GB cada uno además de un único nivel de privilegio PL0.

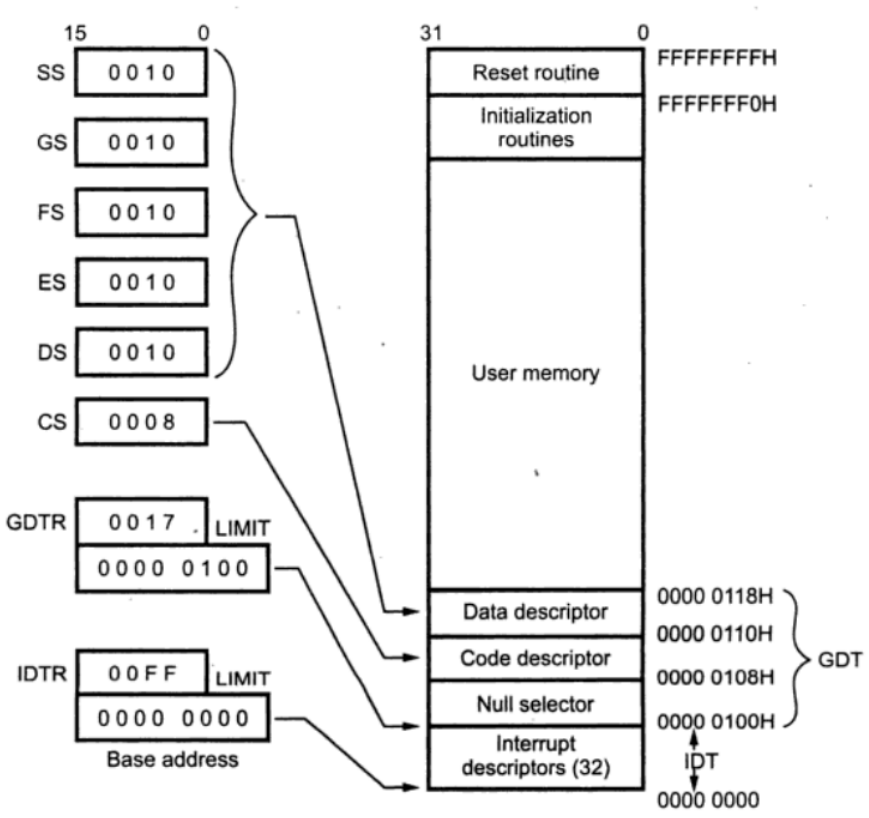


Fig. 4.32 (a) Simple protected system

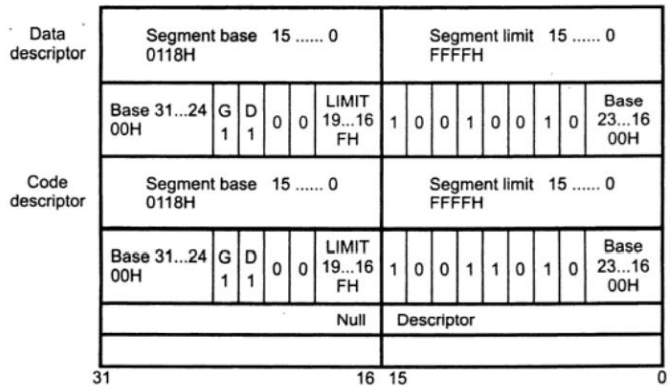


Fig. 4.32 (b) GDT descriptor for simple system

Un enfoque alternativo para entrar en modo protegido en sistemas multitarea es usar el conmutador de tareas para cargar los registros. En dicho caso GDT debe contener 2 descriptores TSS. El procedimiento es:

1. Inicializar la IDT para al menos 32 tipos de interrupciones.
2. Inicializar la GDT para al menos 2 descriptores de segmento TSS, además de los mencionados anteriormente.
3. Inicializar el TR para que apunte a un TSS válido.
4. Cambiar a modo protegido mediante un JMP intersegmento para cargar el registro CS y vaciar la cola del decodificador de instrucciones.

