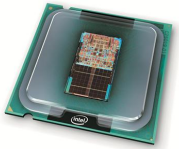


Procesadores IA-32 e Intel® 64

Gestión de Memoria: Segmentación

Alejandro Furfaro

7 de octubre de 2025

Temario

Dirección Lógica, Virtual, Física.... ¿?

Dirección Lógica, Virtual, Física.... ¿?

- ¿Como es que hay diferentes definiciones de direcciones?. ¿No son todas lo mismo?

Dirección Lógica, Virtual, Física.... ¿?

- ¿Como es que hay diferentes definiciones de direcciones?. ¿No son todas lo mismo?
- No siempre.

Dirección Lógica, Virtual, Física.... ¿?

- ¿Como es que hay diferentes definiciones de direcciones?. ¿No son todas lo mismo?
- No siempre.
- En los microcontroladores sencillos o de arquitecturas muy primitivas, estos conceptos significan lo mismo

Dirección Lógica, Virtual, Física.... ¿?

- ¿Como es que hay diferentes definiciones de direcciones?. ¿No son todas lo mismo?
- No siempre.
- En los microcontroladores sencillos o de arquitecturas muy primitivas, estos conceptos significan lo mismo
- Pero a medida que los procesadores ganan complejidad en su arquitectura, y soportan mayor grado de funcionalidades y aspiran a sostener Sistemas Operativos con ejecución dinámica de tareas, estos tres valores suelen ser diferentes.

Dirección Física

Dirección Física

- Cuando cualquier procesador necesita acceder a memoria, deberá escribir en el buffer de direcciones el valor correspondiente a la dirección de memoria externa que debe acceder.

Dirección Física

- Cuando cualquier procesador necesita acceder a memoria, deberá escribir en el buffer de direcciones el valor correspondiente a la dirección de memoria externa que debe acceder.
- Este valor se conoce como Dirección Física, ya que corresponde a la dirección que será decodificada por el hardware externo para acceder a la memoria RAM o ROM según corresponda. Es decir a la memoria física.

Dirección Física

- Cuando cualquier procesador necesita acceder a memoria, deberá escribir en el buffer de direcciones el valor correspondiente a la dirección de memoria externa que debe acceder.
- Este valor se conoce como Dirección Física, ya que corresponde a la dirección que será decodificada por el hardware externo para acceder a la memoria RAM o ROM según corresponda. Es decir a la memoria física.
- Siempre el procesador pone la Dirección Física en los pines de address cuando su unidad de control habilita la salida del bus de direcciones

¿Que dirección manejamos en los programas?

¿Que dirección manejamos en los programas?

- El texto de nuestros programas fuente provee implícitamente una visión mas o menos abstracta de la memoria.

¿Que dirección manejamos en los programas?

- El texto de nuestros programas fuente provee implícitamente una visión mas o menos abstracta de la memoria.
- En mayor o menor medida intentamos abstraernos del valor numérico de la dirección, aun cuando programamos los procesadores mas rudimentarios

¿Que dirección manejamos en los programas?

- El texto de nuestros programas fuente provee implícitamente una visión mas o menos abstracta de la memoria.
- En mayor o menor medida intentamos abstraernos del valor numérico de la dirección, aun cuando programamos los procesadores mas rudimentarios

```
1 buffer  resb 1024 db     ;define 1024 bytes de memoria en 0
2     .....
3 mov al,00        ; valor inicial
4 mov ecx,1024     ; inicializa contador en 1024
5 mov esi,buffer   ; guarda en esi la direción de memoria de buffer
6 repnz stosb      ; Accede al buffer y lo inicializa
```

En este código se define un buffer de 1024 bytes.

¿Que dirección manejamos en los programas?

- El texto de nuestros programas fuente provee implícitamente una visión mas o menos abstracta de la memoria.
- En mayor o menor medida intentamos abstraernos del valor numérico de la dirección, aun cuando programamos los procesadores mas rudimentarios

```
1 buffer  resb 1024 db     ;define 1024 bytes de memoria en 0
2     .....
3 mov al,00        ; valor inicial
4 mov ecx,1024     ; inicializa contador en 1024
5 mov esi,buffer   ; guarda en esi la direción de memoria de buffer
6 repnz stosb      ; Accede al buffer y lo inicializa
```

En este código se define un buffer de 1024 bytes.

- ¿Cual es su dirección de Memoria?

Dirección Lógica

Definición

Dirección Lógica

Definición

- Es la dirección de memoria expresada en términos abstractos por el programador en su código fuente.

Dirección Lógica

Definición

- Es la dirección de memoria expresada en términos abstractos por el programador en su código fuente.
- Es una forma muy simple de trabajar en forma transparente a la memoria.

Dirección Lógica

Definición

- Es la dirección de memoria expresada en términos abstractos por el programador en su código fuente.
- Es una forma muy simple de trabajar en forma transparente a la memoria.
- En lugar de referirse al valor numérico de la dirección de hardware, reemplazamos este valor por una etiqueta que incluso resulte ilustrativa de lo que representa esa dirección, y dejamos que el toolchain resuelva su valor numérico.

¿Que es la dirección virtual?

¿Que es la dirección virtual?

- Generalmente coincide con la dirección lógica (Aun en procesadores de alta gama).

¿Que es la dirección virtual?

- Generalmente coincide con la dirección lógica (Aun en procesadores de alta gama).
- Sin embargo, en algunos procesadores las direcciones lógica y la virtual son diferentes.

¿Que es la dirección virtual?

- Generalmente coincide con la dirección lógica (Aun en procesadores de alta gama).
- Sin embargo, en algunos procesadores las direcciones lógica y la virtual son diferentes.
- Inclusive encontraremos que se refieren a ella como dirección lineal.

Memory Management Unit

Memory Management Unit

- La MMU cambia el mapeo entre una Dirección Lógica y la Dirección Física.

Memory Management Unit

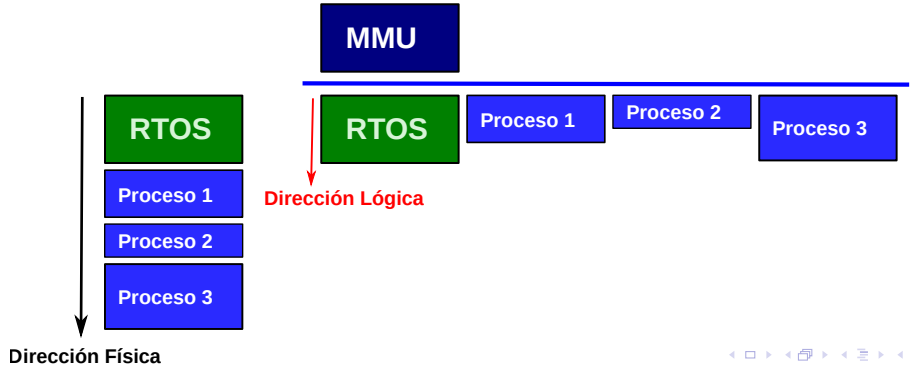
- La MMU cambia el mapeo entre una Dirección Lógica y la Dirección Física.
- Provee una visión de la memoria física, dividida en fragmentos para su mejor administración.

Memory Management Unit

- La MMU cambia el mapeo entre una Dirección Lógica y la Dirección Física.
- Provee una visión de la memoria física, dividida en fragmentos para su mejor administración.
- Permite al Sistema Operativo asignar el mismo espacio de direcciones lógicas para los procesos, separándolos en la memoria física

Memory Management Unit

- La MMU cambia el mapeo entre una Dirección Lógica y la Dirección Física.
- Provee una visión de la memoria física, dividida en fragmentos para su mejor administración.
- Permite al Sistema Operativo asignar el mismo espacio de direcciones lógicas para los procesos, separándolos en la memoria física



Una MMU hace la diferencia

Una MMU hace la diferencia

- La MMU es fundamental para implementar el modelo de proceso en un Sistema Operativo (como es el caso de Linux).

Una MMU hace la diferencia

- La MMU es fundamental para implementar el modelo de proceso en un Sistema Operativo (como es el caso de Linux).
- En este caso cada tarea tiene una o mas áreas de memoria para su código y datos.

Una MMU hace la diferencia

- La MMU es fundamental para implementar el modelo de proceso en un Sistema Operativo (como es el caso de Linux).
- En este caso cada tarea tiene una o mas áreas de memoria para su código y datos.
- Cuando el scheduler retoma la ejecución de una tarea, la MMU mapea su espacio de direccionamiento lógico (que comienza en 0) en un área de memoria física exclusiva y diferente de la de las demás tareas y del propio Sistema Operativo.

Una MMU hace la diferencia

- La MMU es fundamental para implementar el modelo de proceso en un Sistema Operativo (como es el caso de Linux).
- En este caso cada tarea tiene una o mas áreas de memoria para su código y datos.
- Cuando el scheduler retoma la ejecución de una tarea, la MMU mapea su espacio de direccionamiento lógico (que comienza en 0) en un área de memoria física exclusiva y diferente de la de las demás tareas y del propio Sistema Operativo.
- De este modo cada tarea (proceso) gana su espacio de memoria físico exclusivo y queda protegido el del resto de las tareas (procesos).

Una MMU hace la diferencia

- La MMU es fundamental para implementar el modelo de proceso en un Sistema Operativo (como es el caso de Linux).
- En este caso cada tarea tiene una o mas áreas de memoria para su código y datos.
- Cuando el scheduler retoma la ejecución de una tarea, la MMU mapea su espacio de direccionamiento lógico (que comienza en 0) en un área de memoria física exclusiva y diferente de la de las demás tareas y del propio Sistema Operativo.
- De este modo cada tarea (proceso) gana su espacio de memoria físico exclusivo y queda protegido el del resto de las tareas (procesos).
- La desventaja es el overhead que genera remapear la memoria cada vez que se conmuta de un proceso al siguiente.

Una MMU hace la diferencia

- La MMU es fundamental para implementar el modelo de proceso en un Sistema Operativo (como es el caso de Linux).
- En este caso cada tarea tiene una o mas áreas de memoria para su código y datos.
- Cuando el scheduler retoma la ejecución de una tarea, la MMU mapea su espacio de direccionamiento lógico (que comienza en 0) en un área de memoria física exclusiva y diferente de la de las demás tareas y del propio Sistema Operativo.
- De este modo cada tarea (proceso) gana su espacio de memoria físico exclusivo y queda protegido el del resto de las tareas (procesos).
- La desventaja es el overhead que genera remapear la memoria cada vez que se conmuta de un proceso al siguiente.
- La MMU asegura a cada tarea (proceso), la visibilidad de su propia área de memoria, y de las partes relevantes del sistema operativo.

Administración de Memoria con MMU

Administración de Memoria con MMU

- La división de la memoria en bloques, se puede realizar mediante dos criterios diferentes

Administración de Memoria con MMU

- La división de la memoria en bloques, se puede realizar mediante dos criterios diferentes
  - Paginación.

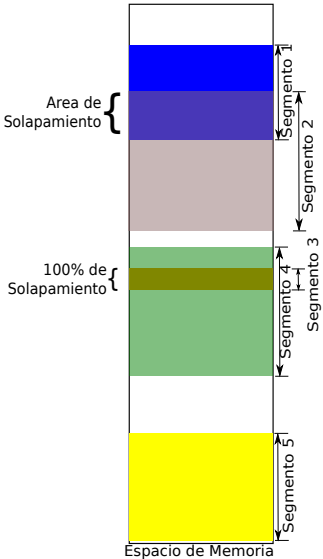
Administración de Memoria con MMU

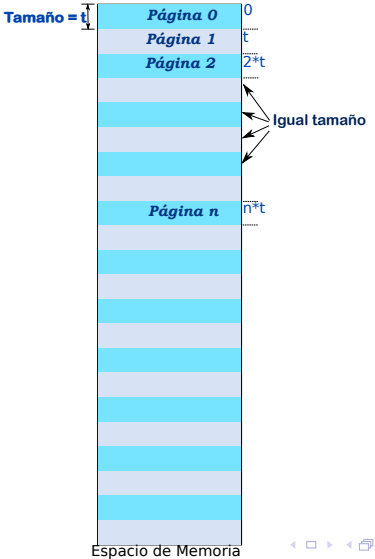
- La división de la memoria en bloques, se puede realizar mediante dos criterios diferentes
  - Paginación.
  - Segmentación.

