

## Procesador IA-32 - Gestión de Memoria

Alejandro Furfaro

Abril de 2013

# Temario

## 1 Como se organiza la memoria

- Modelo de memoria en Modo Protegido
- Modelo de memoria en Modo 64 bits

## 2 Direcciones Lógicas y Lineales

- Traducción de direcciones Lógicas

## 3 Unidad de Segmentación

- Selectores de segmento
- Descriptores de segmento de 32 bits

## 4 Generación de la dirección Lineal (32 bits)

## 5 Modelos de segmentación de memoria

- Segmentación en Modo IA-32e
- Implementación práctica de segmentación en un SO

## 6 Paginación

- Introducción

# 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 continuación 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

### 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



- 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

- 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

- Como hemos visto, se puede ingresar a un modo denominado IA-32e, en el que se trabaja con una arquitectura de 64 bits
- Se generan direcciones lineales de 64 bits, de los que por el momento se utilizan los 48 menos significativos.
- Se define como formato de dirección canónica, al que tiene todos sus bits de la **dirección lineal** desde el 63 al bit de orden inmediato mayor al más significativo en '0', o todos en '1'.
- Por cada acceso a memoria el procesador chequea el formato canónico de la **dirección lineal**

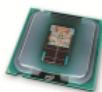


# Formato Canónico

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 Página 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 igual que el espacio físico un rango de direcciones contiguas planas, que comienza 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

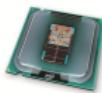
- El procesador debe tener la capacidad de especificar a cada proceso o tarea, donde comienza cada uno de sus segmentos y cual es su tamaño.
- La dirección de comienzo de un segmento es un valor de 32 bits en modo protegido.
- El tamaño máximo que puede tener un segmento es el de su máximo valor de desplazamiento, 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.



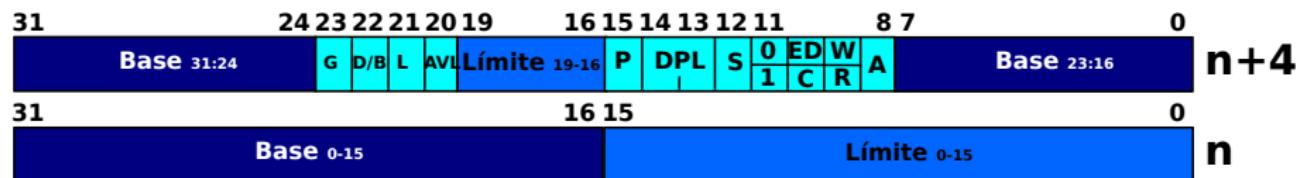
# Formato



- **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 qué 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

- **G.** Granularity. 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** \* 0x400 +0x3FF.
- **D/B.** Default / Big. 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

- **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.** **AVaiLable**. 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.** **Present**. 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 accesible en RAM.



# Atributos

- **DPL.** Descriptor Priviledge Level. Indica el nivel de privilegio del segmento que debe tener el segmento que contiene el código que pretende acceder a éste otro segmento.
- **S.** System. Este bit, **activo bajo** permite administrar en las tablas de descriptoros, dos clases bien determinadas de segmentos:
  - 1 Segmentos de Código o Datos
  - 2 Segmentos de Sistema, que tienen diferentes tipos de formato y que en general no se refieren a zonas de memoria (salvo algún caso aislado), sino que en general se refieren a mecanismos de uso de recursos del procesador por parte del kernel (por ello reciben el nombre de descriptoros de Sistema, ya que **son potestad exclusiva 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

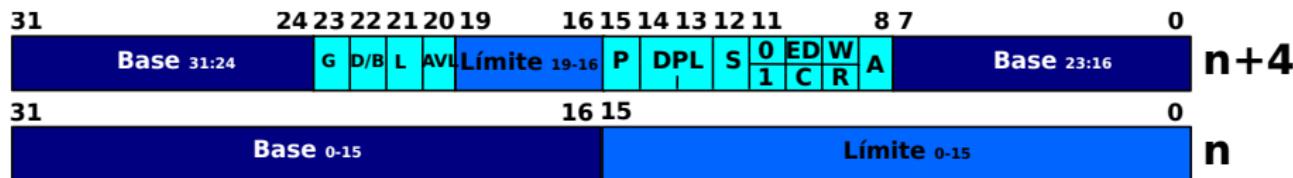


# Tipos de Descriptoros de Sistema ( $S = 0$ )

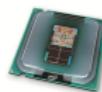
11	10	9	8	Modo 32 bits	Modo IA-32e
0	0	0	0	Reservado	8 bytes superiores en un segmento de 16 bytes
0	0	0	1	TSS de 16 bits disponible	Reservado
0	0	1	0	LDT	LDT
0	0	1	1	TSS de 16 bits Busy	Reservado
0	1	0	0	Call Gate de 16 bits	Reservado
0	1	0	1	Task Gate	Reservado
0	1	1	0	Interrupt Gate de 16 bits	Reservado
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 Descriptoros 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 Descriptoros de Código( $S = 1$ , $Bit_{11} = 1$ )

- **C.** Conforming. 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.** Readable. 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 Descriptoros de Datos( $S = 1$ , $Bit_{11} = 0$ )

- **ED.** Expand Down. 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:

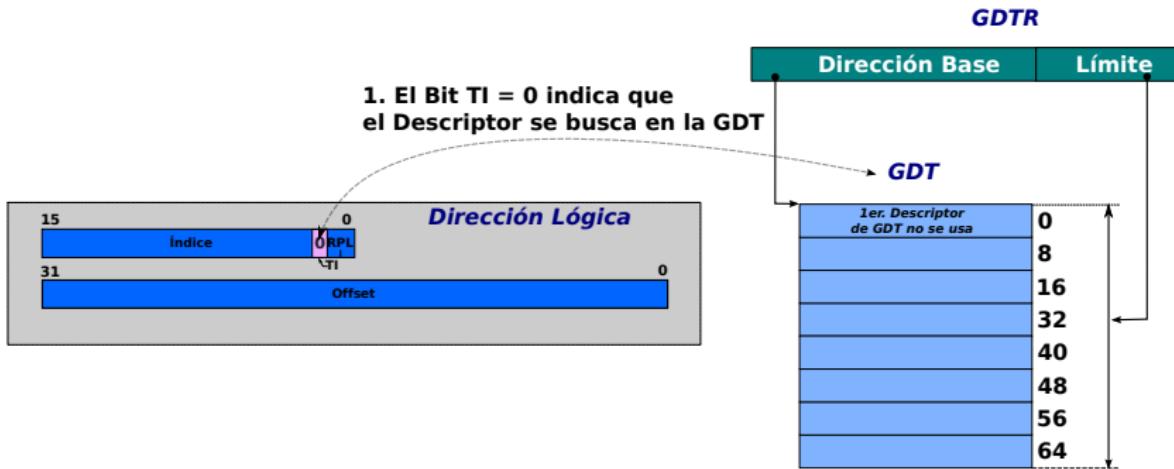
- (límite efectivo +1) hasta 0FFFFh, si el bit **D/B** = 0.
- (límite efectivo +1) hasta 0xFFFFFFFFh, si el bit **D/B** = 1.

Se ampliará al tocar el tema protección.

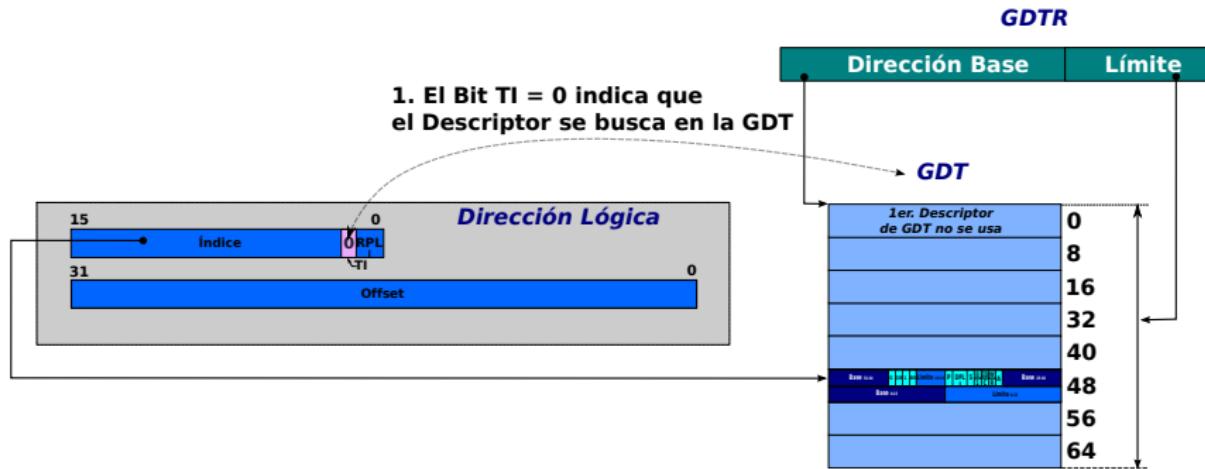
- **W.** Writable. Este bit, indica si el segmento de datos puede escribirse. Si este bit está en '0', el segmento contiene datos pero es Read Only.



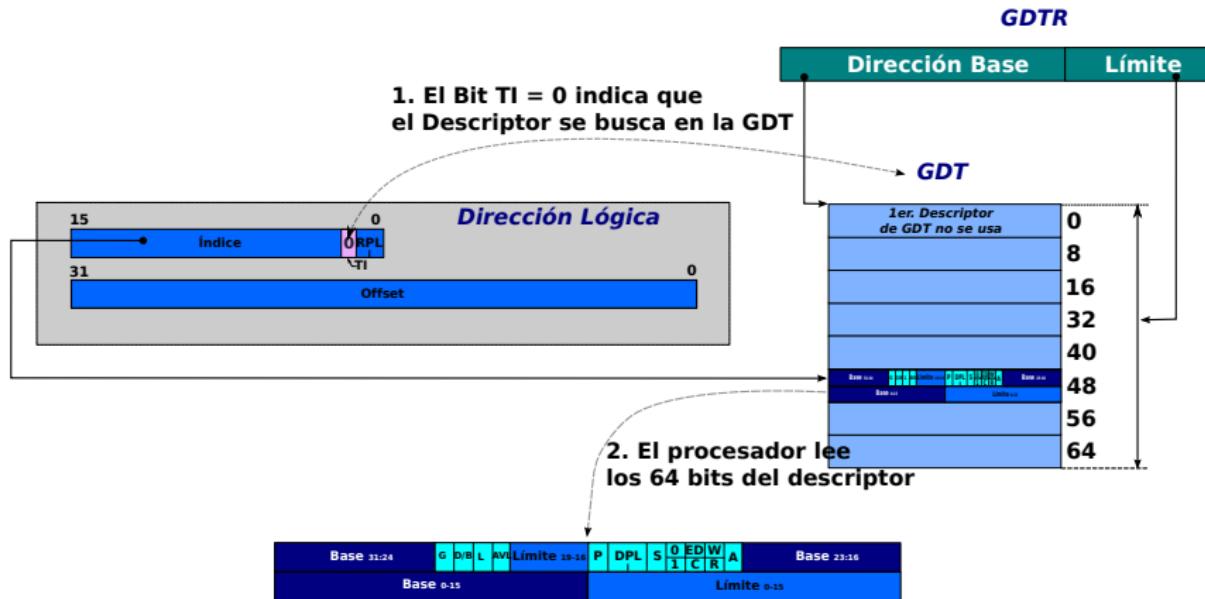
# Descriptoros de Segmento en la GDT ( $TI = 0$ )



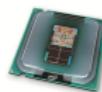
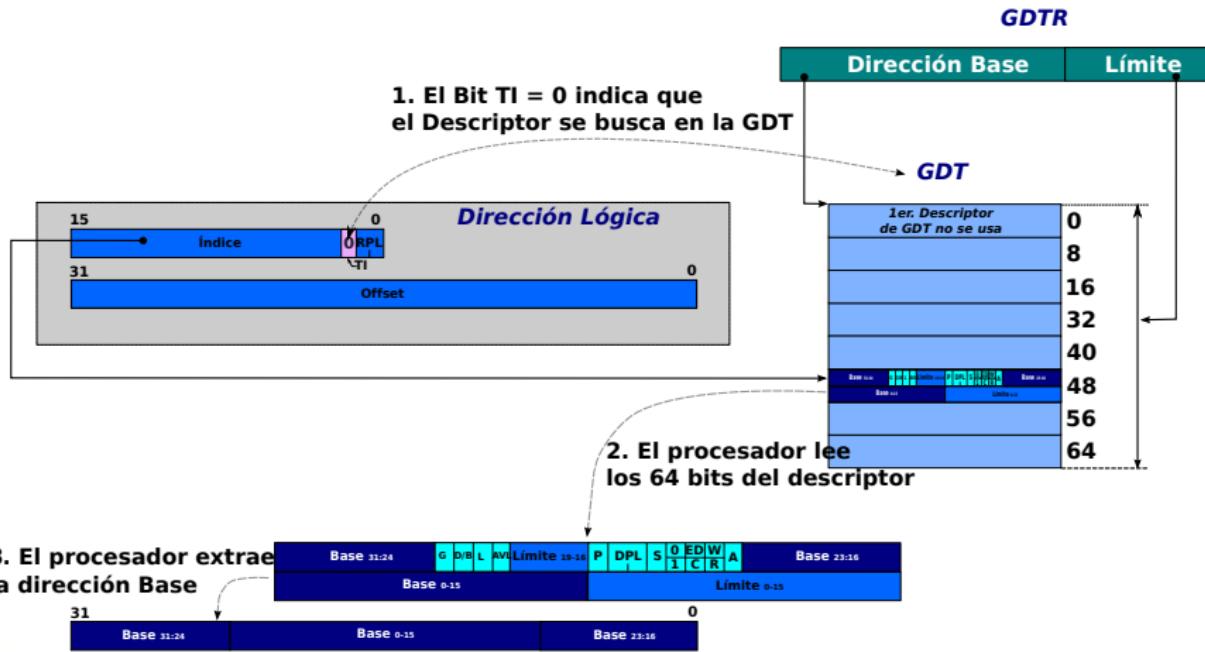
# Descriptoros de Segmento en la GDT ( $TI = 0$ )