Regreso a modo real

1. Si la paginación está habilitada:
    a. Transferir el control a las direcciones donde las lineales son iguales a las físicas.
    b. Borrar bit de paginación PG en CR0.
    c. Cargar ceros en CR3 para borrar la cache de paginación.
2. Transferir el control a un segmento de límite 64 kB para asegurar que el contenido del registro CS esté dentro del límite necesario para modo real.
3. Cargar registros de segmento SS, DS, ES, FS y GS con un selector que apunte a un descriptor con los siguientes valores:
    a. Base: cualquier valor
    b. Limit: 64K (FFFFH)
    c. Present: P = 1
    d. De escritura: W = 1
    e. Expandido hacia arriba: E = 0
    f. Granularidad: G = 0
4. Desactivar interrupciones.
5. Borrar bit PE.
6. Vaciar cola de instrucciones ejecutando un Far JMP al código de modo real.
7. Cargar base y limit de la tabla de vectores de interrupción en modo real / IDT en modo protegido con la instrucción LIDT.
8. Habilitar las interrupciones.
9. Cargar los registros de segmento según lo requiera el código de modo real.

Interrupciones y excepciones

Son eventos que indican que existe una condición en algún lugar del sistema, procesador o en el programa en ejecución actual o tarea que requiere atención del procesador. Típicamente resultan en una transferencia forzada de ejecución desde el programa o tarea actualmente en ejecución a una rutina o tarea de software especial denominada **manejador de interrupción** o **manejador de excepción**.

Las interrupciones ocurren típicamente en tiempos aleatorios durante la ejecución de una programa, en respuesta a señales del hardware. El hardware de sistema usa interrupciones para manejar eventos externos al procesador, como peticiones de servicio de dispositivos periféricos.

Las excepciones ocurren cuando el procesador detecta una condición de error mientras ejecuta una instrucción, como la división por cero. El procesador detecta una variedad de condiciones de error incluyendo violación de protección, fallo de página y fallas internas de máquina.

El mecanismo de manejo de interrupciones y excepciones de la arquitectura IA-32 permite a las interrupciones y excepciones ser manejadas transparentemente a los programas de aplicación y el SO. Cuando una interrupción es recibida o una excepción es detectada, el proceso actualmente en ejecución es suspendido mientras el procesador ejecuta un manejador de interrupción o excepción. La reanudación del procedimiento o tarea interrumpida ocurre sin pérdida de continuidad del programa, a menos que la recuperación de una excepción no fuera posible o una interrupción causara la finalización del programa que se estaba ejecutando actualmente.

MULTITAREA

- **Tiempo compartido:** en sistemas multiusuario, el ordenador atiende a un solo usuario durante un breve periodo de tiempo. A cada usuario le asigna una franja de tiempo.
    - Permite a varios usuarios usar el mismo computador.
    - Proporciona un uso económico de los recursos de procesamiento.
    - Es invisible para usuarios.
    - Permite cualquier número de usuarios.

  Se denomina SO multiusuario e implica multitarea.

- **Multiusuario frente a multitarea:**
    - Multiusuario: gran cantidad de usuarios para un ordenador. Puede realizar gran cantidad de tareas para gran cantidad de usuarios.
    - Multitarea: gran cantidad de tareas en un solo ordenador. Puede realizar gran cantidad de tareas para un solo usuario.

Métodos de programación para SO multiusuario

Existen dos enfoque diferentes: programación por tramos de tiempo y programación basada en prioridad de usuarios.

- **Programación por tramos de tiempo:** el planificador (componente del SO) determina cuándo es momento de cambiar de tarea. La ventaja es que todos los usuarios son atendidos en intervalos de tiempo aproximadamente iguales. Si el N.º de usuarios crece, el tramo de tiempo de cada uno disminuye, degradándose el sistema.

- **Programación basada en prioridad de usuarios:** a cada tarea se asigna un número de prioridad. Las de mayor prioridad pueden interrumpir a las de menor prioridad. Luego de la interrupción se devuelve el control a la tarea de menor prioridad.

Cada tarea usa registros, punteros de datos, de memoria, área de pila de variables de memoria, etc. Esto es denominado **contexto de tarea**. Para un cambio de tarea, el contexto de la tarea interrumpida se guarda para que la misma luego pueda retomarse. Este contexto y el puntero al mismo se guardan en un segmento de memoria especial o en una pila. Para regresar a la tarea interrumpida, el SO usa el puntero para acceder al contexto guardado. Esto se conoce como **cambio de contexto**.

Registros de soporte y descriptores para multitarea

El procesador posee registros y descriptores especiales para soportar sistemas multitarea eficientes y protegidos, estos son:

- TSS.
- Descriptor de TSS.
- TR.
- Descriptor de Task Gate.