Administración de Memoria con MMU

- La división de la memoria en bloques, se puede realizar mediante dos criterios diferentes
  - Paginación.
  - Segmentación.
- O en algunos casos como una combinación o superposición de ambos.

Segmentación vs. Paginación





Espacio Físico

- Los procesadores IA-32 organizan la memoria como una secuencia de bytes, direccionables a través de su Bus de Address.
- La memoria conectada a este bus se denomina ***memoria física*** .
- El espacio de direcciones que pueden volcarse sobre este bus se denomina ***direcciones físicas***.

Capacidad de direccionamiento de memoria

Los procesadores IA-32 son la continuidad del 8086. Este procesador fue el primer de 16 bits de ancho de palabra, y por ende todos sus registros internos tienen ese tamaño. Su espacio de Direccionamiento es de 1 Mbyte ∴ Address Bus es de 20 líneas. Este espacio de direccionamiento se administra por segmentación.

Espacio Lógico

Segmentación

Por diversos motivos que en su momento tuvieron sentido, Intel definió organizar el espacio de direccionamiento de la Familia iAPx86 en segmentos. El compromiso de compatibilidad ató a los siguientes procesadores a mantener este esquema

Espacio Lógico

Condiciones iniciales de segmentación

- 4 registros de segmento para almacenar hasta 4 selectores de segmento.

Espacio Lógico

Condiciones iniciales de segmentación

- 4 registros de segmento para almacenar hasta 4 selectores de segmento.
- Registros de 16 bits los segmentos tienen a lo sumo 64K de tamaño

Espacio Lógico

Condiciones iniciales de segmentación

- 4 registros de segmento para almacenar hasta 4 selectores de segmento.
- Registros de 16 bits los segmentos tienen a lo sumo 64K de tamaño
- Expresión de las direcciones en el modelo de programación mediante dos valores:

Espacio Lógico

Condiciones iniciales de segmentación

- 4 registros de segmento para almacenar hasta 4 selectores de segmento.
- Registros de 16 bits los segmentos tienen a lo sumo 64K de tamaño
- Expresión de las direcciones en el modelo de programación mediante dos valores:
  1. Identificador del segmento en el que se encuentra la variable o la instrucción que se desea direccionar,

Espacio Lógico

Condiciones iniciales de segmentación

- 4 registros de segmento para almacenar hasta 4 selectores de segmento.
- Registros de 16 bits los segmentos tienen a lo sumo 64K de tamaño
- Expresión de las direcciones en el modelo de programación mediante dos valores:
  1. Identificador del segmento en el que se encuentra la variable o la instrucción que se desea direccionar,
  2. Desplazamiento, offset, o ***dirección efectiva*** a partir del inicio de ese segmento en donde se encuentra efectivamente

Espacio Lógico

Dirección Lógica IA-32

Dirección Lógica Intel 64

Espacio Lógico

Dirección Lógica IA-32

Dirección Lógica Intel 64

- Estos dos valores por si solos no tienen significado físico

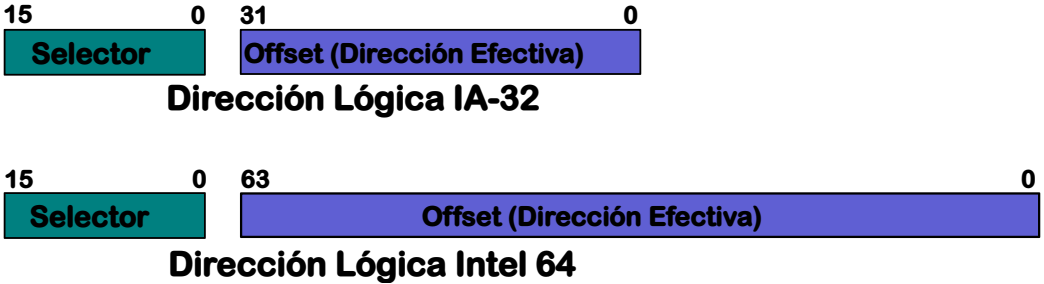
Espacio Lógico

Dirección Lógica IA-32

Dirección Lógica Intel 64

- Estos dos valores por si solos no tienen significado físico
- Para lograr a partir de ellos identificar la ***dirección física*** de la variable o instrucción el procesador debe realizar una serie de operaciones.

Espacio Lógico



- Estos dos valores por si solos no tienen significado físico
- Para lograr a partir de ellos identificar la ***dirección física*** de la variable o instrucción el procesador debe realizar una serie de operaciones.

Dirección Lógica

A una dirección de memoria expresada en términos de los recursos de la arquitectura del procesador las llamaremos ***dirección lógica*** .

Origen y Evolución

Origen y Evolución

- Con el procesador 80286 se incluyó (aún en un procesador de 16 bits) un nuevo modo de trabajo que evolucionaría con la arquitectura IA-32: El Modo Protegido.

Origen y Evolución

- Con el procesador 80286 se incluyó (aún en un procesador de 16 bits) un nuevo modo de trabajo que evolucionaría con la arquitectura IA-32: El Modo Protegido.
- Aún con la aparición de las extensiones de 64 bits para esta familia de procesadores, este modo de trabajo no ha caído en desuso.

Origen y Evolución

- Con el procesador 80286 se incluyó (aún en un procesador de 16 bits) un nuevo modo de trabajo que evolucionaría con la arquitectura IA-32: El Modo Protegido.
- Aún con la aparición de las extensiones de 64 bits para esta familia de procesadores, este modo de trabajo no ha caído en desuso.
- Sin embargo es de esperar que en algún momento todo pase a funcionar en 64 bits. Nos referiremos a éste modo como el Modo Legacy.

Origen y Evolución

- Con el procesador 80286 se incluyó (aún en un procesador de 16 bits) un nuevo modo de trabajo que evolucionaría con la arquitectura IA-32: El Modo Protegido.
- Aún con la aparición de las extensiones de 64 bits para esta familia de procesadores, este modo de trabajo no ha caído en desuso.
- Sin embargo es de esperar que en algún momento todo pase a funcionar en 64 bits. Nos referiremos a éste modo como el Modo Legacy.
- Desde el 80386 en adelante los procesadores de Intel vienen provistos de 32 líneas de datos y 32 líneas independientes de address. Esto hace que cualquier procesador IA-32 direcciona $2^{32} - 1$ bytes.

Origen y Evolución

- Con el procesador 80286 se incluyó (aún en un procesador de 16 bits) un nuevo modo de trabajo que evolucionaría con la arquitectura IA-32: El Modo Protegido.
- Aún con la aparición de las extensiones de 64 bits para esta familia de procesadores, este modo de trabajo no ha caído en desuso.
- Sin embargo es de esperar que en algún momento todo pase a funcionar en 64 bits. Nos referiremos a éste modo como el Modo Legacy.
- Desde el 80386 en adelante los procesadores de Intel vienen provistos de 32 líneas de datos y 32 líneas independientes de address. Esto hace que cualquier procesador IA-32 direcciona $2^{32} - 1$ bytes.
- A partir de la microarquitectura P6 se incluyeron cuatro líneas adicionales en el bus de address las cuales se habilitan desde modo protegido siempre que se haya activado la paginación. De este modo se llega a $2^{36} - 1$ bytes (64 Gbytes de ***memoria física***)

Particularidades

Particularidades

- Como hemos visto, se puede ingresar a un modo denominado IA-32e, en el que se trabaja con una arquitectura de 64 bits

Particularidades

- Como hemos visto, se puede ingresar a un modo denominado IA-32e, en el que se trabaja con una arquitectura de 64 bits
- En este modo se generan direcciones lineales de 64 bits, de los que por el momento se utilizan los 48 menos significativos.

Particularidades

- Como hemos visto, se puede ingresar a un modo denominado IA-32e, en el que se trabaja con una arquitectura de 64 bits
- En este modo se generan direcciones lineales de 64 bits, de los que por el momento se utilizan los 48 menos significativos.
- En realidad la cantidad de bits significativos de la dirección lineal es procesador dependiente y se puede determinar mediante la función 0x80000008 de la instrucción CPUID (vuelve en AH).

Particularidades

- Como hemos visto, se puede ingresar a un modo denominado IA-32e, en el que se trabaja con una arquitectura de 64 bits
- En este modo se generan direcciones lineales de 64 bits, de los que por el momento se utilizan los 48 menos significativos.
- En realidad la cantidad de bits significativos de la dirección lineal es procesador dependiente y se puede determinar mediante la función 0x80000008 de la instrucción CPUID (vuelve en AH).
- Una ***dirección lineal*** tiene formato canónico, cuando sus bits desde el 63 al mas significativo de los válidos devueltos por la instrucción CPUID.80000008H están todos en el mismo estado que el bit mas significativo. Al momento esta función retorna 48 para todos los procesadores, de modo que una ***dirección lineal*** tiene formato canónico si los bits 48 a 63 están el mismo estado lógico que el bit 47.

Particularidades

- Como hemos visto, se puede ingresar a un modo denominado IA-32e, en el que se trabaja con una arquitectura de 64 bits
- En este modo se generan direcciones lineales de 64 bits, de los que por el momento se utilizan los 48 menos significativos.
- En realidad la cantidad de bits significativos de la dirección lineal es procesador dependiente y se puede determinar mediante la función 0x80000008 de la instrucción CPUID (vuelve en AH).
- Una ***dirección lineal*** tiene formato canónico, cuando sus bits desde el 63 al mas significativo de los válidos devueltos por la instrucción CPUID.80000008H están todos en el mismo estado que el bit mas significativo. Al momento esta función retorna 48 para todos los procesadores, de modo que una ***dirección lineal*** tiene formato canónico si los bits 48 a 63 están el mismo estado lógico que el bit 47.
- Por cada acceso a memoria en el modo IA-32e submodo 64 bits, la unidad de protección del procesador chequea el formato canónico de la ***dirección lineal*** .

Formato Canónico

Si no está en formato canónico...excepción!

Generalmente la excepción es de protección general #GP, excepto en el caso que la dirección se genere a partir de una dirección que haga referencia a la pila, y que por lo tanto se calcula a partir del stack segment, en cuyo caso se genera una excepción de pila #SF.

Por lo general las instrucciones que la generan son PUSH, POP, y referencias a memoria que utilicen RSP y RBP, excepto si estas instrucciones utilizan un prefijo de segmento FS o GS que pise el valor default, en cuyo caso generarían una excepción #GP (Notar que los prefijos de segmento DS ES y CS se ignoran en modo 64 bits).

Traducción de direcciones

Estos procesadores en Modo Protegido generan una ***dirección física*** en el bus de address mediante un proceso de traducción en dos niveles:

1. traslación de una dirección lógica
2. paginación del espacio lineal

La MMU

El procesador posee una MMU (Memory Management Unit) compuesta de dos subunidades conectadas en cascada: La Unidad de Segmentación que se encarga de trasladar la ***dirección lógica*** en una ***dirección lineal*** , y la Unidad de Paginación que traduce la ***dirección lineal*** en una ***dirección física*** que enviará por el bus de address hacia la memoria externa.

El espacio Lineal

Definición

El espacio lineal de direcciones es al igual que el espacio físico un rango de direcciones contiguas plano, que inicia en la dirección 0, y llega al máximo valor que puede ocupar un segmento de acuerdo al modo de trabajo. Por ejemplo en modo protegido de 32 bits el rango de direcciones lineales arranca en 0 y puede llegar hasta $2^{32} - 1$, es decir 0xFFFFFFFF. En el modo 64 bits la dirección lógica resultante es de 64 bits y debe estar representada en formato canónico.