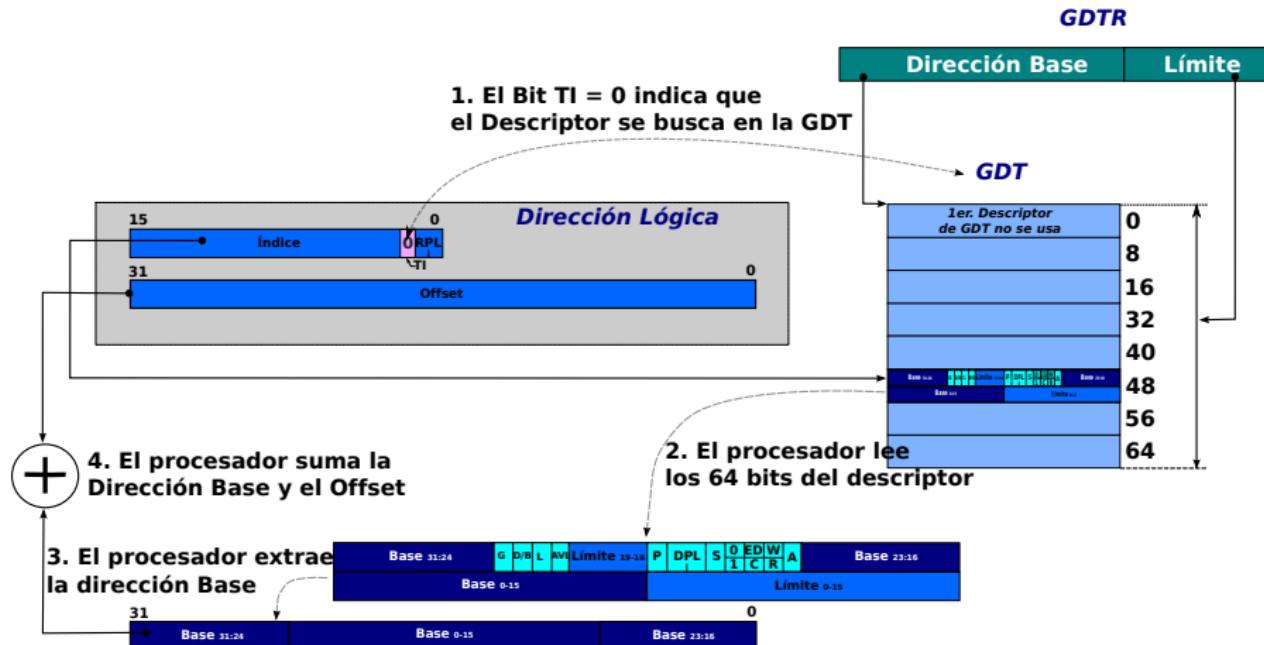
# Descriptoros de Segmento en la GDT( $TI = 0$ )



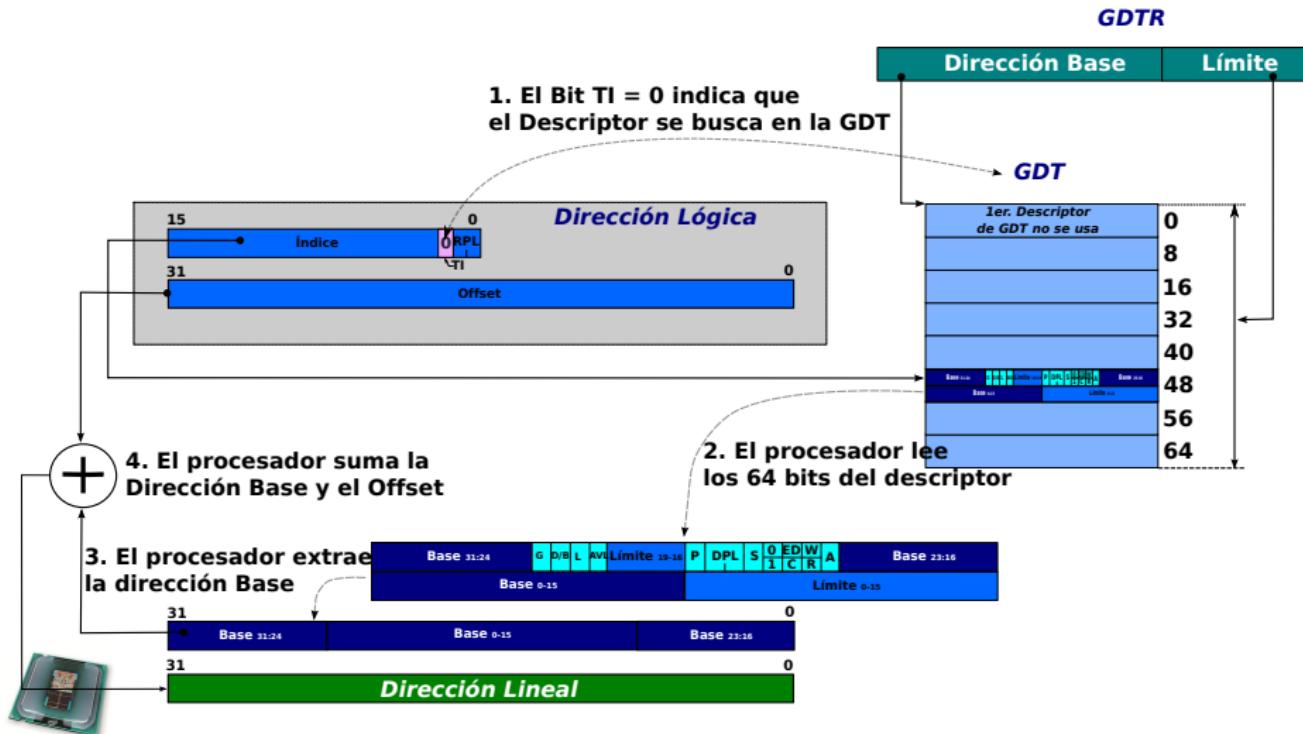
# Descriptoros de Segmento en la GDT( $TI = 0$ )



# Descriptoros de Segmento en la GDT( $TI = 0$ )



# Descriptoros de Segmento en la GDT ( $TI = 0$ )



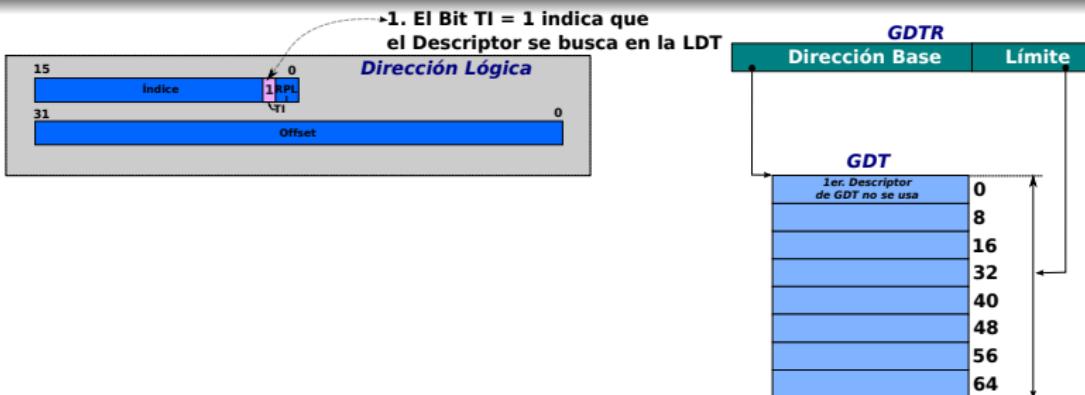
# Descriptoros de Segmento en la *GDT* (*TI* = 0)

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:  $GDT.\text{Base} + 8 * \text{Index}$  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**



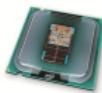
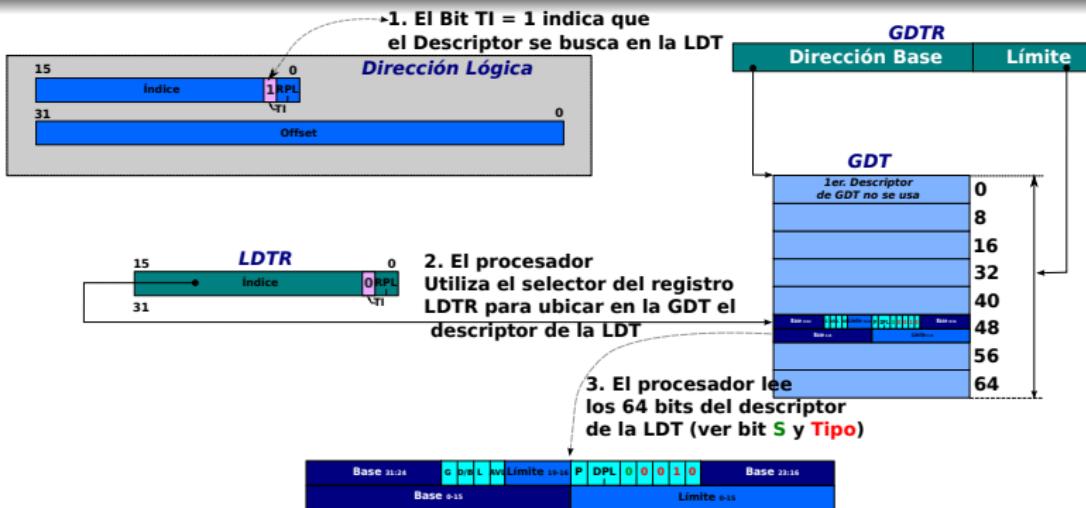
# Descriptoros de Segmento en la LDT ( $TI = 1$ )



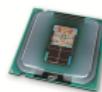
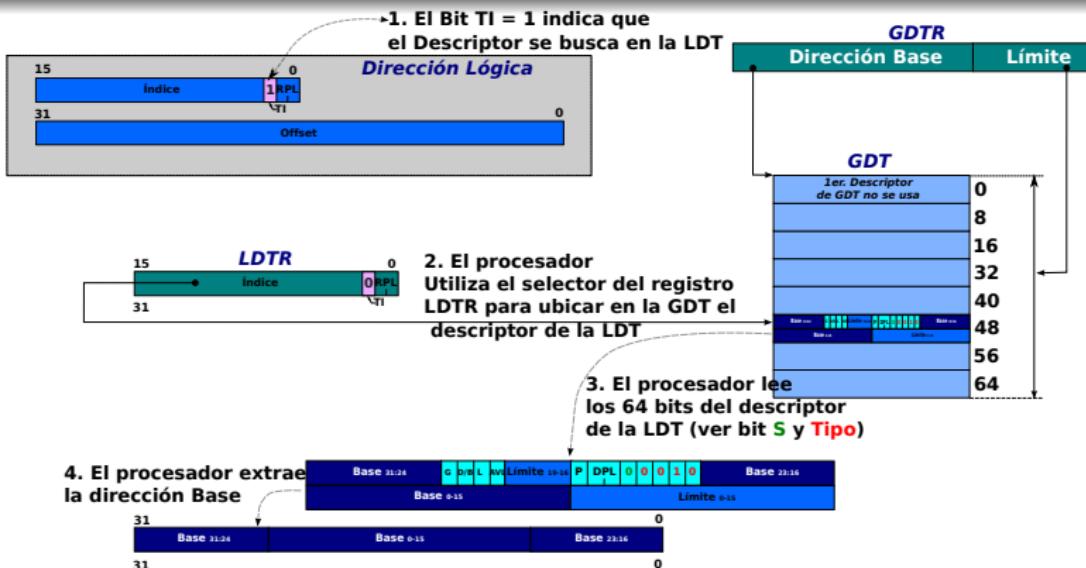
# Descriptoros de Segmento en la LDT ( $TI = 1$ )