Con estos registros y estructuras de datos, Pentium cambia la ejecución de una tarea a otra, guardando el contexto de la tarea actual. Por lo tanto, la tarea puede continuar más tarde.

Además de la conmutación, admite dos funciones más de gestión de tareas:

- Interrupciones y excepciones pueden provocar cambios de tareas. El procesador no solo cambia automáticamente a la tarea que maneja la interrupción o excepción, además cambia automáticamente de vuelta a la tarea interrumpida cuando la interrupción o excepción ya ha sido atendida.
- Como cada tarea tiene LDT y directorio de paginas separado, pueden tener diferentes asignaciones lógicas a lineales y lineales a físicas. Debido a esto las tareas pueden ser aisladas y prevenir la interferencia con otra.

SEGMENTO DE ESTADO DE TAREA TSS

Es usado por el procesador como un block de notas donde almacena el contexto de la tarea. No es accesible al programa de usuario general, ni siquiera con nivel de privilegio 0. Los campos internos del TSS son solo accesibles para Pentium y se dividen en 2 conjuntos: dinámico y estático.

- **Conjunto dinámico:** se actualiza al pasar de una tarea a otra. Incluye:
    - Registros generales.
    - Registros de segmento.
    - EFLAGS.
    - Puntero de instrucción EIP.
    - Back Link.

  Los primeros 4 guardan el estado del microprocesador. Guardar el EIP garantiza que la tarea pueda reiniciarse donde se detuvo y los EFLAGS permiten que Pentium ejecute las instrucciones condicionales cuando se reinicia.

  El Back Link mantiene el seguimiento de una tarea anterior. Al ejecutar una instrucción de retorno al final de una nueva tarea, el selector Back Link para el TSS anterior es cargado

automáticamente en el TR, esto activa la tarea previa y reestablece el entorno del programa previo.

- **Conjunto estático:** el procesador solo lee campos de este conjunto. Estos son:
  - El selector para la LDT de la tarea.
  - El registro PDBR que contiene la dirección base del directorio de página de la tarea.
  - Punteros a las pilas para PL 0 – 2.
  - El bit T (debug trap bit) que provoca que Pentium lance una excepción de depuración al producirse un cambio de tarea.
  - El offset del mapa de I/O.

  La conmutación de tareas puede modificar el PL, en cuyo caso el procesador debe cambiar la pila. Es por ello que el puntero y segmento de pila anterior son abandonados y se usa una nueva pila que corresponde al PL nuevo. Cuando se devuelve el control al nivel anterior se restaura la anterior pila.

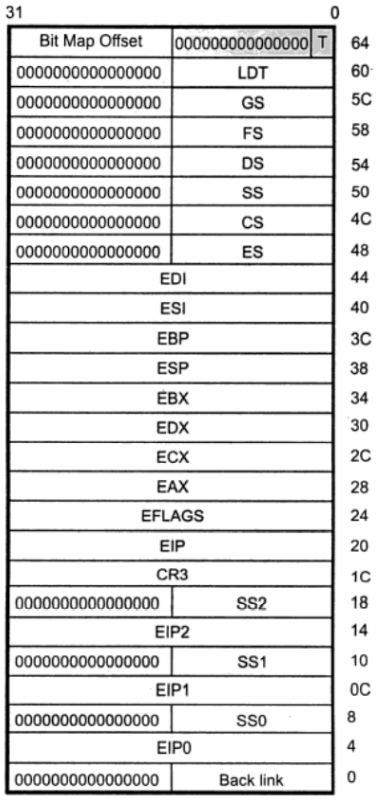


Fig. 5.1 Task state segment

En resumen:

TSS	
Conjunto dinámico	Conjunto estático
• Registros • EFLAGs • EIP • Back Link	• Selector LDT • PDBR • Puntero a pila • bit T • Offset mapa I/O

DESCRIPTOR TSS

El bit B (busy) del campo type indica si la tarea está ocupada. Las tareas no son redundantes, este bit permite detectar un intento de cambiar a una tarea que ya está ocupada. Los demás campos tienen funciones similares a la de los descriptores vistos. El campo limit debe tener un valor igual o superior a

103 ya que Pentium requiere un mínimo de 104 bytes de almacenamiento para poder realizar el guardado del contexto de tarea. Se requiere un limite aun mayor si el mapa de permisos de I/O está presente. El límite máximo para el TSS es de 4 GB.