Traslación de la dirección lógica

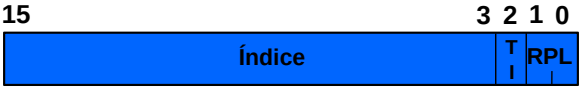
- El procesador debe tener la capacidad de especificar a cada proceso o tarea, donde comienza cada uno de sus segmentos y cual es su tamaño.
- La dirección de comienzo de un segmento es un valor de 32 bits en modo protegido.
- El tamaño máximo que puede tener un segmento es el de su máximo valor de desplazamiento, este valor también es de 32 bits en modo protegido y hasta el momento 48 bits en modo IA-32e.
- Además es lógico pensar que se necesiten algunos bits adicionales de control para administración de acceso a los diferentes segmentos.

Traslación de la dirección lógica

Conclusión

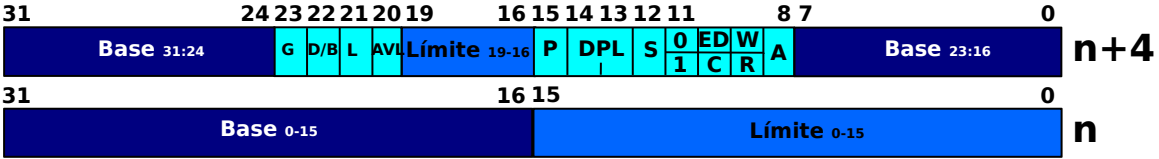
Los modestos 16 bits de un registro de segmento resultan absolutamente insuficientes para almacenar toda esta información. todo lo que puede contener un registro de segmento es un valor, que como ya se ha dicho, se denomina selector de segmento, y que no es otra cosa que una referencia a una estructura de datos mas grande que contiene, la ***Dirección Base*** , el ***Límite*** , y los ***Atributos*** del segmento seleccionado. Como veremos esta estructura se denomina ***Descriptor de Segmento*** , reside en la memoria RAM del sistema, y para mejor organización se los agrupa en tablas, de modo de facilitar su ubicación al procesador mediante un mecanismo predeterminado.

Formato



- ***Index***. Se utiliza como índice en la tabla de descriptores. Un valor de ***Index*** = ***n***, corresponde al n-ésimo elemento de la Tabla. Al tener 13 bits indica que cada tabla puede alojar 2^{13} (8192) descriptores.
- ***TI***. Table Indicator. Selecciona en que tabla de descriptores debe buscarse el segmento seleccionado: *GDT*por Global Descriptor Table (si ***TI*** = 0), o *LDT*por Local Descriptor Table (Si ***TI*** = 1).
- ***RPL*** Requested Priviledge Level. En el Capítulo Protección lo abordaremos en detalle. Por ahora solo interesa saber que este campo es el nivel de privilegio que declara tener el dueño del segmento, o sea el grado de autorización que se tiene para cada acceso: 00 es el mayor privilegio, y 11 el menor.

Formato



- ***Dirección Base*** . Es la dirección a partir de la cual se despliega en forma continua el segmento.
- ***Límite*** . El Límite de un segmento especifica el máximo offset que puede tener un byte direccionable dentro del segmento. Suele confundirse este concepto con el tamaño del segmento. En realidad el ***Límite*** es el tamaño del segmento menos 1, ya que el offset del primer byte del segmento es 0.

Atributos

Atributos

- ***G***. **G**ranularity. Establece la unidad de medida del campo ***Límite*** . Si ***G*** = 0, el máximo offset de un byte es igual a ***Límite*** . Si ***G*** = 1, el máximo offset es igual a ***Límite*** * 0x1000 +0xFFF.

Atributos

- ***G***. **G**ranularity. Establece la unidad de medida del campo ***Límite*** . Si ***G*** = 0, el máximo offset de un byte es igual a ***Límite*** . Si ***G*** = 1, el máximo offset es igual a ***Límite*** * 0x1000 +0xFFF.
- ***D/B***. **D**efault / **B**ig. Configura el tamaño de los segmentos. Si es 0, (Default) el segmento es de 16 bits. Si es 1, (Big) es un segmento de 32 bits. Para segmentos de código, ***D/B*** = 0 implica que el tamaño de datos es de 16 bits u 8 bits, y el de direcciones de 16 bits. Si en cambio ***D/B*** = 1, el tamaño de un offset es 32 bits y el de los operandos es de 32 bits u 8 bits. En ambos casos mediante los prefijos de instrucción 66h y 67h respectivamente podemos alterar los defaults. En el caso de un segmento de datos utilizado como pila, si ***D/B*** = 0 las operaciones de la pila son de 16 bits, aunque el operando de la instrucción sea de 8 bits. Si ***D/B*** = 1, son de 32 bits independientemente del tamaño del operando. El valor tope del segmento será también consecuencia del valor de este bit.

Atributos

Atributos

- ***L***. El procesador solo mira este bit en el Modo IA-32e. Si en un segmento de código este bit es ’1’, indica que el segmento contiene código nativo de 64 bits, caso contrario, ejecutará en Modo Compatibilidad. En modo IA-32e, si ***L*** es ’1’, entonces ***D/B*** debe estar en ’0’. Si el procesador no está en modo IA-32e, o si está en este modo pero el segmento no es de código, el bit ***L*** debe estar siempre en ’0’.

Atributos

- ***L***. El procesador solo mira este bit en el Modo IA-32e. Si en un segmento de código este bit es '1', indica que el segmento contiene código nativo de 64 bits, caso contrario, ejecutará en Modo Compatibilidad. En modo IA-32e, si ***L*** es '1', entonces ***D/B*** debe estar en '0'. Si el procesador no está en modo IA-32e, o si está en este modo pero el segmento no es de código, el bit ***L*** debe estar siempre en '0'.
- ***AVL***. **AV**ai**L**able. Este bit no es usado por el procesador para ningún propósito específico. Queda para que el programador de sistemas le asigne el uso que considere mas apropiado.

Atributos

- ***L***. El procesador solo mira este bit en el Modo IA-32e. Si en un segmento de código este bit es '1', indica que el segmento contiene código nativo de 64 bits, caso contrario, ejecutará en Modo Compatibilidad. En modo IA-32e, si ***L*** es '1', entonces ***D/B*** debe estar en '0'. Si el procesador no está en modo IA-32e, o si está en este modo pero el segmento no es de código, el bit ***L*** debe estar siempre en '0'.
- ***AVL***. **AV**ai**L**able. Este bit no es usado por el procesador para ningún propósito específico. Queda para que el programador de sistemas le asigne el uso que considere mas apropiado.
- ***P***. **P**resent. Cuando es '1' el segmento correspondiente está presente en la memoria RAM. Si es '0', el segmento está en la memoria virtual (disco). Un acceso a un segmento cuyo bit ***P*** está en '0', genera una excepción #NP (Segmento No Presente). Esto permite al kernel solucionar el problema, efectuando el "swap" entre el disco a memoria para ponerlo accesiible en RAM.

Atributos

Atributos

- ***A***. **A**ccedido. Se setea cada vez que se accede una dirección en el segmento. Permite al Sistema Operativo contabilizar los accesos para elaborar estadísticas de uso que permitan identificar cual es el segmento a ser desalojado llegado el momento.

Atributos

- ***A***. **A**ccedido. Se setea cada vez que se accede una dirección en el segmento. Permite al Sistema Operativo contabilizar los accesos para elaborar estadísticas de uso que permitan identificar cual es el segmento a ser desalojado llegado el momento.
- ***DPL***. **D**escriptor **P**riviledge **L**evel. Nivel de privilegio que debe tener el segmento que contiene el código que pretende acceder a éste segmento.

Atributos

- ***A***. **A**ccedido. Se setea cada vez que se accede una dirección en el segmento. Permite al Sistema Operativo contabilizar los accesos para elaborar estadísticas de uso que permitan identificar cual es el segmento a ser desalojado llegado el momento.
- ***DPL***. **D**escriptor **P**riviledge **L**evel. Nivel de privilegio que debe tener el segmento que contiene el código que pretende acceder a éste segmento.
- ***S***. **S**ystem. Este bit, **activo bajo** permite administrar en las tablas de descriptores, dos clases bien determinadas de segmentos:

Atributos

- ***A***. **A**ccedido. Se setea cada vez que se accede una dirección en el segmento. Permite al Sistema Operativo contabilizar los accesos para elaborar estadísticas de uso que permitan identificar cual es el segmento a ser desalojado llegado el momento.
- ***DPL***. **D**escriptor **P**riviledge **L**evel. Nivel de privilegio que debe tener el segmento que contiene el código que pretende acceder a éste segmento.
- ***S***. **S**ystem. Este bit, **activo bajo** permite administrar en las tablas de descriptores, dos clases bien determinadas de segmentos:
  1. Segmentos de Código o Datos

Atributos

- ***A***. **A**ccedido. Se setea cada vez que se accede una dirección en el segmento. Permite al Sistema Operativo contabilizar los accesos para elaborar estadísticas de uso que permitan identificar cual es el segmento a ser desalojado llegado el momento.
- ***DPL***. **D**escriptor **P**riviledge **L**evel. Nivel de privilegio que debe tener el segmento que contiene el código que pretende acceder a éste segmento.
- ***S***. **S**ystem. Este bit, **activo bajo** permite administrar en las tablas de descriptores, dos clases bien determinadas de segmentos:
  1. Segmentos de Código o Datos
  2. Segmentos de Sistema. Tienen diferentes formatos y en general no se refieren a zonas de memoria (salvo TSS). En general se refieren a mecanismos de uso de recursos del procesador por parte del kernel (por ello reciben el nombre de descriptores de Sistema, ya que **son para uso exclusivo del Sistema Operativo**)

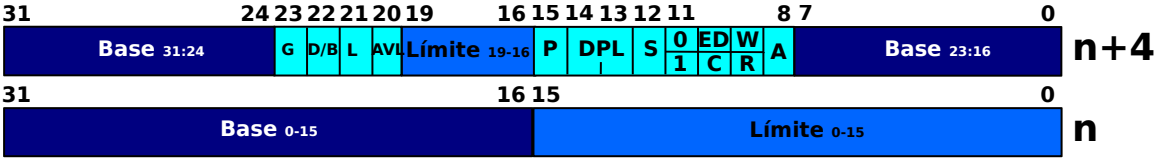
Atributos

- ***A***. **A**ccedido. Se setea cada vez que se accede una dirección en el segmento. Permite al Sistema Operativo contabilizar los accesos para elaborar estadísticas de uso que permitan identificar cual es el segmento a ser desalojado llegado el momento.
- ***DPL***. **D**escriptor **P**riviledge **L**evel. Nivel de privilegio que debe tener el segmento que contiene el código que pretende acceder a éste segmento.
- ***S***. **S**ystem. Este bit, **activo bajo** permite administrar en las tablas de descriptores, dos clases bien determinadas de segmentos:
  1. Segmentos de Código o Datos
  2. Segmentos de Sistema. Tienen diferentes formatos y en general no se refieren a zonas de memoria (salvo TSS). En general se refieren a mecanismos de uso de recursos del procesador por parte del kernel (por ello reciben el nombre de descriptores de Sistema, ya que **son para uso exclusivo del Sistema Operativo**)
- ***Tipo***. Este campo de 4 bits es fuertemente dependiente del tipo (de allí el nombre, según se trate de un segmento de Código, de Datos, o de Sistema). Su detalle, se establece a continuación:

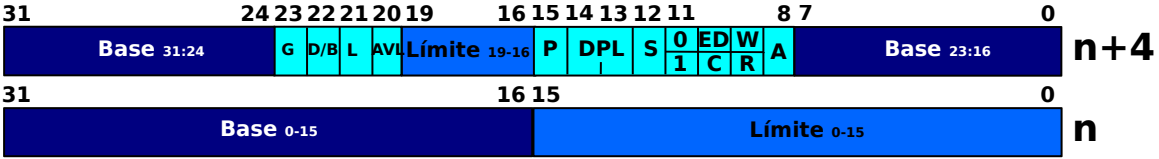
Tipos de Descriptores de Sistema (S = 0)

11	10	9	8	Modo 32 bits	Modo IA-32e
0	0	0	0	Reservado	8 bytes superiores en un segmento de 16 bytes
0	0	0	1	TSS de 16 bits disponible	Reservado
0	0	1	0	LDT	LDT
0	0	1	1	TSS de 16 bits Busy	Reservado
0	1	0	0	Call Gate de 16 bits	Reservado
0	1	0	1	Task Gate	Reservado
0	1	1	0	Interrupt Gate de 16 bits	Reservado
0	1	1	1	Trap Gate de 16 bits	Reservado
1	0	0	0	Reservado	Reservado
1	0	0	1	TSS de 32 bits disponible	TSS de 64 bits disponible
1	0	1	0	Reservado	Reservado
1	0	1	1	TSS de 32 bits Busy	TSS de 64 bits Busy
1	1	0	0	Call Gate de 32 bits	Call Gate de 64 bits
1	1	0	1	Reservado	Reservado
1	1	1	0	Interrupt Gate de 32 bits	Interrupt Gate de 64 bits
1	1	1	1	Trap Gate de 32 bits	Trap Gate de 64 bits

Tipos de Descriptores de Código y Datos (S = 1)

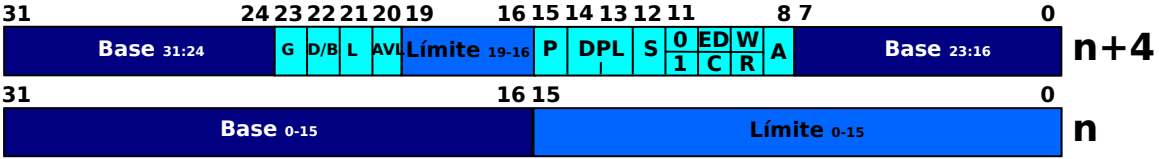


Tipos de Descriptores de Código y Datos (S = 1)



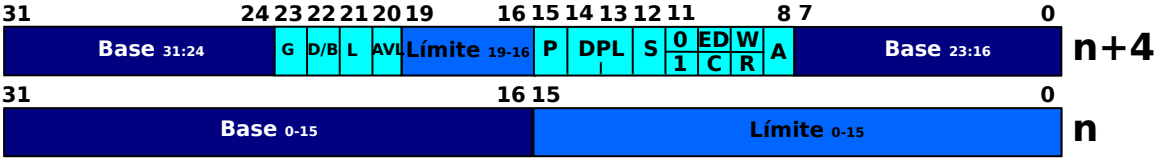
- En este caso el bit 11 define si el segmento correspondiente a este descriptor es de código o de datos según valga ’1’, o ’0’ respectivamente. En cada caso los dos bits subsiguientes tienen un significado diferente.

Tipos de Descriptores de Código y Datos (S = 1)



- En este caso el bit 11 define si el segmento correspondiente a este descriptor es de código o de datos según valga ’1’, o ’0’ respectivamente. En cada caso los dos bits subsiguientes tienen un significado diferente.
- Si el bit 11 es ’1’ el segmento es de código

Tipos de Descriptores de Código y Datos (S = 1)



- En este caso el bit 11 define si el segmento correspondiente a este descriptor es de código o de datos según valga ’1’, o ’0’ respectivamente. En cada caso los dos bits subsiguientes tienen un significado diferente.
- Si el bit 11 es ’1’ el segmento es de código
- Si el bit 11 es ’0’ el segmento es de datos

Atributos de Descriptores de Código(S = 1, Bit_{11} = 1)

Atributos de Descriptores de Código(S = 1, Bit_{11} = 1)

- ***C***. **C**onforming. Significa ajustable. Estos segmentos de código “ajustan” su nivel de privilegio al del código que los ha invocado. Permiten que un segmento de código pueda ser invocado desde otro segmento de código menos privilegiado mediante por ejemplo una instrucción CALL a una subrutina residente en este segmento. Sin embargo el código privilegiado ajustará su nivel de privilegio al del segmento de código invocante.

Atributos de Descriptores de Código(S = 1, Bit_{11} = 1)

- ***C***. **C**onforming. Significa ajustable. Estos segmentos de código “ajustan” su nivel de privilegio al del código que los ha invocado. Permiten que un segmento de código pueda ser invocado desde otro segmento de código menos privilegiado mediante por ejemplo una instrucción CALL a una subrutina residente en este segmento. Sin embargo el código privilegiado ajustará su nivel de privilegio al del segmento de código invocante.
- ***R***. **R**eadable. Este bit habilita cuando es ’1’ la lectura de direcciones de memoria residente en el segmento. En general se usa cuando se tienen constantes en el segmento que necesitan ser accedidas para su lectura. Si el segmento solo tiene código, puede ponerse R = 0 en el descriptor para prevenir que se pueda leer cualquier ítem de este segmento utilizando el prefijo CS en la instrucción para modificar el comportamiento del procesador en la asignación por default del registro de segmento en el modo de direccionamiento empleado.

Atributos de Descriptores de Datos(S = 1, $Bit_{11} = 0$)

Atributos de Descriptores de Datos(S = 1, Bit_{11} = 0)

- ***ED***. **E**xpand **D**own. Cuando el segmento de datos va a ser utilizado como Pila, puede optarse por tratarlo como un segmento común de datos, o definirlo como Expand Down, poniendo de manifiesto que es una pila, y su puntero de direcciones decrece hacia las direcciones de memoria numéricamente menores a medida que se expande el segmento (de allí el término Expand Down). En este caso, el concepto de límite efectivo se interpreta de modo diferente: Es el último valor de offset a partir de la dirección base que no puede ser accedido. Para estos segmentos el rango de offsets válidos es el siguiente:
  - (límite efectivo +1) hasta 0FFFFh, si el bit ***D/B*** = 0.
  - (límite efectivo +1) hasta 0FFFFFFFFh, si el bit ***D/B*** = 1.

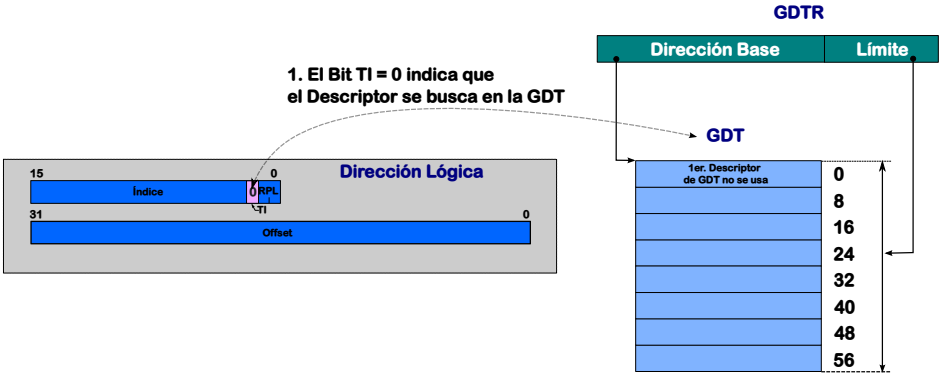
  Se ampliará al tocar el tema protección.

Atributos de Descriptores de Datos(S = 1, Bit_{11} = 0)

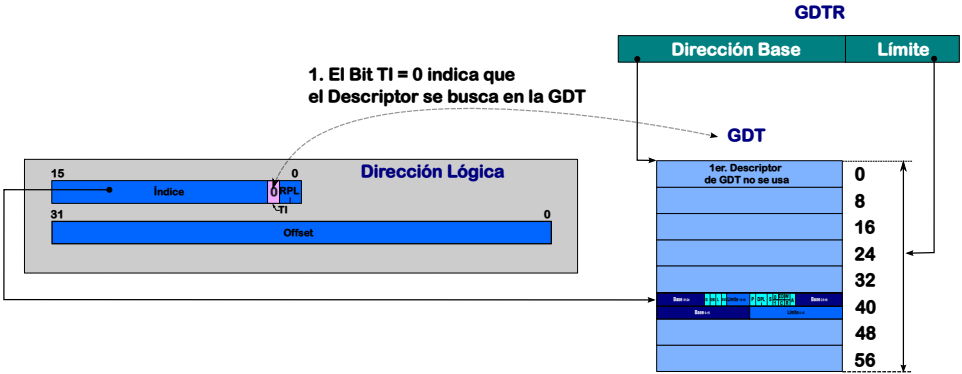
- ***ED***. **E**xpand **D**own. Cuando el segmento de datos va a ser utilizado como Pila, puede optarse por tratarlo como un segmento común de datos, o definirlo como Expand Down, poniendo de manifiesto que es una pila, y su puntero de direcciones decrece hacia las direcciones de memoria numéricamente menores a medida que se expande el segmento (de allí el término Expand Down). En este caso, el concepto de límite efectivo se interpreta de modo diferente: Es el último valor de offset a partir de la dirección base que no puede ser accedido. Para estos segmentos el rango de offsets válidos es el siguiente:
  - (límite efectivo +1) hasta 0FFFFh, si el bit ***D/B*** = 0.
  - (límite efectivo +1) hasta 0FFFFFFFFh, si el bit ***D/B*** = 1.

  Se ampliará al tocar el tema protección.
- ***W***. **W**ritable. Este bit, indica si el segmento de datos puede escribirse. Si este bit está en '0', el segmento contiene datos pero es Read Only.

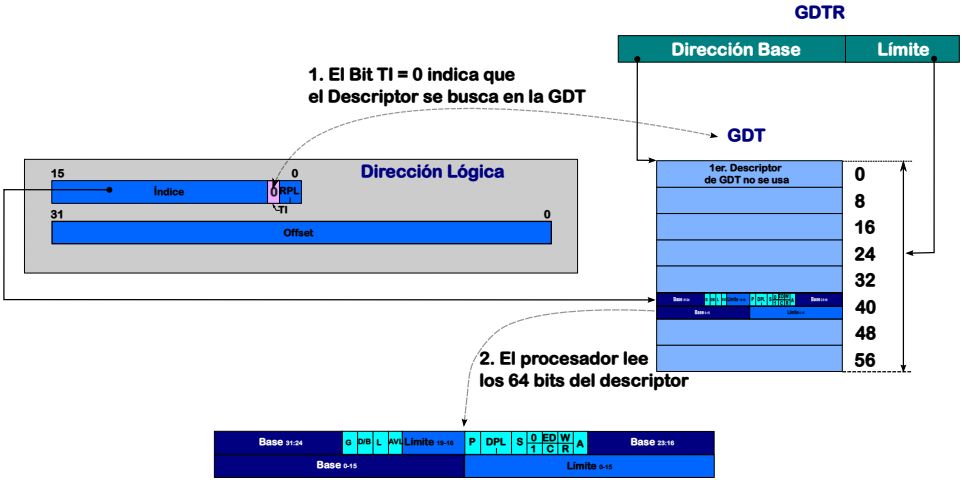
Descriptores de Segmento en la GDT (*TI* = 0)