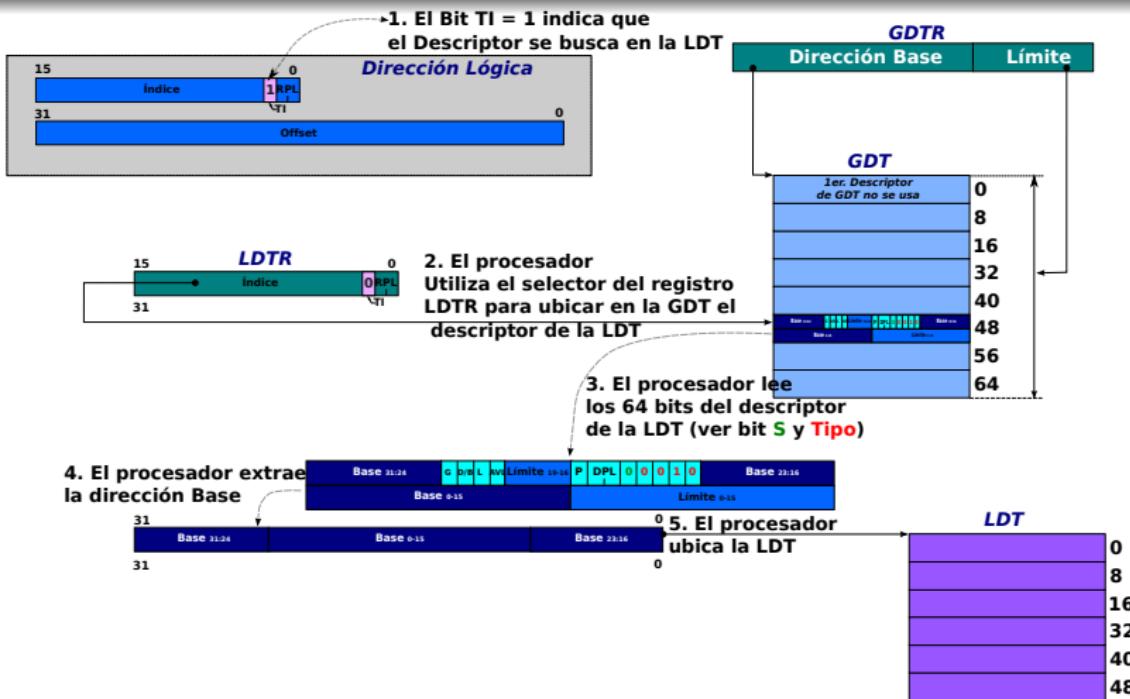
# Descriptoros de Segmento en la LDT ( $TI = 1$ )



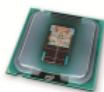
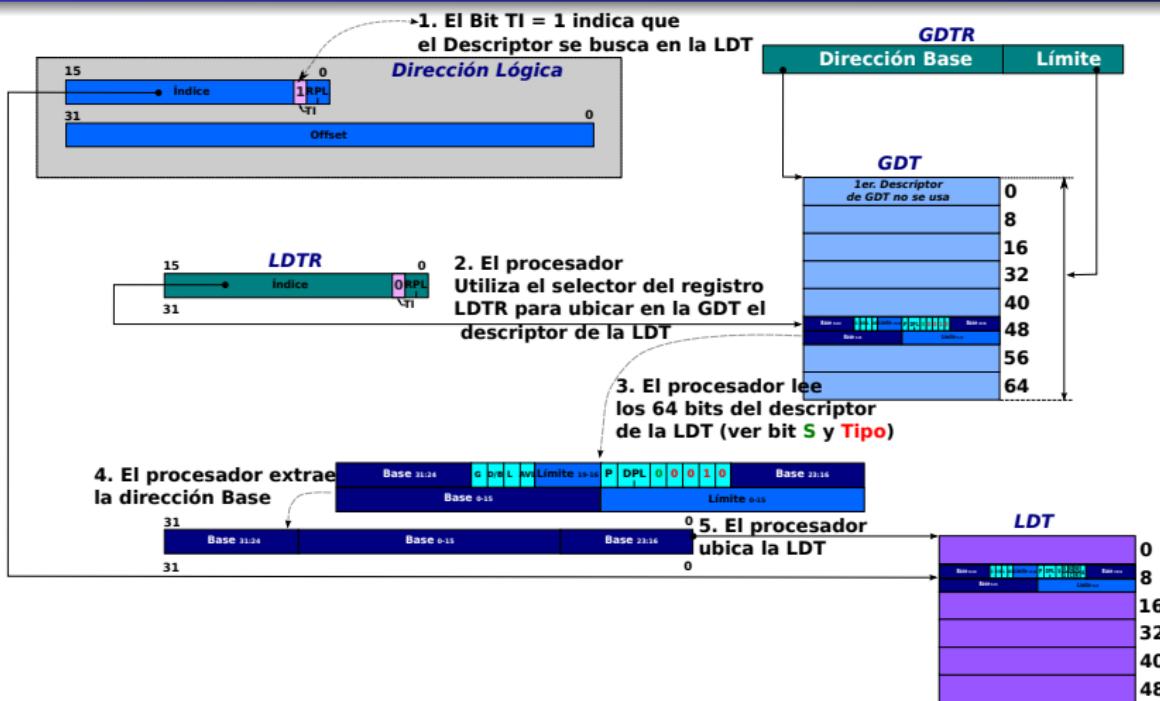
# Descriptoros de Segmento en la LDT ( $TI = 1$ )



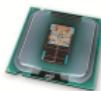
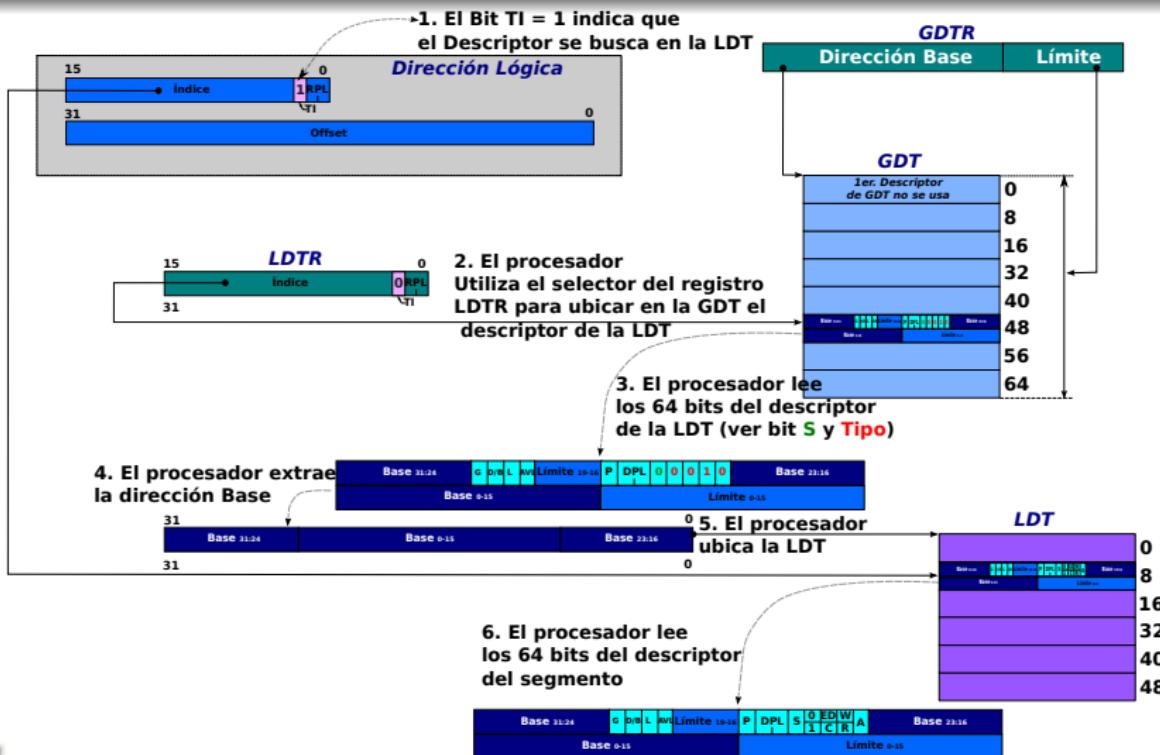
# Descriptoros de Segmento en la LDT ( $TI = 1$ )



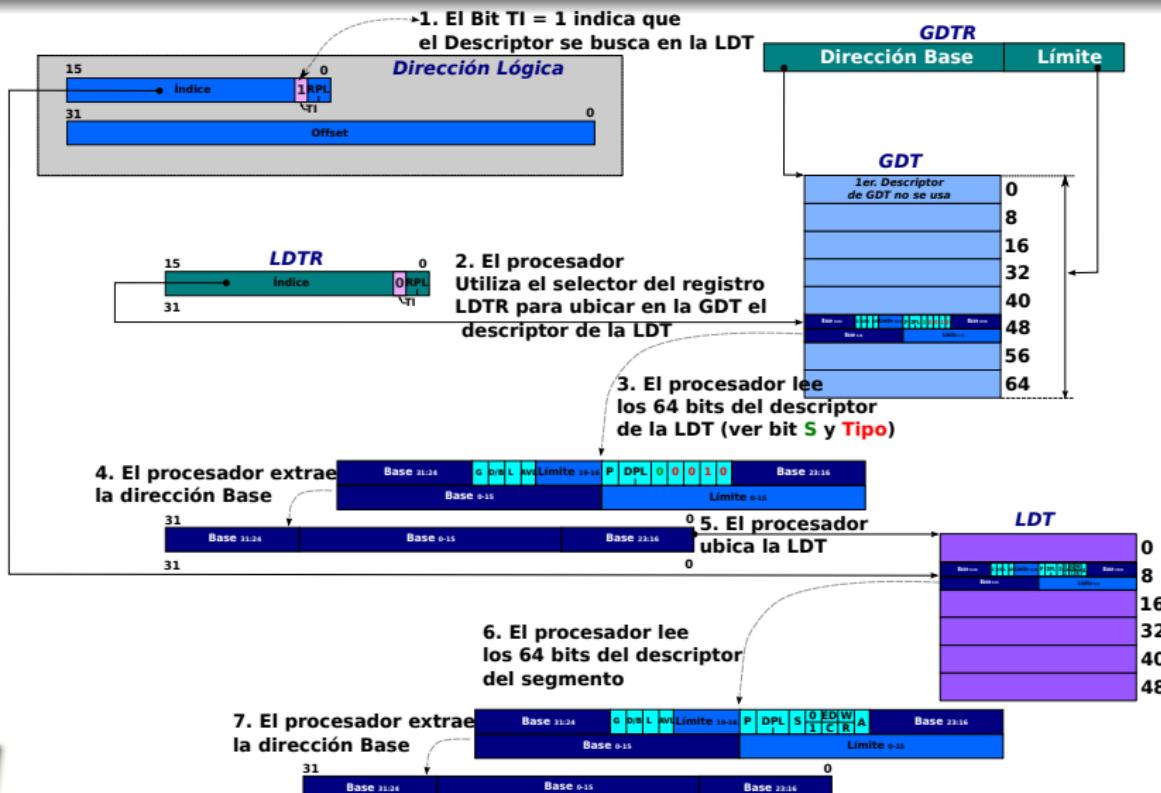
# Descriptoros de Segmento en la LDT ( $TI = 1$ )



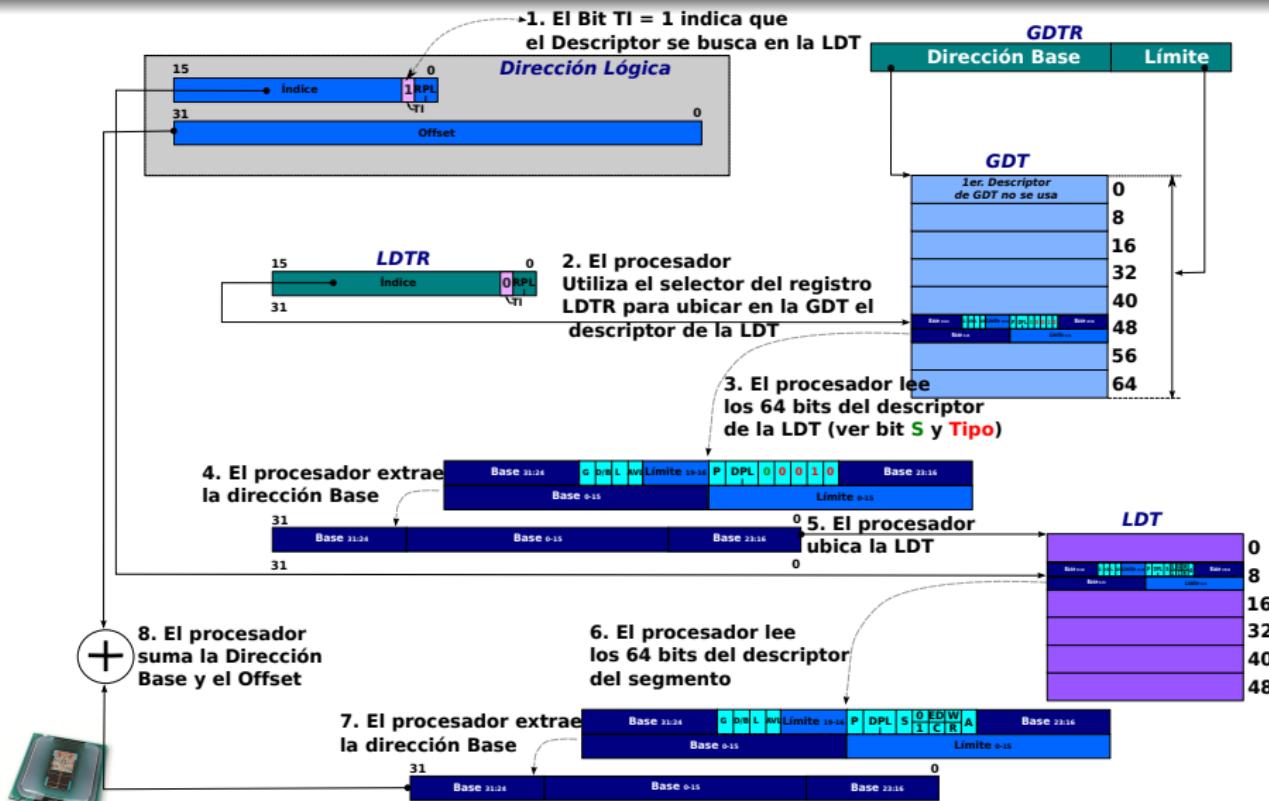
# Descriptoros de Segmento en la LDT ( $TI = 1$ )