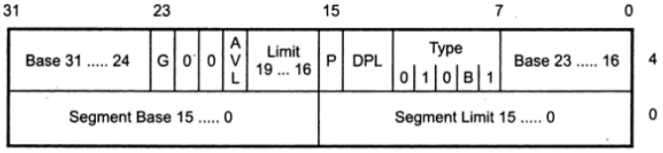


Fig. 5.2 Task state segment descriptor

Para poder acceder al descriptor de TSS, el proceso debe tener un nivel de privilegio menor o igual numéricamente al PL especificado por el campo DPL del descriptor del TSS. Este acceso suele estar restringido a programas de confianza, cuyo PL = 0.

Registros de tarea TR

Especifica la tarea que está en ejecución apuntando al TSS. Es un selector para el TSS.

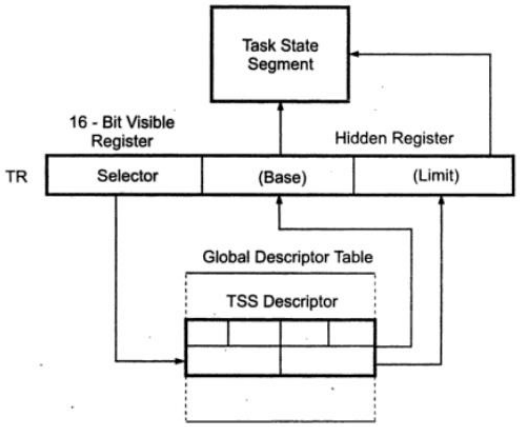


Fig. 5.3 Task register

Posee una parte visible que puede ser leída y modificada mediante instrucciones y una parte no visible. El selector en la parte visible es usado para especificar un descriptor TSS en la GDT y la parte no visible es usada para almacenar en caché los valores de base y límite desde el descriptor TSS. Esto último hace que la ejecución de la tarea sea más eficiente.

El procesador provee dos instrucciones para leer y modificar la parte visible: LTR y STR.

Task Gate y Descriptor Task Gate

Se trata de Gates de sistema especiales. Tienen su propio descriptor, el cual actúa como un punto de interfaz entre el código de usuario y el TSS. Proporciona una referencia indirecta y protegida al TSS.

El descriptor define un selector a un descriptor TSS que identifica una única tarea. El selector puede ser usado en lugar de un selector a un segmento de código en las instrucciones FAR JMP y FAR CALL.

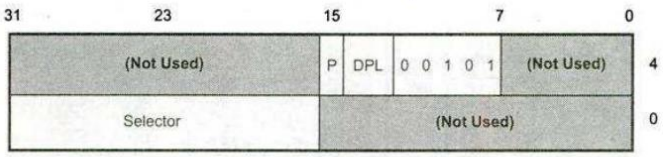


Fig. 5.4 Task gate descriptor

El campo DPL controla el derecho a usar el descriptor para cambiar de tarea. El descriptor selecciona un descriptor de Task Gate si el máximo entre el RPL del selector y el CPL del procedimiento es numéricamente menor o igual que el DPL del descriptor.

$$\max(CPL_{procedimiento}, RPL_{selecto}) \leq DPL_{Task\ Gate}$$

Si $DPL = 0$, se impide que procedimientos no confiables provoquen cambios de tarea. Sin embargo, a través de Task Gates podemos cambiar de un privilegio más bajo a uno más alto, ya que al usarlas el DPL del descriptor TSS de destino no se usa para la comprobación. Por ello, todo procedimiento con acceso a un Task Gate puede causar un cambio de tarea.

Cambios de tarea

Luego de cada cambio de tarea, es decir, luego de cargar un nuevo contexto de tarea desde el TSS y actualizar el TR, el procesador marca el nuevo TSS como "Busy" (ocupado). Por lo tanto, la tarea actual en ejecución siempre es una tarea ocupada. Como no puede realizarse un cambio de tarea a una tarea ocupada, las tareas no son reentrantes, es decir, los cambios de tarea no son recursivos.

El procesador realiza cambios de tarea en cualquiera de los siguientes casos:

- LONG JMP o CALL con un selector que referencia a un descriptor TSS.
- Selector en una instrucción LONG JMP o CALL referido a un Task Gate.
- Selector de interrupción referido a un Task Gate en la IDT.
- Ejecución de instrucción IRET (retorno de interrupción) con el bit NT del registro EFLAGS activado.