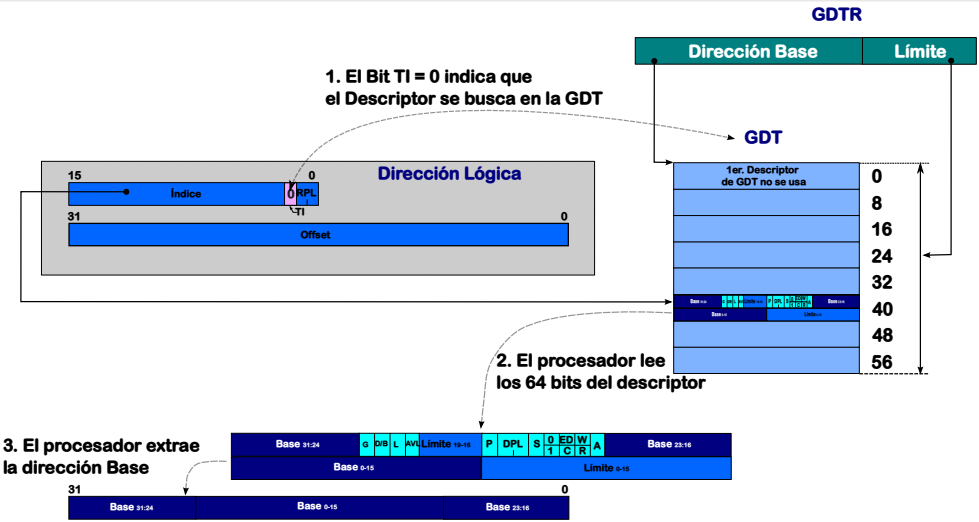
Descriptores de Segmento en la GDT (*TI* = 0)



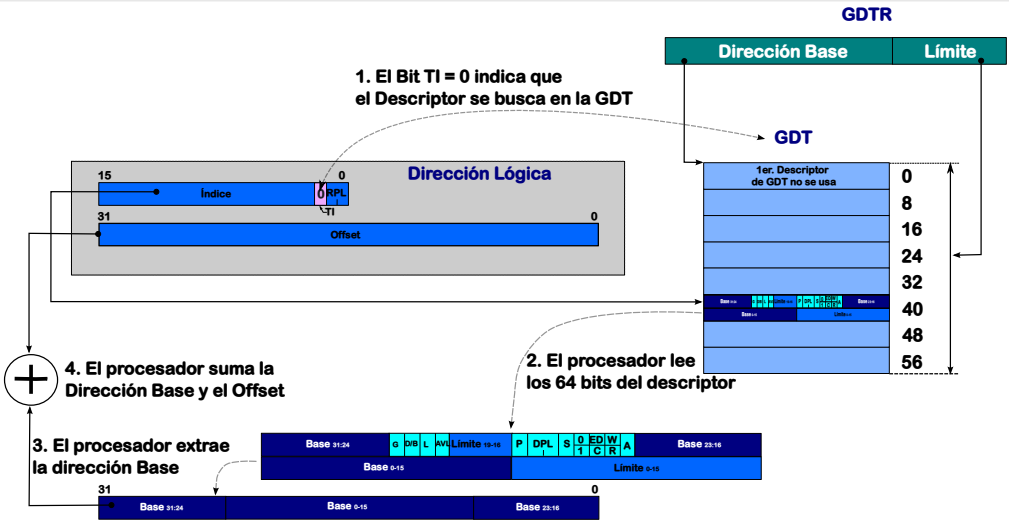
Descriptores de Segmento en la GDT (*TI* = 0)



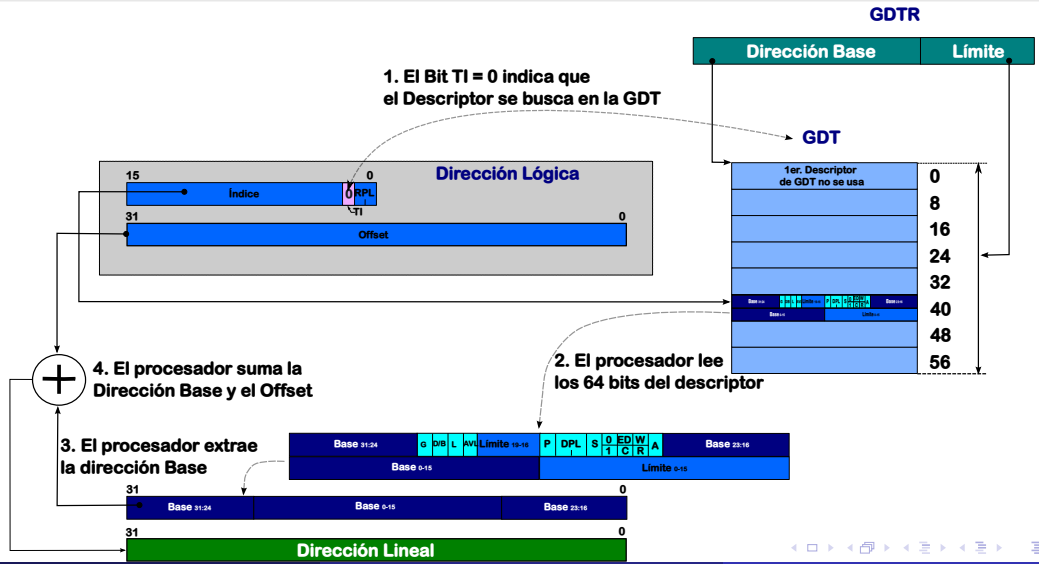
Descriptores de Segmento en la GDT (*TI* = 0)



Descriptores de Segmento en la GDT (*TI* = 0)



Descriptores de Segmento en la GDT (*TI* = 0)

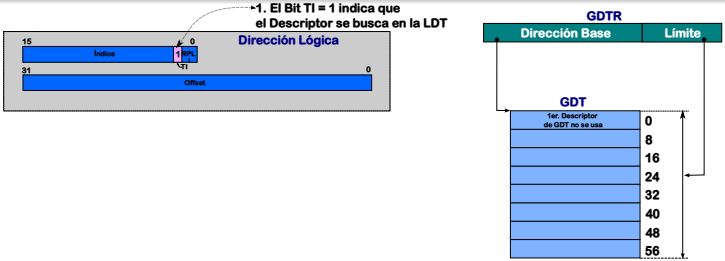


Descriptores de Segmento en la GDT (*TI* = 0)

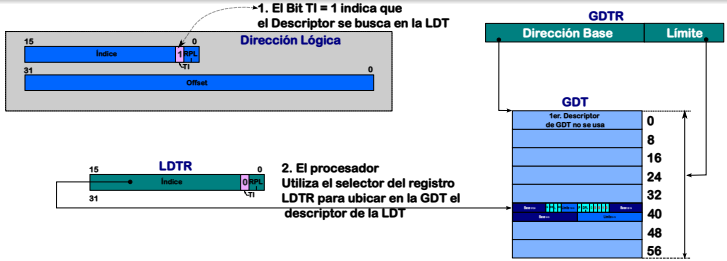
Lo que podemos observar es que partiendo de la ***dirección lógica*** , lo primero que trabaja el procesador es el selector y el offset recién se utiliza al final del cálculo de la ***dirección lineal*** .
La operatoria que realiza el procesador es la siguiente:

1. El procesador evalúa el estado del bit 2 del selector, es decir **TI**. En este caso TI = 0, de modo que el procesador asume que buscará el descriptor en la tabla *GDT*.
2. El registro GDTR del procesador contiene en su campo ***Dirección Base*** la ***dirección física*** en donde comienza la *GDT*.
3. El valor ***n*** contenido por los 13 bits del campo Index del selector, referencia al n-ésimo elemento de la tabla *GDT*.
4. El procesador accede a la dirección de ***memoria física*** dada por: $GDT.Base + 8 * Index$ y lee 8 bytes a partir de ella.
5. Una vez leído el descriptor, internamente reordena la ***Dirección Base*** y el ***Límite*** y agrupa los ***Atributos*** .
6. La Unidad de protección verifica que el offset contenido en el registro correspondiente de la ***dirección lógica*** corresponda al rango de offsets válidos del segmento de acuerdo al valor del campo ***Límite*** , y de los bits de ***Atributos G***, ***D/B***, y ***ED***.
7. La Unidad de protección chequea que la operación a realizarse en el segmento se corresponda con los bits de ***Atributos R*** y ***C***, si es de código, ***W*** si es de datos, que el código de acceso tenga los privilegios necesarios de acuerdo a los bits ***DPL*** del descriptor, que P = 1, entre los mas comunes.
8. Si todo está de manera correcta, el procesador suma el valor de offset contenido en la ***dirección lógica*** , con la ***Dirección Base*** del segmento y conforma la ***dirección lineal***

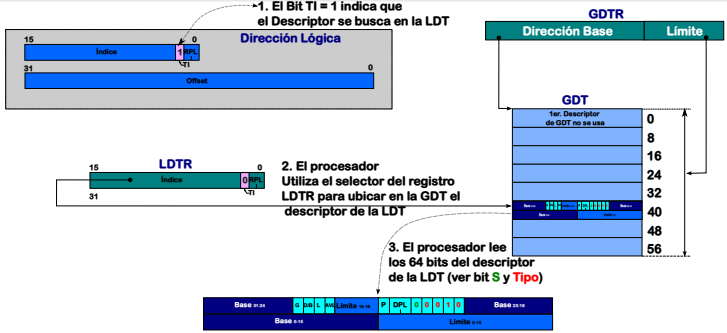
Descriptores de Segmento en la LDT (*TI* = 1)



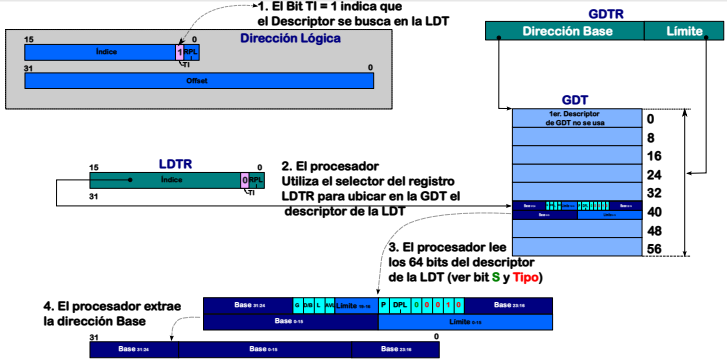
Descriptores de Segmento en la LDT (*TI* = 1)



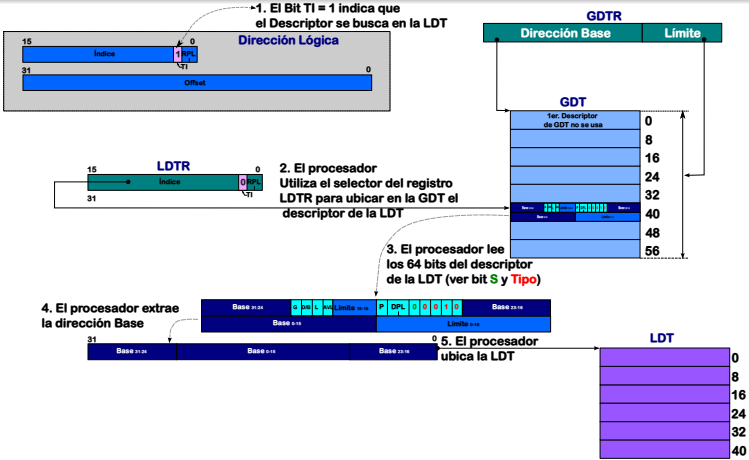
Descriptores de Segmento en la LDT (*TI* = 1)



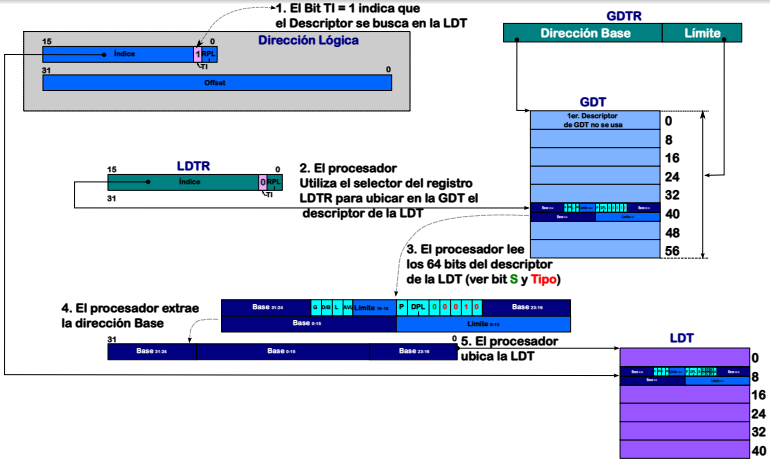
Descriptores de Segmento en la LDT (*TI* = 1)