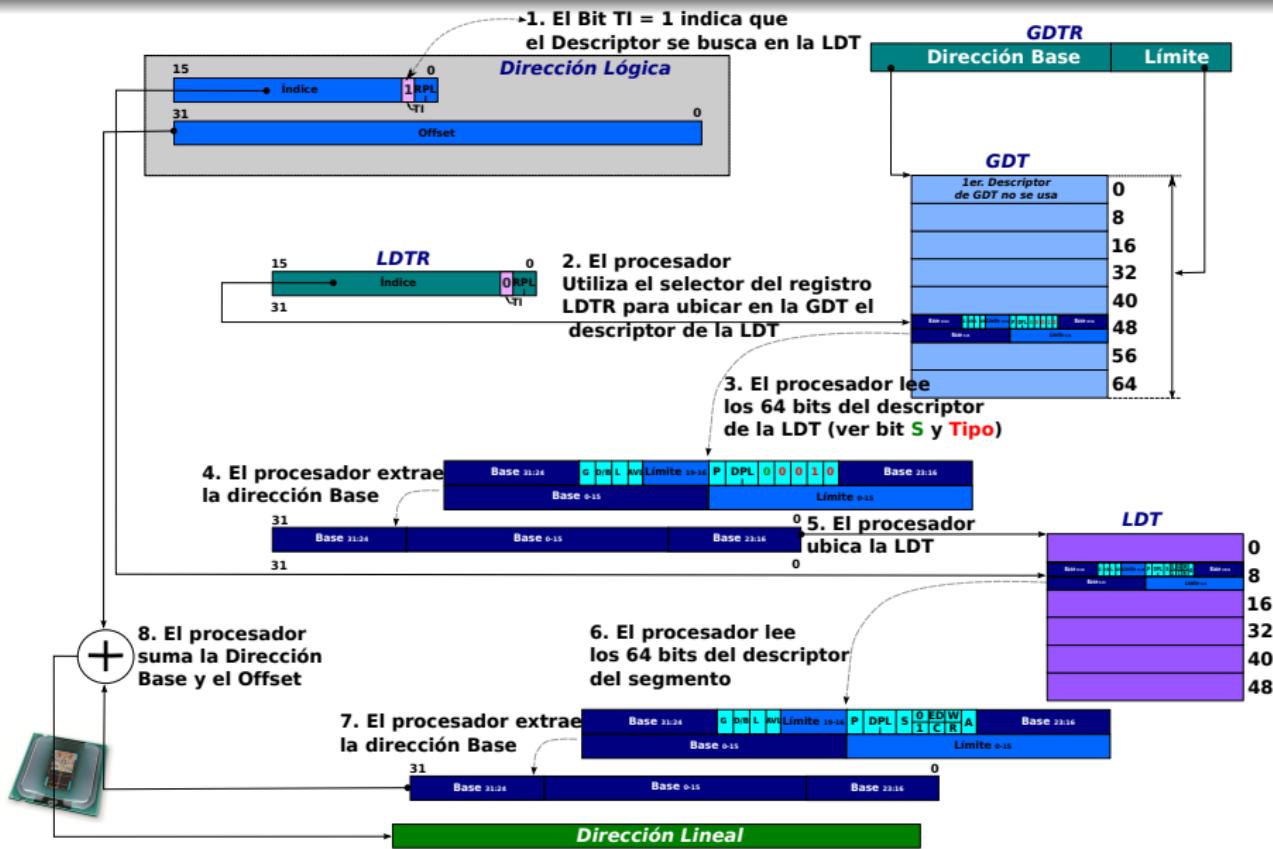
# Descriptoros de Segmento en la LDT ( $TI = 1$ )



# Descriptoros de Segmento en la LDT ( $TI = 1$ )



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



# Descriptoros 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:  $GDT.\text{Base} + 8 * \text{Index}$  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).



# Descriptoros 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*

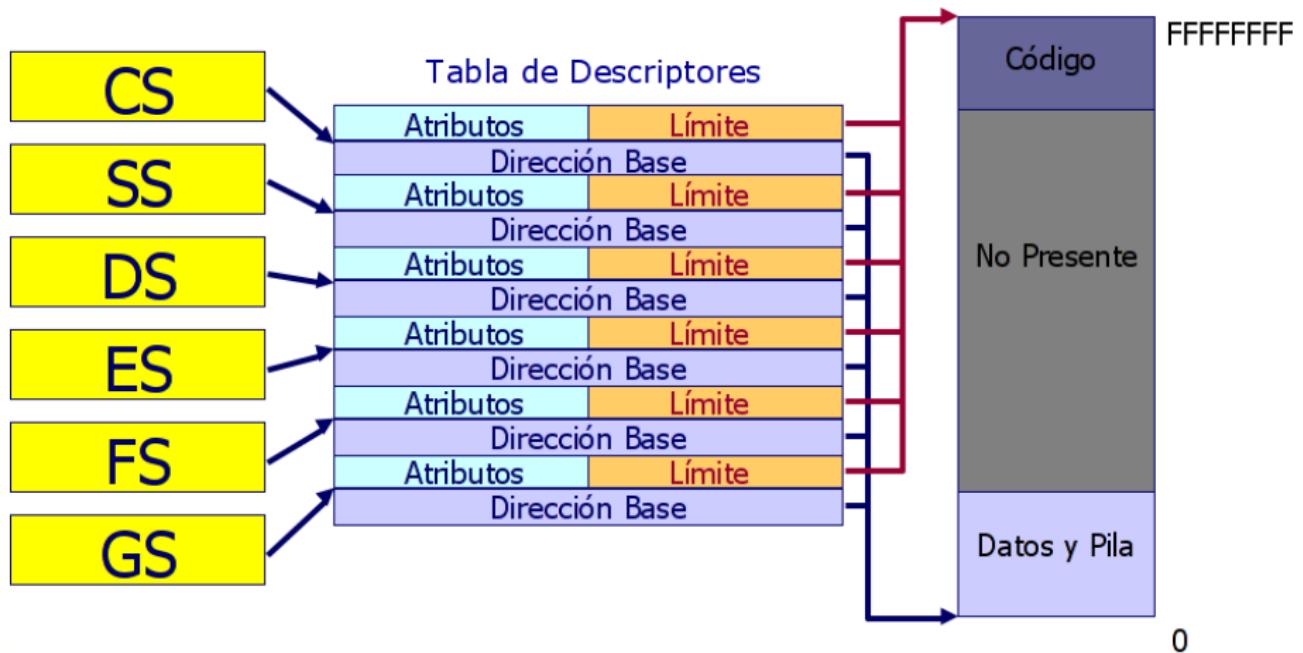


# Registros cache ocultos (hidden)

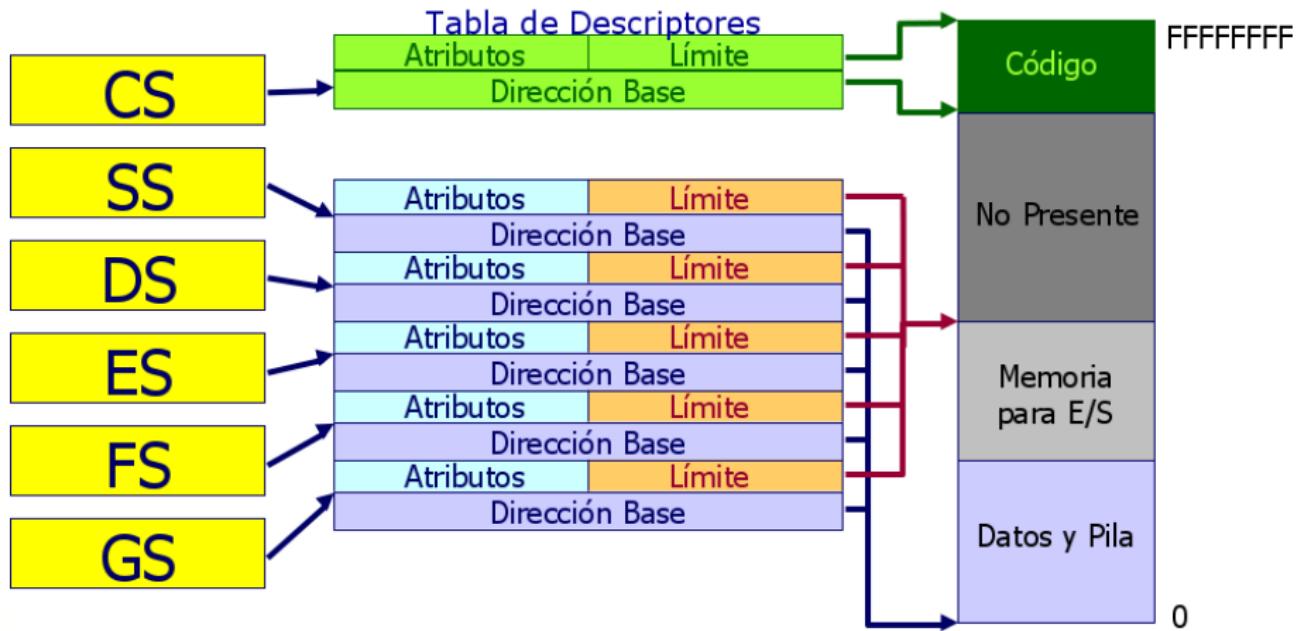
CS	Dirección Base	Límite	Atributos
SS	Dirección Base	Límite	Atributos
DS	Dirección Base	Límite	Atributos
ES	Dirección Base	Límite	Atributos
FS	Dirección Base	Límite	Atributos
GS	Dirección Base	Límite	Atributos
LDTR	Dirección Base	Límite	Atributos
TR	Dirección Base	Límite	Atributos

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.

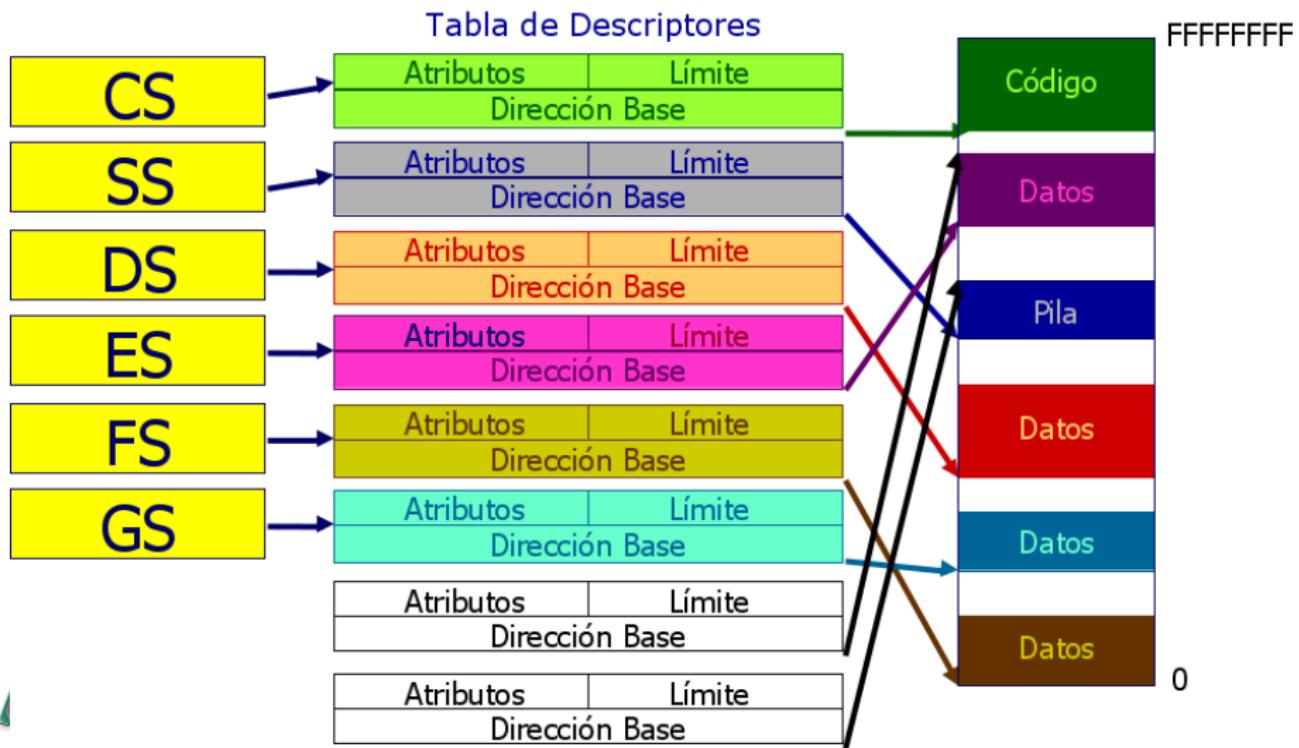
# Modelo Flat



# Modelo Flat Protegido



# Modelo Multisegmento



# 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

- 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

- 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

- 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 **CR2** y **CR4**. 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.