CONMUTACIÓN DE TAREAS SIN TASK GATES

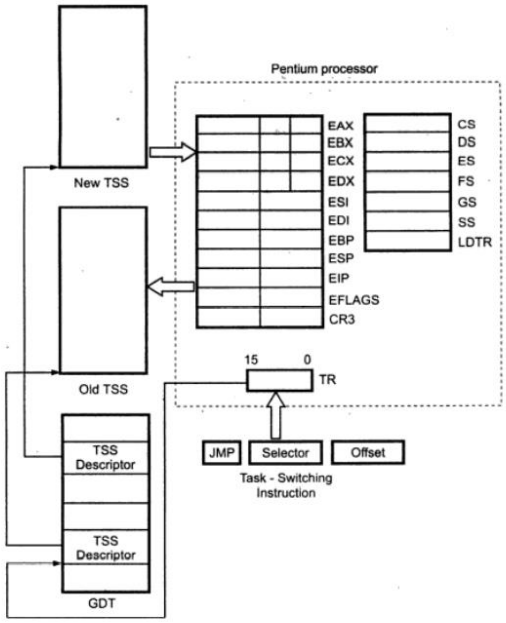


Fig. 5.5 Task switch operation

Requiere los siguientes pasos:

1. Comprobación de privilegios.
2. Comprobación de bits de límite y P.
3. Guardado de TSS actual.
4. Carga del TR.
5. Reanudación de ejecución.

El entorno del programa previo se preserva guardando el selector de dicho TSS como selector Back Link del nuevo TSS.

CONMUTACION DE TAREAS CON TASK GATES

Se utiliza el método indirecto, en el cual la conmutación se realiza mediante un JMP o CALL a un Task Gate. Consiste en los siguientes pasos:

1. El DPL del nuevo descriptor no se usa, se usa el DPL del Task Gate comparándolo con el CPL y RPL de selector de Gate.
2. Los pasos siguientes son similares, salvo que para cargar el selector del descriptor TSS en el TR se remita a la instrucción CALL o JMP.

En caso de excepciones, interrupciones e IRETs, se permite el cambio de tarea.

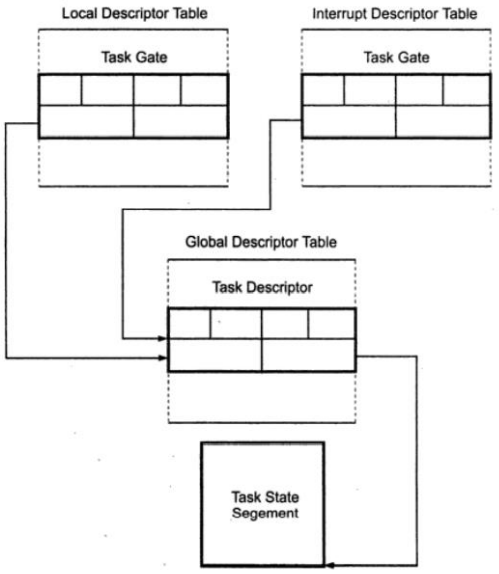


Fig. 5.6 Task switching through task gate

Tareas anidadas

Si el cambio de tarea fue por una instrucción FAR CALL o una excepción, falla o trampa, la nueva tarea se considera anidada a la antigua. Cuando se ejecuta una instrucción IRET el procesador vuelve a la tarea que la invocó. Para ello existe un mecanismo de vinculación de tareas, el cual consiste en el **Back Link** y la bandera de tarea anidada **NT**. Pentium usa el campo NT como bandera para saber si el campo Back Link de la TSS actual es válido. Este es el único medio por el que Pentium determina si debe realizar un cambio de tarea o un IRET normal.

Cambios en tareas anidadas:

- Las tareas anidadas actúan como subrutinas.
- CALL a Task Gate anida tareas.
- **Interrupción** o **Excepción** a Task Gate anida tareas.
- JMP **no** anida tareas.
- La nueva TSS obtiene el selector de la antigua TSS en **Back Link**.
- La nueva tarea obtiene el bit de tarea anidada en el registro **EFLAGS**.
- La nueva tarea debe volver a la tarea anterior con la instrucción **IRET**.