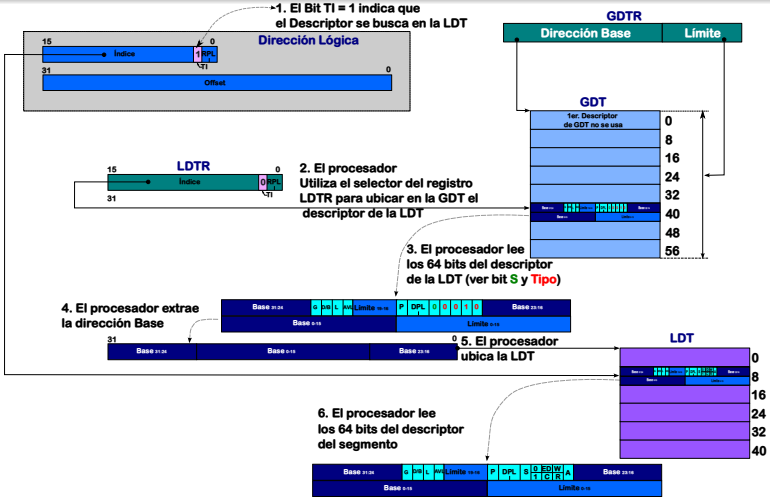
Descriptores de Segmento en la LDT (*TI* = 1)



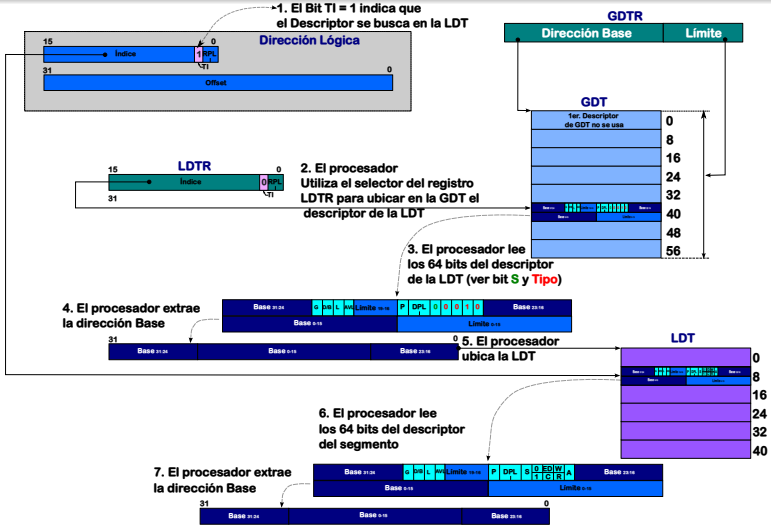
Descriptores de Segmento en la LDT (*TI* = 1)



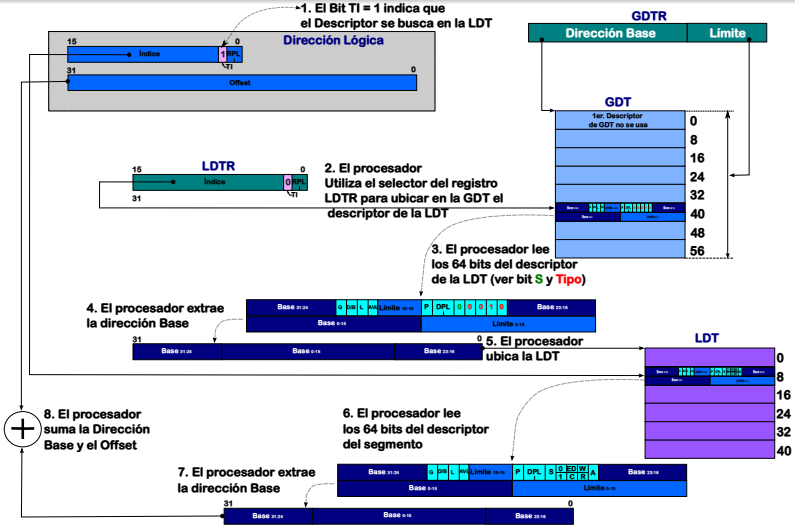
Descriptores de Segmento en la LDT (*TI* = 1)



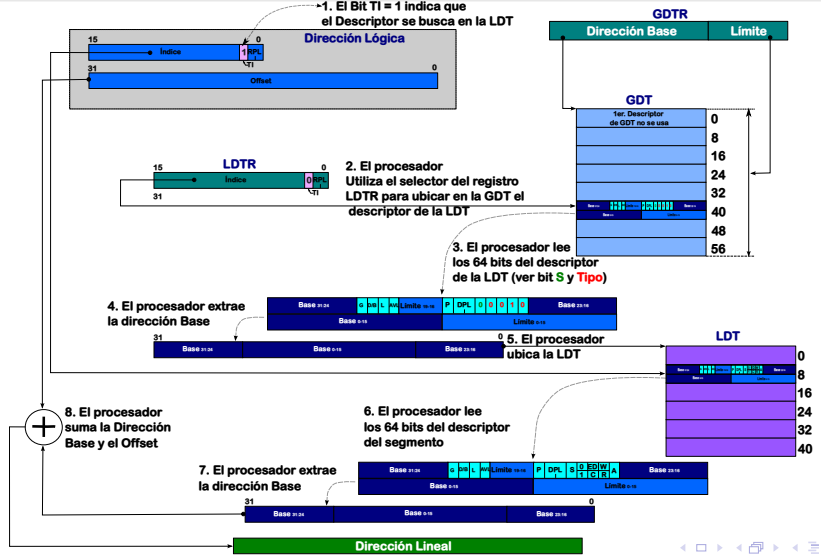
Descriptores de Segmento en la LDT (*TI* = 1)



Descriptores de Segmento en la LDT (*TI* = 1)



Descriptores de Segmento en la LDT (*TI* = 1)



Descriptores de Segmento en la LDT (*TI* = 1)

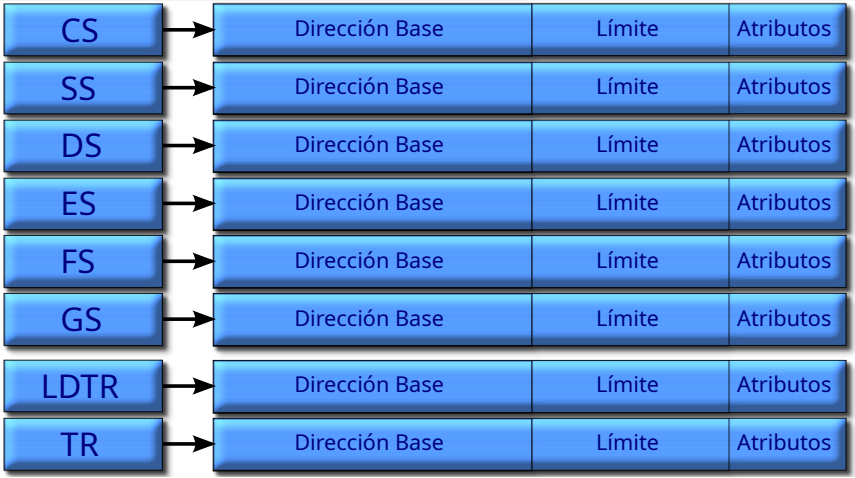
La operatoria en este caso es la siguiente:

1. El procesador evalúa el estado del bit 2 del selector, es decir **TI**. En este caso TI = 1, de modo que el procesador asume que buscará el descriptor en la tabla *LDT*.
2. El registro LDTR del procesador contiene el selector de segmento que permitirá ubicar en la *GDT*el descriptor de sistema del segmento que contiene la *LDT*. Por lo tanto antes de trabajar con la ***dirección lógica*** , el procesador necesita obtener la *LDT*.
3. El valor ***n*** contenido por los 13 bits del campo Index del selector presente en el registro LDTR, referencia al n-ésimo elemento de la tabla *GDT*. En ese sitio de la GDT debe haber necesariamente un descriptor de segmento de sistema, cuyo valor en el campo Tipo sea 0010, es decir el código binario correspondiente a un descriptor de *LDT*. De otro modo el procesador generará una excepción.
4. El procesador accede a la dirección de ***memoria física*** dada por: $GDT.Base + 8 * Index$ y lee 8 bytes a partir de ella.
5. Una vez leído el descriptor, internamente reordena la ***Dirección Base*** , comprueba los ***Atributos*** , en especial que el valor del bit S sea 0 y que el descriptor corresponda a un Descriptor de *LDT*(es decir Tipo = 0010).

Descriptores de Segmento en la LDT (***TI*** = 1)

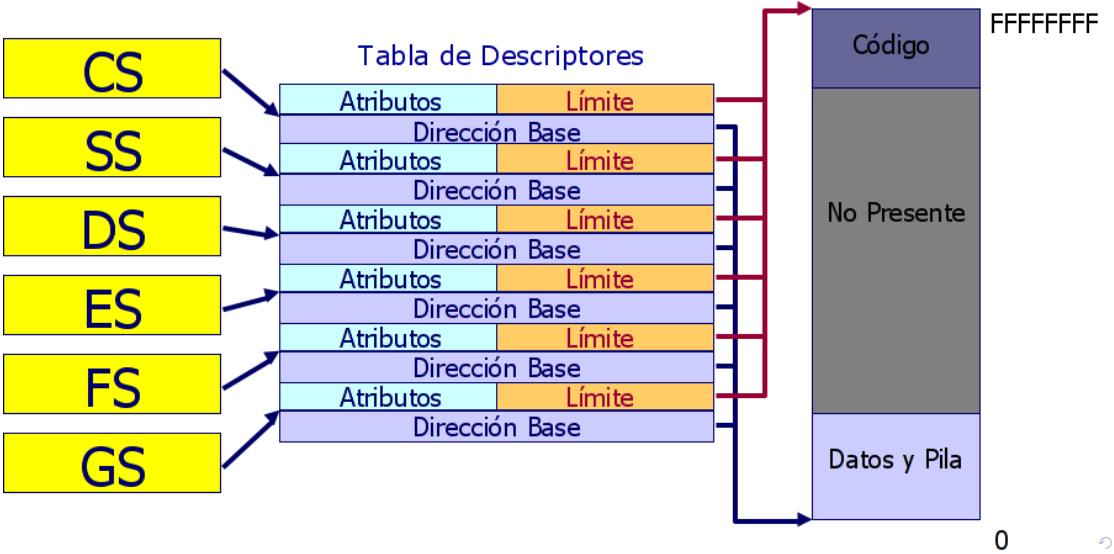
6. Una vez leído y validado el descriptor de segmento de la *LDT*, el procesador puede leer desde la *LDT*el descriptor del segmento de la ***dirección lógica*** . Para ello utiliza, ahora si, el valor ***n*** contenido por los 13 bits del campo Index del selector de segmento que compone dicha ***dirección lógica*** , que referenciará al n-ésimo elemento de la tabla *LDT*.
7. El procesador accede a la dirección de ***memoria física*** dada por: $LDT.Base + 8 * Index$ y lee 8 bytes a partir de ella.
8. Una vez leído el descriptor del segmento, la Unidad de protección verifica que el offset contenido en el registro correspondiente de la ***dirección lógica*** corresponda al rango de offsets válidos del segmento de acuerdo al valor del campo ***Límite*** , y de los bits de ***Atributos G***, ***D/B***, y ***ED***.
9. La Unidad de protección chequea que la operación a realizarse en el segmento se corresponda con los bits de ***Atributos R*** y ***C***, si es de código, ***W*** si es de datos, que el código de acceso tenga los privilegios necesarios de acuerdo a los bits ***DPL*** del descriptor, que P = 1, entre los mas comunes.
10. Si todo está de manera correcta, el procesador suma el valor de offset contenido en la ***dirección lógica*** , con la ***Dirección Base*** del segmento y conforma la ***dirección lineal***