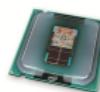
# Carga de descriptores

## Sub Modo 64 bits

- 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

- Al tener disponibles las 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

- 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 controlar 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.

```
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



GDT	Valor del Selector
Null	<b>0x0000</b>
Reservado	
Reservado	
Reservado	
No utilizado	
No utilizado	
TLS#1	<b>0x0033</b>
TLS#2	<b>0x003b</b>
TLS#3	<b>0x0043</b>
Reservado	
Reservado	
Reservado	
Kernel Code	<b>0x0060 (_KERNEL_CS)</b>
Kernel Data	<b>0x0068 (_KERNEL_DS)</b>
User Code	<b>0x0073 (_USER_CS)</b>
User Data	<b>0x007b (_USER_DS)</b>

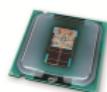
GDT	Valor del Selector
TSS	<b>0x0080</b>
LDT	<b>0x0088</b>
PnP BIOS 32-bit code	<b>0x0090</b>
PnP BIOS 16-bit code	<b>0x0098</b>
PnP BIOS 16-bit data	<b>0x00a0</b>
PnP BIOS 16-bit data	<b>0x00a8</b>
PnP BIOS 16-bit data	<b>0x00b0</b>
APM BIOS 32-bit code	<b>0x00b8</b>
APM BIOS 16-bit code	<b>0x00c0</b>
APM BIOS data	<b>0x00c8</b>
ESPFIX small SS	<b>0x00d0</b>
per-cpu	<b>0x00d8</b>
stack_canary-20	<b>0x00e0</b>
No utilizado	
No utilizado	
Double Fault TSS	<b>0x00f8</b>



# 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
__USER_CS	0x000000000	1	0xfffff	1	1010	11	1
__USER_DS	0x000000000	1	0xfffff	1	0010	11	1
__KERNEL_CS	0x000000000	1	0xfffff	1	1010	00	1
__KERNEL_DS	0x000000000	1	0xfffff	1	0010	00	1



**Conclusión:** Linux utiliza un modelo de segmentación FLAT Básico.

# Descriptoros de Sistema

## 1 TSS

- Un descriptor de **TSS** único por cada CPU. (ampliaremos su significado al abordar el tema Tareas).
- Linux mantiene un array denominado **init\_tss**
- Cada descriptor de **TSS** contendrá una dirección Base que apunta al la dirección de inicio del TSS de esa CPU dentro de este array.

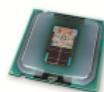
## 2 Un Descriptor de **LDT** default común para todas las tareas por cada CPU. En principio no se utilizan segmentos locales.

- Este array se define en el archivo `linux/include/asm-i386/desc.h` con la línea que se muestra en el listado siguiente.

```
1 extern struct desc_struct default_Idt[];
```

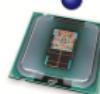
- Por otra parte, en el archivo `/source/linux/arch/i386/kernel/traps.c` se inicializa una **LDT** por default para cada CPU que se incluirá al armar la estructura de tablas. Se muestra en el listado siguiente. Como vemos se genera un arreglo de 5 descriptoros nulos.

```
1 struct desc_struct default_Idt[] = { { 0, 0 }, { 0, 0 }
2 , { 0, 0 }, { 0, 0 } };
3 { 0, 0 }, { 0, 0 } };
```



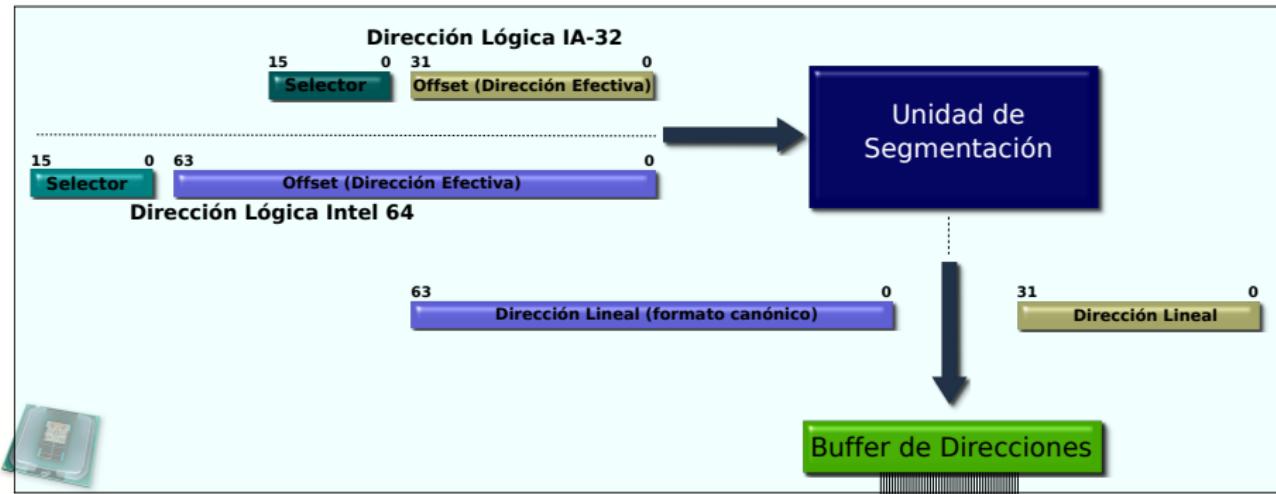
# Paginación de la Memoria

- La segmentación posee claras ventajas para proveer un entorno flexible en la programación de aplicaciones. Sin embargo, para Administración de la memoria por parte del sistema operativo, la variabilidad del tamaño de los segmentos introduce complejidad en los algoritmos de un sistema de memoria virtual.
- La Paginación en cambio, si bien es mas rígido como sistema de programación de aplicaciones, al trabajar con bloques del mismo tamaño, hace mucho mas sencillo el desarrollo del algoritmo de memoria virtual.
- Esto ha hecho que tradicionalmente los sistemas operativos hayan diseñado su sistema de memoria virtual mediante la administración de bloques de memoria de tamaño fijo.
- De hecho, UNIX por citar al decano de los Sistemas Operativos, desde su concepción implementó la administración de la memoria utilizando bloques de tamaño uniforme.



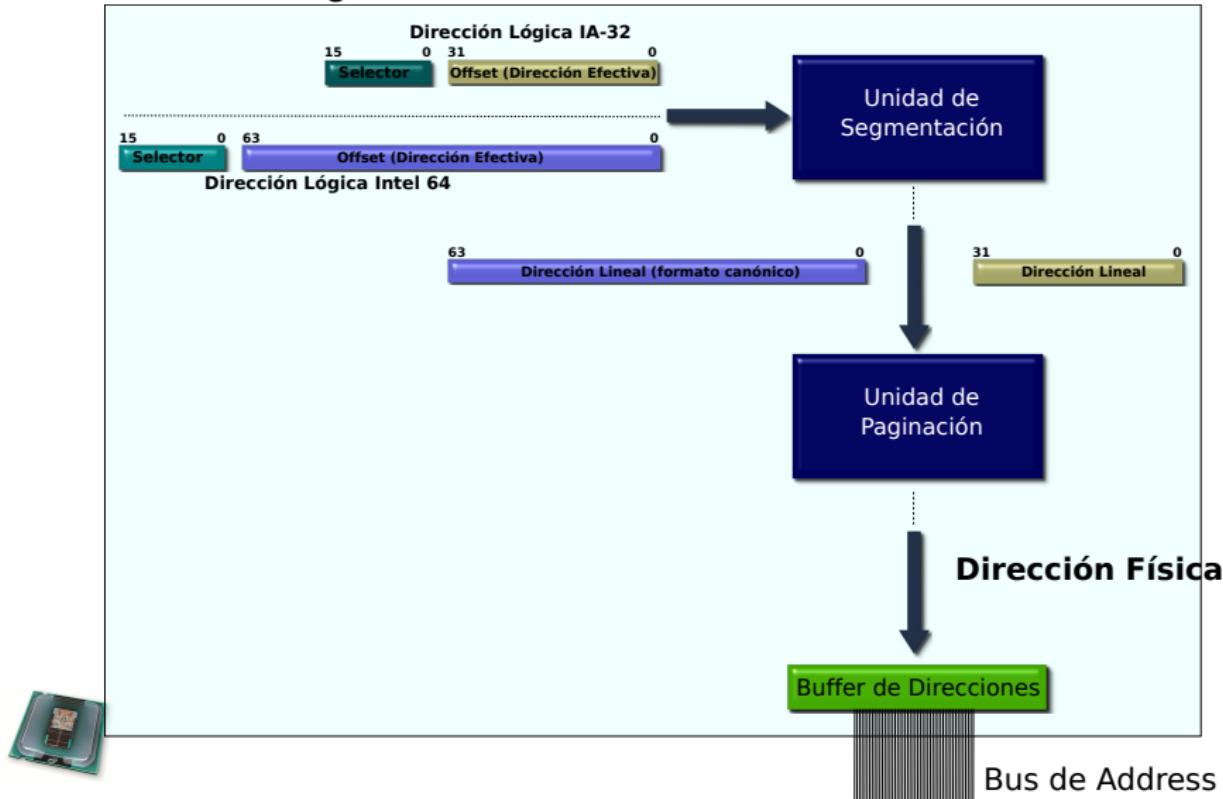
# Diagrama de generación de la dirección física

- Un segmento es finalmente un espacio lineal de direcciones, razón por la cual se denomina a este número de 32 o 64 bit obtenido por la Unidad de Segmentación, **dirección lineal**
- De no mediar otra etapa, de hardware, este número sale por el bus de Address convertido en **dirección física**



# Diagrama de generación de la dirección física

La Unidad de Paginación debe habilitarse seteando el bit **CR0.PG**



# Unidad de Paginación IA-32

La Unidad de Paginación, permitió a los procesadores IA-32 :

- Ejecutar un Sistema Operativo UNIX,
- Fraccionar el espacio lineal de un segmento en páginas de menor tamaño, permitiendo al Sistema Operativo tener en RAM únicamente unas pocas páginas de cada segmento, dejando el resto en la Memoria Virtual para ser intercambiadas cuando se las necesite. No requiere el segmento completo el RAM.
- Así el Sistema Operativo puede alojar en memoria mayor cantidad de procesos, ya que cada uno requiere menos memoria para iniciar su ejecución.
- Si consideramos que el software por lo general se compone de una serie de sentencias de iteración, la conclusión es que una página de código puede ser ejecutada durante muchos ciclos de clock antes de requerirse otra. Por lo tanto la frecuencia con que cada proceso bien programado requeriría bajar desde la Memoria Virtual otra página a la memoria Física, será bastante moderada.



# Modos de paginación

- 32 Bits** Se trata del modo original de paginación con páginas de 4 Kbytes al que se le agrega un modo de extensión de memoria física y tamaño de página introducido con el procesador Pentium Pro, denominado PSE (Page Size Extention).
- PAE** Introducido junto con PSE en el Pentium Pro, es el método que finalmente adoptaron los Sistemas Operativos como Linux para generar imágenes del kernel con este método de administración de memoria. Por este motivo desde entonces hasta el presente se ha “ganado” el derecho a que se lo considere un modo en si mismo.
- IA-32e** Basado en el PAE, es el modo de paginación que se utiliza cuando el procesador se setea en modo IA-32e, es decir 64 bits puro.



# Tamaño de página

## Paginación IA-32

Inicialmente el procesador 80386, adoptó un tamaño de página de 4 Kbytes. Obviamente al ser una familia compatible este tamaño es el tamaño estándar que por default usa cualquier procesador subsiguiente que active la paginación. De modo que el espacio lineal de 4 Gbytes, puede ser dividido en páginas de 4 Kbytes de manera completa (es decir  $2^{20}$  páginas).

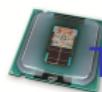
Procesadores posteriores han incluido otros tamaños de página, pero si se desea utilizar páginas de diferente tamaño que 4 Kbytes, siempre debe ser explícitamente seleccionado el otro tamaño. Si el software no hace ninguna selección especial se sigue trabajando como un 80386 en lo que a la gestión de Paginación se refiere: Páginas de 4 Kbytes dividiendo el espacio lineal de cada segmento

# Descriptores de Página

Al igual que para tratar un segmento, el procesador necesitará similar información para tratar a las páginas. Aunque hay algunas simplificaciones, a saber:

**Dirección base** Cada página comienza en la dirección de memoria siguiente a la del último byte de la página anterior. Esto significa que inician a partir de direcciones de memoria que sean múltiplos de su tamaño. Siendo  $2^n$  el tamaño de la página. Por lo tanto en páginas de 4 Kbytes ( $2^{12}$  bytes) los 12 bits menos significativos de su dirección base valen 0. **Page Frame**:conjunto de bits mas significativos de la dirección de la página que necesitamos para especificar su dirección base.

**Atributos de acceso** Inevitablemente se necesitarán algunos bits para determinar permisos y derechos de acceso.



**Tamaño** No hace falta especificarlo, ya que las páginas tienen tamaño fijo.

# ¿Cuantas páginas cubren el espacio lineal?

- Si las páginas son de 4 Kbytes ( $2^{12}$  bytes) necesitamos  $2^{20}$  páginas para cubrir los 4 Gbytes ( $2^{32}$  bytes) que significan el máximo espacio lineal que puede cubrir un segmento.
- Esta situación deriva en una tabla de descriptores de página gigantesca: En este caso,  $2^{20}$  descriptores de 4 bytes, nos llevan a una tabla que ocupa 4 MBytes de memoria física.
- Por ello conviene pasar a estructuras tablas de descriptores de paginación por niveles.
- ¿Como por niveles?... Veamos.



# Niveles de Paginación

## 1er. Nivel

**Directorio de Tablas de Páginas.**

**1024 descriptores de página = 4Kbytes**

**Cada Página corresponde a una de las 1024 del 2do. Nivel**



**4 Gbytes de Memoria dividido en Páginas de 4 Kbytes.**



# Paginas y tareas...

- La implementación en IA-32 de la paginación se ha pensado como un sistema de administración de memoria por tarea, de modo que cada tarea tiene su propia estructura de páginas.
- Esta característica robustece la seguridad del Sistema Operativo en la administración de memoria.
- Por esta razón una tabla de 4 Mbytes por cada tarea para gestionar la memoria es un contrasentido en términos de eficiencia.
- Se puede diseñar un sistema con una única tabla de páginas común a todas las tareas, pero tal enfoque da por tierra con la robustez en la de administración de memoria señalada anteriormente.



# Paginas y tareas...

- El método permite iniciar la tarea con solo dos tablas de Paginación.
  - ① El Directorio de Tablas de Página (indispensable pues es la raíz de la estructura de datos), que ocupará una página de 4 Kbytes de RAM, y contendrá tan solo un descriptor válido.
  - ② Una Tabla de Descriptores de Páginas (apuntada por la única entrada válida del Directorio), que puede almacenar hasta 1024 descriptores de páginas válidos de 4 Kbytes c/u.
- Estas 1024 páginas contendrán el código y los datos de la tarea.
- De este modo se puede iniciar un proceso con solo dos tablas de 4 Kbytes para administrar hasta 1024 páginas de 4 Kbytes, es decir 4 Mbytes para acomodar su código y datos.
- Esta cantidad es mas que suficiente para iniciar un proceso. ¿no?
- Por supuesto que cada proceso puede utilizar menos de 4 Mbytes de memoria al inicio (de hecho es lo habitual). Lo anterior es solo a efectos de cuantificar la cantidad de memoria que podemos asignarle consumiendo para su gestión solo 8 Kbytes.



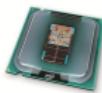
# Administración de memoria - Guidelines

## Memoria dinámicamente alojable por proceso

Lo habitual es que el Sistema Operativo habilite a cada proceso unas pocas páginas para código y datos, y a medida que el proceso requiere memoria, se irán agregando en la tabla de páginas del segundo nivel de la estructura los descriptores necesarios.

Este procedimiento permite asignar al proceso hasta 4 Mbytes de memoria.