Registros cache ocultos (hidden)

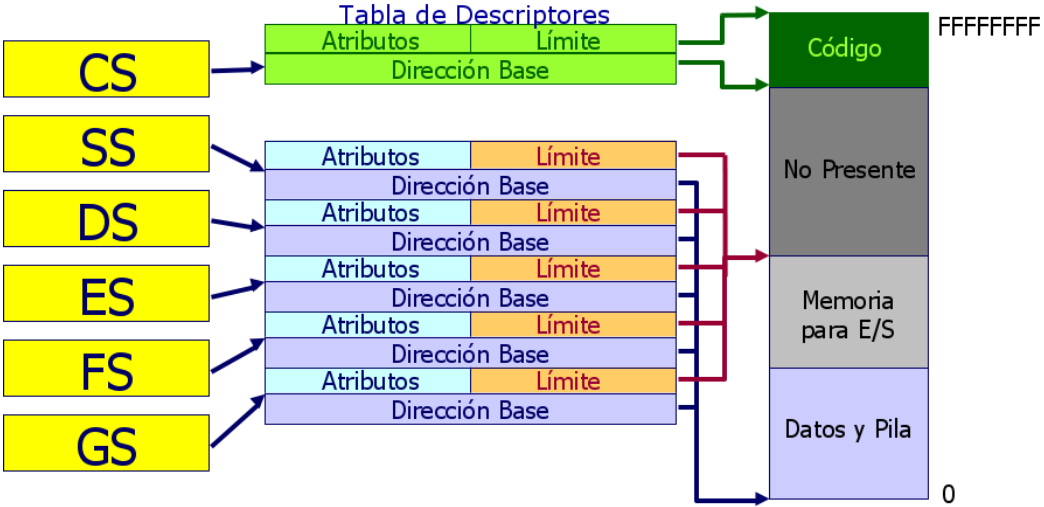


Cuando el procesador trabaja en modo 64 bits los tamaños de los campos del descriptor se ajustan para contener los valores correspondientes a los selectores de 64 bits.

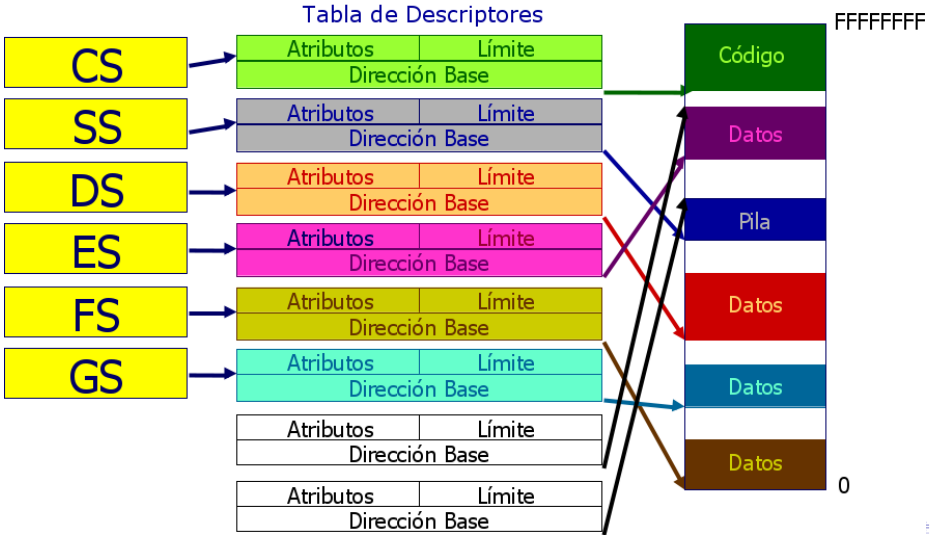
Modelo Flat


CS
SS
DS
ES
FS
GS
Tabla de Descriptores
Atributos
Límite
Dirección Base
Atributos
Límite
Dirección Base
Atributos
Límite
Dirección Base
Atributos
Límite
Dirección Base
Atributos
Límite
Dirección Base
Atributos
Límite
Dirección Base
Código
No Presente
Datos y Pila
FFFFFFFF
0

Modelo Flat Protegido



Modelo Multisegmento



Segmentación - -

Segmentación - -

- Si el procesador está seteado en el modo IA-32e, el submodo de trabajo depende del valor del atributo ***L*** del descriptor segmento de código que contiene el código actualmente en ejecución.

Segmentación - -

- Si el procesador está seteado en el modo IA-32e, el submodo de trabajo depende del valor del atributo ***L*** del descriptor segmento de código que contiene el código actualmente en ejecución.
- Si ***L*** = 0 el procesador está en el sub modo Compatibilidad con lo cual, el tratamiento de los segmentos por parte del procesador es similar al del modo IA-32.

Segmentación - -

- Si el procesador está seteado en el modo IA-32e, el submodo de trabajo depende del valor del atributo ***L*** del descriptor segmento de código que contiene el código actualmente en ejecución.
- Si ***L*** = 0 el procesador está en el sub modo Compatibilidad con lo cual, el tratamiento de los segmentos por parte del procesador es similar al del modo IA-32.
- En el caso que ***L*** = 1, el procesador está en el sub modo de 64 bits, y si bien mantiene la segmentación, prácticamente la deshabilita, generando un espacio lineal FLAT de 64 bits, en el que CS, SS, DS, y ES, tienen una dirección base 0, de modo que la ***dirección lineal*** se iguala a la efectiva u offset dentro del segmento. Los registros FS y GS, son la excepción pudiendo definir direcciones base diferentes. Esta diferencia en el tratamiento tiene por objeto facilitar algunas operaciones del sistema operativo y acceso a datos locales.

Chequeo del límite en modo IA-32e

Sub-modo 64 bits

Chequeo del límite en modo IA-32e

Sub-modo 64 bits

- El procesador no chequea el Límite del segmento. Vale decir que en este modo los registros SS, DS, y ES, se ignoran prácticamente, por lo tanto, sus registros cache Hidden son ignorados.

Chequeo del límite en modo IA-32e

Sub-modo 64 bits

- El procesador no chequea el Límite del segmento. Vale decir que en este modo los registros SS, DS, y ES, se ignoran prácticamente, por lo tanto, sus registros cache Hidden son ignorados.
- Cualquier referencia a ellos en una ***dirección lógica*** se trata como si su base fuese cero siempre. En base a ello, algunas operaciones de carga de registros de segmento resultan inválidas, como Mov Sreg, POP Sreg, LDS, LES y otras.

Chequeo del límite en modo IA-32e

Sub-modo 64 bits

- El procesador no chequea el Límite del segmento. Vale decir que en este modo los registros SS, DS, y ES, se ignoran prácticamente, por lo tanto, sus registros cache Hidden son ignorados.
- Cualquier referencia a ellos en una ***dirección lógica*** se trata como si su base fuese cero siempre. En base a ello, algunas operaciones de carga de registros de segmento resultan inválidas, como Mov Sreg, POP Sreg, LDS, LES y otras.
- Por lo demás la generación de una ***dirección lineal*** es similar a lo explicado en el modo 32 bits, solo que al ser la base 0, siempre coincide con el offset, y el resultado es una dirección de 64 bits que deberá estar en formato canónico, ya que de lo contrario el procesador generará una excepción #GP.

Carga de descriptores

Sub-modo Compatibilidad

Carga de descriptores

Sub-modo Compatibilidad

- En el Sub Modo Compatibilidad, las instrucciones de carga de registros de segmento funcionan normalmente en cuanto al procedimiento de búsqueda del descriptor en la tabla *GDT*o *LDT*, su lectura por parte del procesador, y escritura de los campos de los registros Hidden del registro de segmento en cuestión acorde con los valores de los campos Base Límite y Atributos del descriptor leído. No obstante, los valores de los registros Hidden son ignorados durante la ejecución del código.

Carga de descriptores

Sub-modo Compatibilidad

- En el Sub Modo Compatibilidad, las instrucciones de carga de registros de segmento funcionan normalmente en cuanto al procedimiento de búsqueda del descriptor en la tabla *GDT*o *LDT*, su lectura por parte del procesador, y escritura de los campos de los registros Hidden del registro de segmento en cuestión acorde con los valores de los campos Base Límite y Atributos del descriptor leído. No obstante, los valores de los registros Hidden son ignorados durante la ejecución del código.
- Cuando se sobre escriben en un programa los registros FS y GS, se calcula la ***dirección lineal*** en base al valor de dirección base almacenado en el registro cache Hidden del registro de segmento. Puede resultar que el valor de una vuelta alrededor del la dirección tope. Lo importante es que el valor resultante esté en formato canónico.

Carga de descriptores

Sub-modo Compatibilidad

Carga de descriptores

Sub-modo Compatibilidad

- Cuando se usan los prefijos FS y GS para sobre escribir el default de la instrucción no se chequea límite ni atributos.

Carga de descriptores

Sub-modo Compatibilidad

- Cuando se usan los prefijos FS y GS para sobre escribir el default de la instrucción no se chequea límite ni atributos.
- Las cargas normales de FS y GS. cargan un valor standard de 32 bits en el campo base del registro cache Hidden del registro y ponen el resto en 0 para mantener consistencia con implementaciones que usen menos de 64 bits.

Carga de descriptores

Sub-modo Compatibilidad

- Cuando se usan los prefijos FS y GS para sobre escribir el default de la instrucción no se chequea límite ni atributos.
- Las cargas normales de FS y GS. cargan un valor standard de 32 bits en el campo base del registro cache Hidden del registro y ponen el resto en 0 para mantener consistencia con implementaciones que usen menos de 64 bits.
- Los campos Base del descriptor se escriben en MSRs para mantener los 64 bits de la dirección. Se trata de IA32_KernelGSbase y IA32_KernelFSbase. Cualquier programa que ejecute con Privilegio '00', puede ejecutar la instrucción WRMSR para escribir la dirección base completa. Si esta dirección escrita en el MSR no está en formato canónico, el resultado es una excepción #GP.

Carga de descriptores

Sub-modo Compatibilidad

- Cuando se usan los prefijos FS y GS para sobre escribir el default de la instrucción no se chequea límite ni atributos.
- Las cargas normales de FS y GS. cargan un valor standard de 32 bits en el campo base del registro cache Hidden del registro y ponen el resto en 0 para mantener consistencia con implementaciones que usen menos de 64 bits.
- Los campos Base del descriptor se escriben en MSRs para mantener los 64 bits de la dirección. Se trata de IA32_KernelGSbase y IA32_KernelFSbase. Cualquier programa que ejecute con Privilegio '00', puede ejecutar la instrucción WRMSR para escribir la dirección base completa. Si esta dirección escrita en el MSR no está en formato canónico, el resultado es una excepción #GP.
- En sub modo compatibilidad, los registros FS y GS utilizan solo los 32 bits menos significativos del campo base que esté escrito en el registro cache Hidden.

Carga de descriptores

Sub Modo 64 bits

Carga de descriptores

Sub Modo 64 bits

- En Modo 64 bits se incluye una instrucción nueva SWAPGS para cargar la base completa del segmento que se selecciona con el registro GS.

Carga de descriptores

Sub Modo 64 bits

- En Modo 64 bits se incluye una instrucción nueva SWAPGS para cargar la base completa del segmento que se selecciona con el registro GS.
- Respecto de las tablas de descriptores, en modo IA-32e, el registro GDTR se expande a 80 bits para poder almacenar una base de 64 bits para la *GDT*. Lo mismo ocurre con el registro cache Hiden del LDTR.