Si se llegara a ese límite y se requiere mas memoria, se genera un segundo descriptor en el Directorio de Tablas de Página, que permitirá habilitar otros 4 Kbytes de memoria para colocar allí hasta 1024 nuevos descriptores de Página, es decir otros 4 Mbytes máximo.

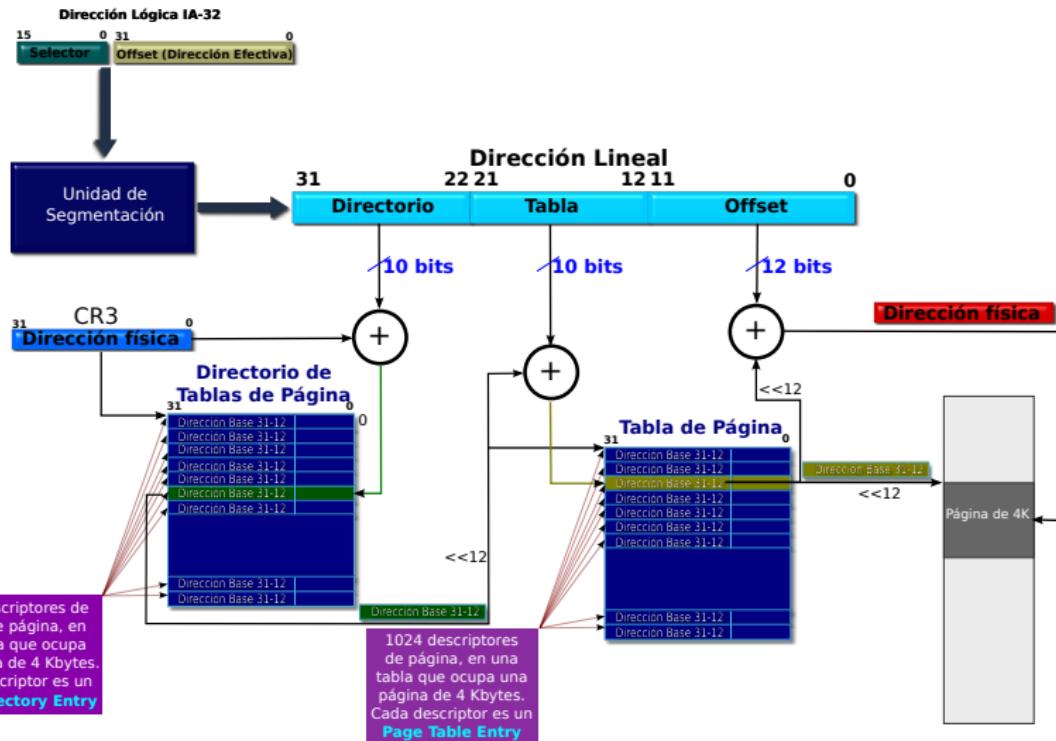


# Traducción de *dirección lineal* a *dirección física* con Páginas de 4 Kbytes en 32 bits

- Es el modo original de paginación del procesador 80386.
- El procesador necesita conocer para cada tarea, la dirección física en donde se inicia la estructura de Tablas de Páginas.
- En estos procesador el Registro de Control **CR3** contiene esta dirección (*dirección física* ).
- En el próximo slide se muestra el proceso que describiremos a continuación.
- El procesador toma la *dirección lineal* y la divide en tres campos de bits que serán utilizados respectivamente como:
  - Índice en el Directorio de Tablas de Páginas, para determinar la dirección de inicio y los atributos de la página que contendrá la tabla de Páginas uno de cuyos descriptores corresponde a la página que se está buscando acceder en memoria física.
  - Índice en la Tabla de Páginas que permitiré encontrar el descriptor de la página de memoria física que se está direccionando.
  - Desplazamiento relativo al comienzo de la página en la que se encuentra la variable o el código que se está direccionando.



# Traducción de dirección lineal a dirección física utilizando Páginas de 4 Kbytes en Modo 32 bits



# Traducción de *dirección lineal* a *dirección física* utilizando Páginas de 4 Kbytes en Modo 32 bits

- Como el procesador trabaja con Páginas de 4 Kbytes, las tablas se acomodarán en bloques del mismo tamaño.
- Al ser cada descriptor de página de 32 bits cada tabla podrá de este modo almacenar 1024 descriptores.
- Así tendremos que para ingresar a cada Tabla de Página el procesador necesitará 10 bits ( $2^{10} = 1024$ ).
- Los 10 bits mas significativos de la *dirección lineal* se usan como índice en el Directorio de Tabla de Páginas.
- El descriptor seleccionado (denominado PDE, por Page Directory Entry), contiene los 20 bits mas significativos de la dirección física de la página que contiene la Tabla de Páginas.



# Traducción de *dirección lineal* a *dirección física* utilizando Páginas de 4 Kbytes en Modo 32 bits

- Esta Tabla es la que contendrá finalmente el descriptor de la página de memoria direccionada en la MMU del procesador.
- Una vez leída la PDE y extraída la dirección física donde comienza la Tabla de Páginas, los siguientes 10 bits de la dirección lineal se utilizan como índice en ésta para acceder al descriptor de la página direccionada.
- Esta entrada se denomina PTE por Page Table Entry. En el descriptor estarán los 20 bits mas significativos de la dirección física de memoria en la que comienza la página direccionada.
- Finalmente, obtenida la dirección base de la página, sumándole los 12 bits menos significativos de la dirección lineal se obtiene la dirección física de memoria direccionada por el procesador.



# Primeras conclusiones

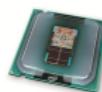
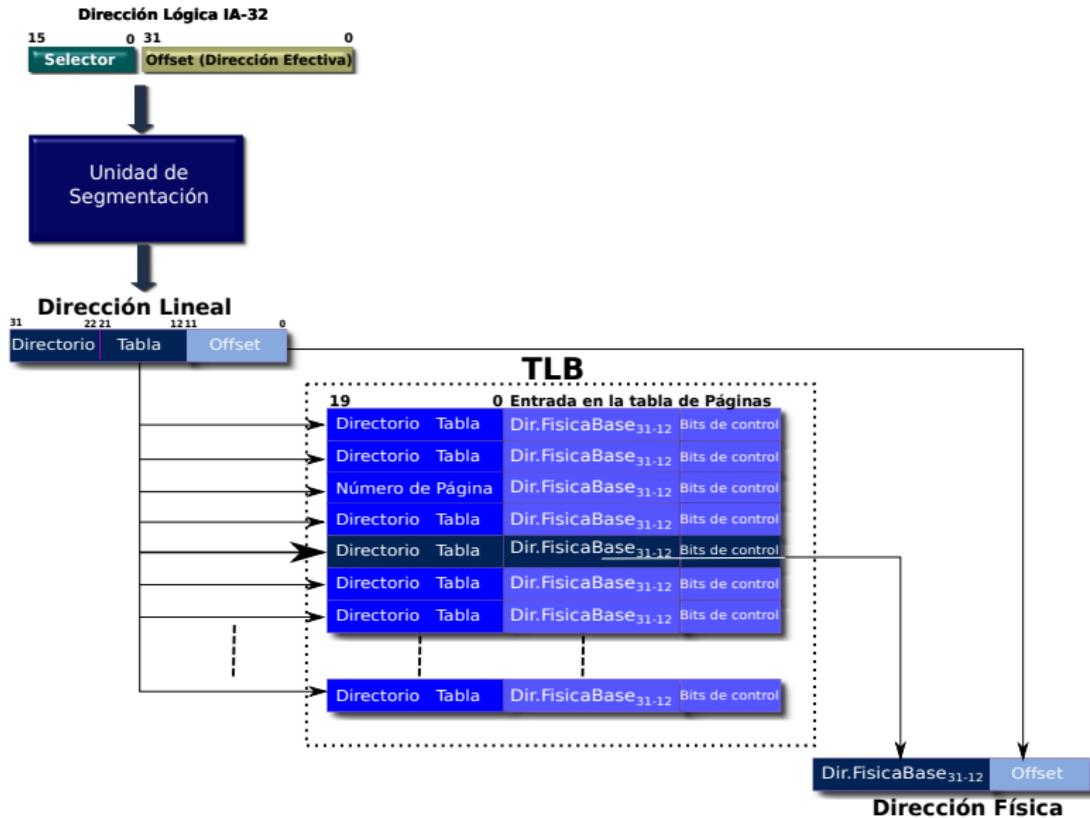
1

Del análisis de funcionamiento del mecanismo de traducción de **dirección lineal** a **dirección física** surge que no es necesario que las páginas de código estén contiguas. Si una instrucción queda al final de una página, al realizarse el próximo ciclo de fetch, el sistema de Paginación generará un par de entradas PDE - PTE diferentes de la instrucción anterior de modo que el programa ejecutará de manera transparente a la ubicación del código en memoria.

2

Por cada opcode fetch, memory read, o memory write, es necesario acceder a memoria dos veces para leer el PDE y el PTE y luego hacer sumas y demás. Al igual que en el caso de la segmentación en donde el procesador posee un cache asociado a cada registro de segmento para leer una sola vez el descriptor, en la Unidad de Paginación existe un cache de traducciones: el Translation Lookaside Buffer (o TLB)

# Translation Lookaside Buffer (TLB)



# Translation Lookaside Buffer (TLB)

- Se almacenan solo las partes de interés de las direcciones lineal y física. Mas específicamente, los bits de la **dirección lineal** que se utilizan para acceder al Directorio de Tablas de Página, y a la Tabla de Páginas, y los bits de la **dirección física** contenidos en el descriptor de la página direccionada, es decir, 31 a 12. El resto no son relevantes ya que están comprendidos en la página almacenada en la tabla y a los efectos de la traducción no actúan.
- El Buffer de Traducciones actúan como una pequeña memoria cache, almacenando los pares de valores indicados, para las últimas traducciones, ya que una vez traducida la dirección de una página, si ésta contiene código el procesador seguirá ejecutando dentro de esa página de modo que recurrirá asiduamente a esa entrada del TLB. El mismo comportamiento se verifica para variables, buffers y demás elementos de un trozo de software. Se denomina Vecindad.



# Translation Lookaside Buffer (TLB)

- Los bits de control por lo general son los atributos de la página seleccionada, mas otros que no están documentados, pero siendo un cache, imaginamos mínimamente bits LRU para determinar cual es el elemento que debe ser desalojado al necesitarse una entrada para almacenar una nueva traducción estando el TLB lleno.
- El tamaño del TLB ha ido creciendo conforme se desarrollaron nuevos procesadores. Actualmente se tiene uno para código y otro para datos ubicados en la cache L1 de código y datos del procesador.
- Por otra parte como la paginación se organiza en base a tareas, cada tarea tendrá su propia estructura con lo cual el registro **CR3** cambiará de una tarea a otra. La escritura de un valor en el registro **CR3** flushea el contenido de la TLB (Excepto aquellas entradas que, como veremos, se setean como Globales)



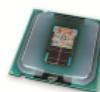
# Extensiones de Memoria Física

- Con el procesador Pentium Pro, Intel comenzó a dividir la línea de procesadores IA-32 en una para desktop Pentium y otra para servidores Pentium Pro y posteriormente Xeon.
- Este procesador introduce dos métodos para extender la capacidad de direccionamiento físico. Se trata de Physical Address Extensions (PAE).
- El siguiente modelo de servers, Pentium II Xeon introdujo una alternativa denominada Page Size Extensions (PSE-36).
- Ambos son mutuamente excluyentes y se seleccionan mediante flags del registro **CR4**.
- PAE es en sí un modo de paginación (es el que se ha impuesto entre los desarrolladores de Sistemas Operativos). No obstante vale la pena un vistazo a la traducción basada en PSE-36.



# Page Size Extensions (PSE-36)

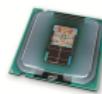
- Se activa seteando el bit **CR4.PSE**. Es el bit 4.
- Debe asegurarse de tener **CR4.PAE = 0**, ya que ambos modos de extensión son mutuamente excluyentes.
- Habilitaba en el procesador 4 terminales de address adicionales:  $A_{35-32}$ . Esto llevaba el bus a 36 líneas, permitiendo manejar 64 Gbytes de memoria física.
- La limitación de este modo es que el remanente de 4 Gbytes solo puede ser manejado por páginas de 4 Mbytes de tamaño
- Con la evolución de esta arquitectura se activaron mas terminales de Address, y debido a la amplia aceptación de PAE, puede haber procesadores que no soporten PSE-36, o que sí lo soporte pero con mas de 64 Gbytes de memoria física (es decir, mas de 36 terminales de address). Por lo tanto antes de activar PSE-36, lo recomendable es chequear previamente que el procesador soporte este modo. Esta comprobación se realiza mediante la instrucción **CPUID.01H:EDX.PSE[bit 3]=1**.



# Comprobación y activación de Page Size Extensions (PSE-36)

Este código comprueba el soporte del procesador al modo PSE-36 y su activación tomando los recaudos antes mencionados.