Carga de descriptores

Sub Modo 64 bits

- En Modo 64 bits se incluye una instrucción nueva SWAPGS para cargar la base completa del segmento que se selecciona con el registro GS.
- Respecto de las tablas de descriptores, en modo IA-32e, el registro GDTR se expande a 80 bits para poder almacenar una base de 64 bits para la *GDT*. Lo mismo ocurre con el registro cache Hiden del LDTR.
- Las tablas de descriptores están dimensionadas para almacenar 2^{13} estructuras de 8 bytes de tamaño. Es decir, que cada entrada a la tabla de descriptores corresponde a 8 bytes. Muchos descriptores de sistema utilizan 16 bytes para poder incluir el campo Base de 64 bits. En tal caso ocuparán el espacio de dos entradas o descriptores de segmento.

Lecturas indispensables

Lecturas indispensables

- Al tener disponibles los fuentes del sistema operativo Linux, existe abundante información de sus detalles de implementación.

Lecturas indispensables

- Al tener disponibles los fuentes del sistema operativo Linux, existe abundante información de sus detalles de implementación.
- Understanding the Linux Kernel, de Daniel Bovet y Marco Cesati, Ed. O'Reilly (¿cuando no?), cuenta con muy interesantes detalles de como se utilizan los recursos de un procesador IA-32 para implementar lo que llamaríamos el kernel de bajo nivel del Sistema Operativo.

Lecturas indispensables

- Al tener disponibles los fuentes del sistema operativo Linux, existe abundante información de sus detalles de implementación.
- Understanding the Linux Kernel, de Daniel Bovet y Marco Cesati, Ed. O'Reilly (¿cuando no?), cuenta con muy interesantes detalles de como se utilizan los recursos de un procesador IA-32 para implementar lo que llamaríamos el kernel de bajo nivel del Sistema Operativo.
- Secciones de código que trabajan directo con los registros del procesador y detalles del hardware.

Lecturas indispensables

- Al tener disponibles los fuentes del sistema operativo Linux, existe abundante información de sus detalles de implementación.
- Understanding the Linux Kernel, de Daniel Bovet y Marco Cesati, Ed. O'Reilly (¿cuando no?), cuenta con muy interesantes detalles de como se utilizan los recursos de un procesador IA-32 para implementar lo que llamaríamos el kernel de bajo nivel del Sistema Operativo.
- Secciones de código que trabajan directo con los registros del procesador y detalles del hardware.
- Tener en cuenta que la última edición disponible apareció en ocasión del lanzamiento del kernel 2.6. Este kernel ha tenido evoluciones hasta el sub-release 2.6.31, y actualmente está disponible la versión de kernel 3.7. Por lo tanto es necesario revisar si no hubo actualizaciones. Afortunadamente Linux no tiene misterios y la información es pública y accesible, esto lo hace atractivo como caso de estudio.

Hurgando en los fuentes

Hurgando en los fuentes

- A partir de la versión de kernel 2.2, (hace muchos años ya) Linux comenzó a soportar SMP (Symmetric Multi Processing), es decir la capacidad de controlas sistemas de hardware con mas de un procesador, todos iguales (por eso Symmetric).

Hurgando en los fuentes

- A partir de la versión de kernel 2.2, (hace muchos años ya) Linux comenzó a soportar SMP (Symmetric Multi Processing), es decir la capacidad de controlas sistemas de hardware con mas de un procesador, todos iguales (por eso Symmetric).
- Cuando se lanza la versión 2.4 del kernel, Linux mejora su eficiencia en el manejo SMP manteniendo una *GDT*por cada procesador presente en el sistema. En un array denominado `cpu_gdt_table` guarda las estructuras de las *GDT*, correspondiendo cada elemento del arreglo a una CPU.

Hurgando en los fuentes

- A partir de la versión de kernel 2.2, (hace muchos años ya) Linux comenzó a soportar SMP (Symmetric Multi Processing), es decir la capacidad de controlas sistemas de hardware con mas de un procesador, todos iguales (por eso Symmetric).
- Cuando se lanza la versión 2.4 del kernel, Linux mejora su eficiencia en el manejo SMP manteniendo una *GDT*por cada procesador presente en el sistema. En un array denominado `cpu_gdt_table` guarda las estructuras de las *GDT*, correspondiendo cada elemento del arreglo a una CPU.
- El código que se emplea se muestra a continuación. En la primer línea de dicho listado, que corresponde al archivo /source/linux/include/asm-i386/desc.h de los fuentes del sistema, se declara un arreglo de GDT_ENTRIES descriptores, en donde GDT_ENTRIES ha sido inicializada con la cantidad de descriptores totales a almacenar en la *GDT*menos 1, ya que el primer descriptor lleva el índice 0. Es decir, que allí se define un arreglo de descriptores que arma una *GDT*.

Hurgando en los fuentes

Hasta el momento Linux usa *GDT* de 32 entradas, definiendo en la línea 112 del archivo /source/linux/include/asm-i386/segment.h el valor a GDT_ENTRIES en 32.

Hurgando en los fuentes

Hasta el momento Linux usa *GDT* de 32 entradas, definiendo en la línea 112 del archivo /source/linux/include/asm-i386/segment.h el valor a GDT_ENTRIES en 32.

```
1 extern struct desc_struct cpu_gdt_table[GDT_ENTRIES];
2 DECLARE_PER_CPU (struct desc_struct, cpu_gdt_table[GDT_ENTRIES]);
```

Hurgando en los fuentes

Hasta el momento Linux usa *GDT* de 32 entradas, definiendo en la línea 112 del archivo /source/linux/include/asm-i386/segment.h el valor a GDT_ENTRIES en 32.

```
1 extern struct desc_struct cpu_gdt_table[GDT_ENTRIES];
2 DECLARE_PER_CPU (struct desc_struct, cpu_gdt_table[GDT_ENTRIES]);
```

Donde

```
1 #define GDT_ENTRIES 32
```

Hurgando en los fuentes

Hasta el momento Linux usa *GDT*de 32 entradas, definiendo en la línea 112 del archivo /source/linux/include/asm-i386/segment.h el valor a GDT_ENTRIES en 32.

```
1 extern struct desc_struct cpu_gdt_table[GDT_ENTRIES];
2 DECLARE_PER_CPU (struct desc_struct, cpu_gdt_table[GDT_ENTRIES]);
```

Donde

```
1 #define GDT_ENTRIES 32
```

La estructura **`desc_struct`** señalada como externa se define en el archivo fuente /source/linux/include/asm-i386/processor.h

Hurgando en los fuentes

Hasta el momento Linux usa *GDT*de 32 entradas, definiendo en la línea 112 del archivo /source/linux/include/asm-i386/segment.h el valor a GDT_ENTRIES en 32.

```
1 extern struct desc_struct cpu_gdt_table[GDT_ENTRIES];
2 DECLARE_PER_CPU (struct desc_struct, cpu_gdt_table[GDT_ENTRIES]);
```

Donde

```
1 #define GDT_ENTRIES 32
```

La estructura **desc_struct** señalada como externa se define en el archivo fuente /source/linux/include/asm-i386/processor.h

```
1 struct desc_struct {
2         unsigned long a,b;
3 };
```

Hurgando en los fuentes

Hasta el momento Linux usa *GDT*de 32 entradas, definiendo en la línea 112 del archivo /source/linux/include/asm-i386/segment.h el valor a GDT_ENTRIES en 32.

```
1 extern struct desc_struct cpu_gdt_table[GDT_ENTRIES];
2 DECLARE_PER_CPU (struct desc_struct, cpu_gdt_table[GDT_ENTRIES]);
```

Donde

```
1 #define GDT_ENTRIES 32
```

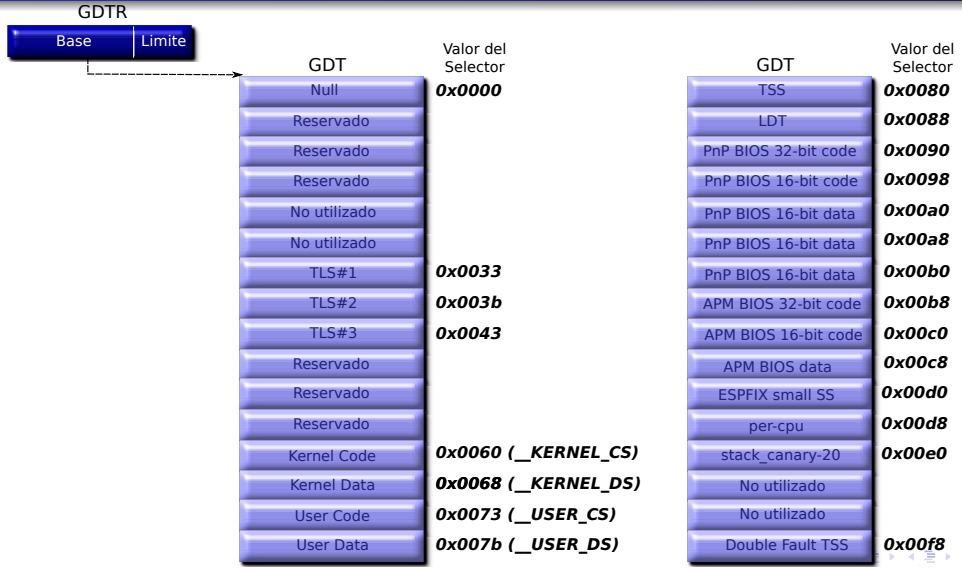
La estructura **desc_struct** señalada como externa se define en el archivo fuente /source/linux/include/asm-i386/processor.h

```
1 struct desc_struct {
2         unsigned long a,b;
3 };
```

En el archivo /source/linux/include/asm-i386/percpu.h se define.

```
1 #define DECLARE_PER_CPU(type, name) extern __typeof__(type) per_cpu__##name
```

GDT en Linux


GDTR
Base
Limite
GDT
Valor del Selector
Null
0x0000
Reservado
Reservado
Reservado
No utilizado
No utilizado
TLS#1
0x0033
TLS#2
0x003b
TLS#3
0x0043
Reservado
Reservado
Reservado
Kernel Code
0x0060 (__KERNEL_CS)
Kernel Data
0x0068 (__KERNEL_DS)
User Code
0x0073 (__USER_CS)
User Data
0x007b (__USER_DS)
GDT
Valor del Selector
TSS
0x0080
LDT
0x0088
PnP BIOS 32-bit code
0x0090
PnP BIOS 16-bit code
0x0098
PnP BIOS 16-bit data
0x00a0
PnP BIOS 16-bit data
0x00a8
PnP BIOS 16-bit data
0x00b0
APM BIOS 32-bit code
0x00b8
APM BIOS 16-bit code
0x00c0
APM BIOS data
0x00c8
ESPFIX small SS
0x00d0
per-cpu
0x00d8
stack_canary-20
0x00e0
No utilizado
No utilizado
Double Fault TSS
0x00f8

Modelo de segmentación en Linux

- Los selectores de segmento 0x0060, y 0x0068, definidos por las macros __KERNEL_CS, y __KERNEL_DS, corresponden a los descriptores de código y datos respectivamente del kernel.
- Por su parte los selectores 0x0073 y 0x007b definidos por las macros __USER_CS, y __USER_DS, corresponden a los descriptores de código y datos respectivamente de modo usuario.
- Además estos descriptores tiene los siguientes valores dentro de la tabla siguiente.

Selector	Base	G	Límite	S	Tipo	DPL	D/B	P
__USER_CS	0x00000000	1	0xfffff	1	1010	11	1	1
__USER_DS	0x00000000	1	0xfffff	1	0010	11	1	1
__KERNEL_CS	0x00000000	1	0xfffff	1	1010	00	1	1
__KERNEL_DS	0x00000000	1	0xfffff	1	0010	00	1	1

Conclusión: Linux utiliza un modelo de segmentación FLAT Básico.

Eso es todo

¿Preguntas?