```
1  mov    eax,1                      ;cpuid function
2  cpuid
3  test   edx, 8                   ;bit 3 en 1?
4  jz     PSE_not_supported
5  mov    eax, cr4
6  test   eax, 0x20                ;bit PAE en '1'?
7  jnz   PAE_activo
8  or    eax, 0x10                ;bit 4 a '1'
9  mov    cr4, eax                ;PSE-36 activo
```

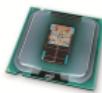


# Ampliación de memoria física mas allá de 64 Gbytes

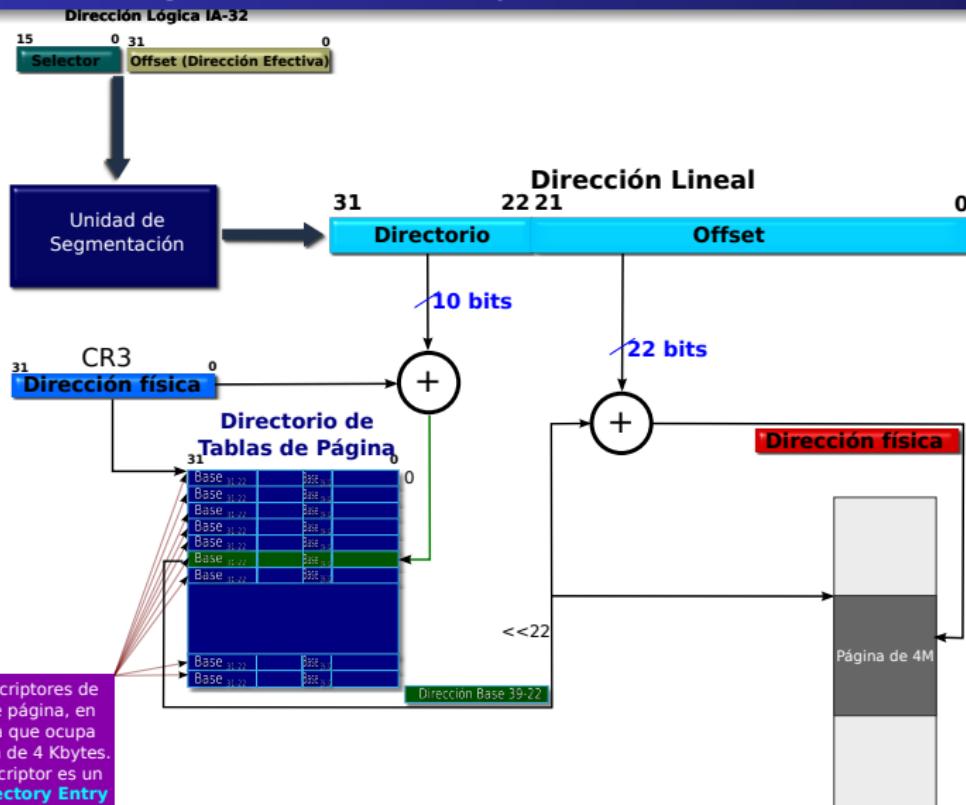
## Comprobando la cantidad de memoria física

La cantidad de memoria física cambia de modelo a modelo. Por lo tanto la instrucción CPUID.80000008H:EAX[7:0] es la manera indicada de determinar la cantidad de terminales de address que posee el procesador. Esta cantidad es el número binario contenido en el registro **AL (EAX)[7:0]**, coincide con este registro de 8 bits) a la salida de la CPUID invocada con **EAX** en 0x80000008. Intel denomina a este parámetro **MAXPHYADDR**.

Si el procesador no soporta CPUID.8000008H, son solo 36 bits de direcciones.



# Traducción de dirección lineal a dirección física utilizando Páginas de 4 Mbytes en Modo 32 bits



## Traducción de *dirección lineal* a *dirección física* utilizando Páginas de 4 Mbytes en Modo 32 bits

La figura anterior muestra el esquema de traducción solo con el Directorio de Tablas de Páginas, ya que en una página de 4 Mbytes los 22 bits menos significativos de la dirección base son '0'.

Por lo tanto los 10 bits que en el sistema de 4 Kbytes se utilizaban como índice para la Tabla de Páginas ahora no se necesitan. Se aprovecha para alojar allí los bits  $A_{39-32}$  de la dirección física.

La cantidad significativa surge de la instrucción CPUID.80000008H.

Se sigue teniendo descriptores de 32 bits, ya que los bits adicionales de address se alojan en el espacio dejado por los bits  $A_{21-12}$ , que en una página de 4 Mbytes son siempre '0'.

Cabe destacar que es posible diseñar un sistema de paginación mixto con páginas de 4 Kbytes y de 4 Mbytes. A continuación veremos como son los descriptores de página en detalle para cada caso y cuales son las limitaciones.

# CR3: Descriptor del frame de la Página que contiene el Directorio de Páginas

CR3	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	Dirección física del Directorio de Páginas	Ignorados	P C D	P W T	Ignorados
-----	---	--	-----------	-------------	-------------	-----------

- Los bits 31 a 12 del registro contienen el page frame del Directorio de Tabla de Páginas. Cabe destacar que esta dirección es física. O sea que el registro **CR3** contiene la dirección de RAM en donde comienza el Directorio de Tablas de Página.
- En modo 64 bits, este registro tiene 64 bits. En el modo 32 bits, los bits 32 a 63 se ignoran.
- Bits que permiten controlar en el momento de la traducción el tipo de memoria que se desea implementar.

**PWT** **P**age-**L**evel **W**rite **T**hrough. Establece el modo de escritura que tendrá la página en el Cache.



**PCD** **P**age-**L**evel **C**ache **D**isable. Establece que una página integre el tipo de memoria no cacheable.

# Descriptores en el Directorio de Tablas de Páginas

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dirección física de la Tabla de Página															Ignorados		P S	I G N	A D	P C D	P W T	U S	R /	W P							

Figura: Descriptor del Page Frame una Tabla de Páginas de 4 Kbytes

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bits 31 a 22 de la Dirección física del frame de la Página de 4M										Reservados deben ser 0		Bits 36 a 32 de la Dirección			P A T	Ignorados		G S	P D	D A	P C D	P W T	U S	R /	W P						

Figura: Descriptor del Page Frame una Página de 4 Mbytes

- Ambos tipos de descriptores pueden coexistir en la misma tabla.
- Los bits 31 a 12 del descriptor de una página de 4 Kbytes contienen el valor correspondiente a los mismos bits de la dirección física de la Página (el Page Frame) que contiene la tabla de descriptores de página, uno de cuyos 1024 descriptores corresponderá a la página que contiene la dirección de memoria direccionada por el procesador.



# Descriptores en el Directorio de Tablas de Páginas

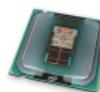
- Esto indica que en el modo de paginación de 32 bits, las páginas de 4 Kbytes mapean trozos de memoria física correspondientes a los primeros 4 Gbytes del espacio físico de direccionamiento. Nunca mas allá.
- El descriptor de página de 4 Mbytes contiene los 4 bits adicionales en el espacio del descriptor que libera el hecho de manejar un Page Frame de 4 Mbytes. A continuación del campo de los bits adicionales se ve un grupo de bits reservados (bits 17 a 21). Estos bits deben ser 0. Si el procesador solo tiene 36 líneas de address (Pentium Pro y Pentium II Xeon), el Page Frame se completa con los bits 13 a 16 del descriptor, que contienen sus bits de peso 32 a 35.
- Para procesadores posteriores es posible tener mas de 36 bits de address los cuales irán consumiendo los bits que figuran como reservados, a partir del 17 y hasta donde indique el parámetro **MAXPHYADDR**.



# Formato del descriptor de una Página de 4 Kbytes

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dirección física del frame de la Página de 4 Kbytes	Ignorados	G	P A T	D	A	C D	P W T	P U S	R /	W	P																				

- 1024 de estos descriptores se almacenan en una página de memoria cuyo descriptor corresponde al del Directorio de Tabla de Páginas.
- Es el último nivel de traducción.
- Los bits 31 a 12, contienen el Page Frame de la página de memoria, es decir los 20 bits mas significativos de la dirección física de memoria en donde comienza la página (los 12 menos significativos son '0').



# Atributos de los descriptores

Los atributos son los mismos para todos los descriptores vistos y tienen el mismo significado.

**PS** **Page Size:** Existe solo en el Directorio de Tablas de Página. Si es '0' la entrada es de una Tabla de Páginas de 4 Kbytes, y si es '1' la entrada describe una página de 4 Mbytes.

**PAT** **Page Attribute Table:** Los MSRs **MTRR** definen y controlan el cache de diferentes zonas de memoria. A partir del Procesador Pentium III se dispone de este atributo que los complementa en base a páginas.

**G** **Global:** Si **CR4.PGE = 1** Tiene efecto la activación de la funcionalidad Global, que otorga ese carácter a la traducción de esa página almacenada en la TLB. La entrada no se flushea cuando se recarga el registro **CR3**.



# Atributos de los descriptores

- D **Dirty.** Indica que la página ha sido modificada (está “sucia“). El Sistema Operativo lo inicializa en ‘0’, y se setea en forma automática cuando se escribe la página. El algoritmo de swap en el momento de desalojar una página de la memoria RAM, analiza este bit y no la copia de memoria a disco si no ha sido modificada, mejorando de este modo su eficiencia. Solo tiene sentido en descriptores de Página (de 4K o 4M). En las entradas del Directorio de Tablas de Páginas que describen Tablas de Páginas de 4 Kbytes se ignora ya que el Bit D existirá en el siguiente descriptor de la estructura.
- A **Accedido.** Cada vez que la página es accedida este bit se setea. El Sistema Operativo puede contabilizar los accesos de modo de elaborar estadísticas de uso que permitan identificar cual es la página candidata a ser desalojada llegado el momento.



# Atributos de los descriptores

- U/S** **User / Supervisor:** Privilegio de la Página: '0' Supervisor (Kernel), y '1' Usuario. En general corresponden U/S = 0 a **DPL** = 00, y U/S = 1 al resto de los valores de **DPL**. El procesador chequea el **CPL** del segmento de código para autorizar o no el acceso a las páginas de acuerdo a la combinación de los valores de U/S de los diferentes descriptores de su estructura.
- R/W** **Readable / Writable:** Establece si la página es Read Only (0) o si puede ser escrita (1).
- P** **Presente:** Indica si la página está en la memoria ( $P=1$ ), generando una excepción #PF cuando se intenta acceder a una dirección de memoria que tiene al menos un descriptor con  $P=0$  a lo largo de la estructura de tablas. Cuando  $P = 0$ , el resto del contenido del descriptor se ignora.



**PCD y PWT** Idem registro **CR3**.

# Características del Modo PAE

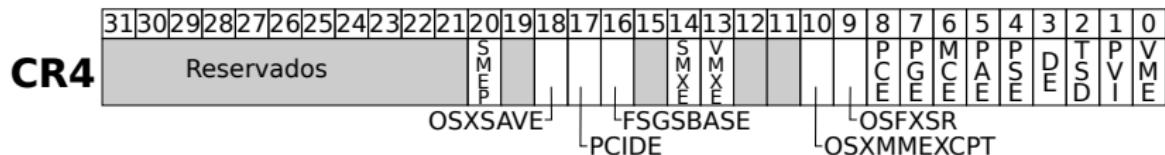
- Introducido como hemos dicho por el procesador Pentium Pro.
- Permite trabajar con páginas de 4 Kbytes o de 2 Mbytes, pudiendo particionar con cualquiera de los tamaños de páginas o incluso combinando ambos tamaños, el espacio físico completo.
- PAE evolucionó con la arquitectura y en la actualidad permite trabajar hasta con 52 bits de memoria física (hasta 4 Pbyte de RAM).
- Recordemos que la cantidad de bits de memoria Física se obtiene como resultado de CPUID.0x80000008, en el registro **AL**, y se denomina **MAXPHYADDR**.
- Si **MAXPHYADDR** < 52, los bits 51 y consecutivos en orden descendente, hasta el que corresponda al peso **MAXPHYADDR**, deben estar en '0'.



Es la base del modo de paginación IA-32e.

# Activación del Modo PAE

PAE se activa haciendo **CR4.PAE = 1**. Hemos nombrado varios bits del registro **CR4**. A continuación el layout de este registro.



Una vez activado, PAE permite, traducir una **dirección lineal** de 32 bits en una **dirección física** de 52 bits.  
 Observar que de todos modos accedemos a un espacio lineal de 4 Gbytes tope en cada momento.



# Estructura de traducción del Modo PAE

**CR3**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dirección física de la Tabla de punteros a Directorio de Páginas																										Ignorados					

- En PAE el procesador arma una estructuras jerárquicas tres niveles.
- La base es siempre **CR3** (que como vemos difiere del layout para el modo 32 bits)
- El primer nivel es un array de cuatro punteros a Directorios de Tablas de Páginas. El registro **CR3** contendrá la dirección base de esta tabla de 4 punteros, **la cual debe estar alineada a 32 bytes**. Esta condición debe ser respetada **o el procesador generará una excepción**.



# Estructura de traducción del Modo PAE

## Tabla de Punteros de Directorio de Página

- Al activarse el modo PAE, **CR3** apunta a una tabla de cuatro entradas denominada Tabla de Punteros de Directorio de Página. Cada entrada de esta Tabla se denomina PDPTE (por Page Directory Pointer Table Entry) y controla el acceso a un área de memoria lineal de 1 Gbyte.
- El procesador mantiene un set de 4 registros internos (no arquitecturales): **PDPTE0**, **PDPTE1**, **PDPTE2**, y **PDPTE3**.
- Estos registros tienen por misión mantener el puntero a su respectivo Directorio de Tabla de Páginas asociado, con el objetivo de mejorar y agilizar la traducción de direcciones lineales a físicas, y se recargan desde sus respectivas copias en la memoria RAM del sistema cada vez que se efectúan operaciones que pueden alterar sus valores. Como el sistema trabaja con la versión de memoria, el procesador debe releerlas para tener actualizada esta suerte de cache interna.

# Estructura de traducción del Modo PAE

En general se recargan estos registros cuando:

- Se ejecuta una instrucción cuyo operando destino es el registro **CR0**, o el registro **CR4** y como resultado de la cual se modifica uno o mas de los bits:

**CR0.CD** Cache Disable. Con '1' el cache del procesador se deshabilita.

**CR0.NW** Non\_Write Through. Establece globalmente esta política de escritura, siempre que **CR0.CD** = 0. De otro modo no tiene sentido.

**CR0.PG** Activa Paginación.

**CR4.PAE** Activa el modo PAE.

**CR4.PGE** Permite definir el uso de Páginas globales que se habilitan una a una con el bit **PG** de DPTE.

**CR4.PSE** Habilita o deshabilita el modo PSE-36

**CR4.SMEP** Habilita un modo de protección denominado **Supervisor Mode Execution Prevention**.



# Estructura de traducción del Modo PAE

- Cuando se carga un valor en el registro **CR3**, ya que al cambiar éste registro debe buscarse de memoria el nuevo set de cuatro punteros que gestionará el espacio lineal de 4 Gbytes.
- Si como consecuencia de un task switch en modo IA-32 cambia el valor de **CR3**. Tal como veremos al abordar el manejo de Tareas, el registro **CR3** forma parte del contexto de ejecución que el procesador salvará y recuperará cuando llegue el momento. Por tal motivo un task switch automático por parte de los recursos del procesador el registro **CR3** se modifica debiendo re leer los cuatro punteros desde memoria.
- Algunas transacciones VMX motivarán también la carga de estos registros por parte del procesador.



# PAE: Traducción de dirección lineal a dirección física

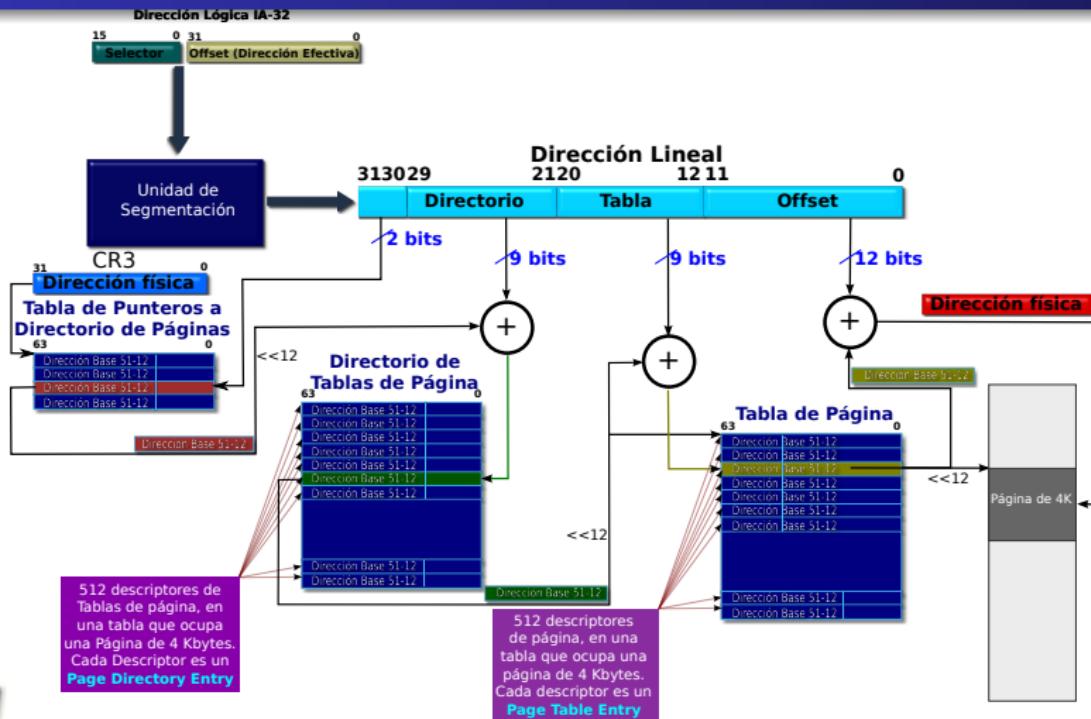


Figura: Mecanismo de traducción de 3 niveles con PAE activo y páginas de 4K

# PAE: Traducción de dirección lineal a dirección física

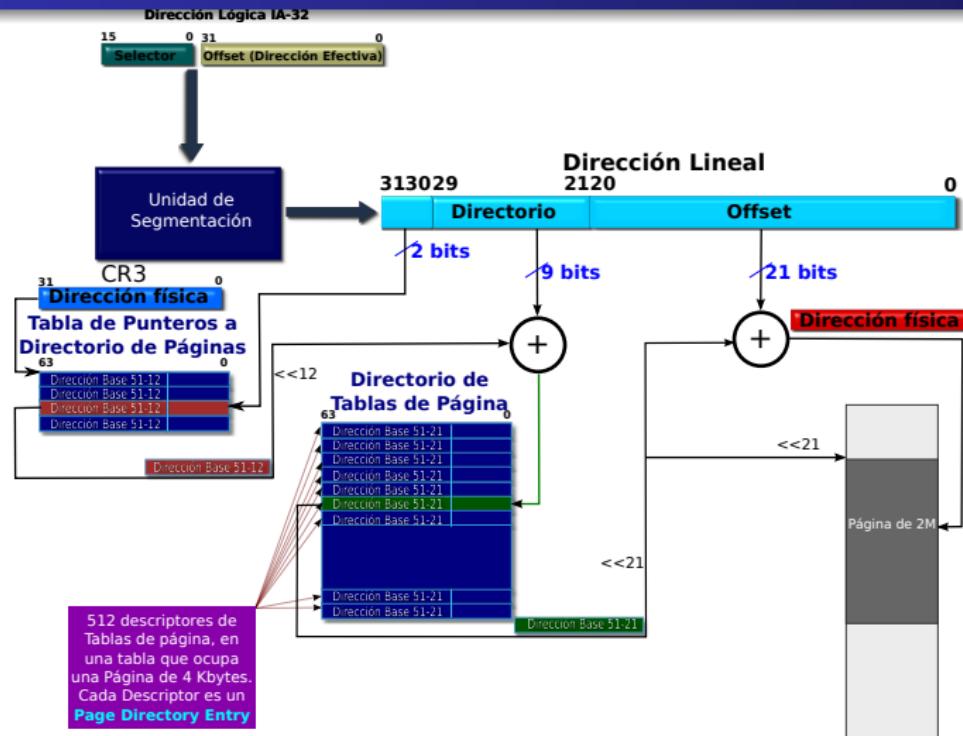


Figura: Mecanismo de traducción de 2 niveles con PAE activo y páginas de 2M

# PAE highlights

- Mapear páginas de 4 Kbytes o 2 Mbytes en cualquier parte de la dirección físicamente y soportar la cantidad de memoria física que fuese necesaria sin tener restricciones.
- Al tener 64 bits, los descriptores de página ocupan en esta jerarquía el doble de tamaño. Por lo tanto en cada página física de 4 Kbytes caben ahora 512 descriptores, es decir la mitad respecto del Modo de Paginación de 32 bits.
- Como de todos modos, se utilizan páginas de 4 Kbytes para cada tabla de la estructura de paginación PAE, significa que cada tabla tiene ahora 512 descriptores y por lo tanto se requieren 9 bits para indexar al descriptor buscado.



# Formato de los Descriptores

63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32	
Reservados	Dirección física Page Directory de 4 Kbytes <sup>1</sup>

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	Dirección física Page Directory de 4 Kbytes <sup>1</sup>	Ignorados	Reservados (deben ser 0)	P C D	P W T	Resrv deben ser 0	P
---	--	-----------	-----------------------------	-------------	-------------	-------------------------	---

<sup>1</sup> Los Bits 51 a M (con M=MAXPHYADDR) deben ser '0' si M < 52. MAXPHYADDR se obtiene, en el registro AL como resultado de la instrucción CPUID.0x80000008.

**Figura:** Formato de una entrada de la Tabla de Punteros a Directorio de Páginas (PDPTE)



# Formato de los Descriptores

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
X D	Reservados																						Dirección física del frame del DPT de 4 Kbytes <sup>1</sup>								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dirección física de la Tabla del DPT de 4 Kbytes <sup>1</sup>																							Ignorados	P S	I G N <sup>2</sup>	A C D	P W T	P U S	R /W	P	

<sup>1</sup> Los Bits 51 a M (con M=MAXPHYADDR) deben ser '0' si M < 52. MAXPHYADDR se obtiene, en el registro AL como resultado de la instrucción CPUID.0x80000008.

<sup>2</sup> En el Bit 6 **IGN** significa ignorado.

**Figura:** Formato de una entrada del Directorio de Páginas (DPTE), que describe una Tabla de Páginas de 4 Kbytes



# Formato de los Descriptores

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	
X D	Reservados																								Dirección física del frame de la Página de 2 Mbytes <sup>1</sup>							

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dirección física del frame de Página de 2 Mbytes <sup>1</sup>	Reservados (deben ser 0)												P A T	Ignorados	G	P S	D	A	P C D	P W T	P U S	R / W	P								

<sup>1</sup> Los Bits 51 a M (con M=MAXPHYADDR) deben ser '0' si M < 51. MAXPHYADDR se obtiene, en el registro AL como resultado de la instrucción CPUID.0x80000008.

**Figura:** Formato de una entrada del Directorio de Páginas (DPTE), que describe una Página de 2 Mbytes



# Formato de los Descriptores

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
X	D	Reservados																								Dirección física del frame de la Página de 4 Kbytes <sup>1</sup>					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dirección física del frame de la Página de 4 Kbytes																									Ignorados	G	P A T	P C D	P W T	U S R /W P	

<sup>1</sup> Los Bits 51 a M (con M=MAXPHYADDR) deben ser '0' si M < 51. MAXPHYADDR se obtiene, en el registro AL como resultado de la instrucción CPUID.0x80000008.

**Figura:** Formato de una entrada del Tabla de Páginas (PTE), que describe una Página de 4 Kbytes



# En 64 bits, la paginación lo es casi todo...

- Este modo se activa únicamente al ingresar el procesador al modo de trabajo IA-32e sub modo 64 bits.
- De acuerdo con lo visto anteriormente hay tres modos de Paginación posibles: IA-32, PAE, IA-32e.
- Los tres modos dependen del estado de tres bits: ***CR0.PG***, ***CR4.PAE***, y ***IA32\_EFER.LME***: Este último es el que completa el establecimiento del modo de paginación IA-32e.
- Para cada uno de los modos los tres bits deben tener estas combinaciones
  - ① **Paginación de 32 bits**.Para ingresar a este modo se deben establecer los siguientes valores: ***CR0.PG*** = 1, ***CR4.PAE*** = 0, y ***IA32\_EFER.LME*** = 0.
  - ② **Paginación PAE**.Para ingresar a este modo se deben establecer los siguientes valores: ***CR0.PG*** = 1, ***CR4.PAE*** = 1, y ***IA32\_EFER.LME*** = 0.
  - ③ **Paginación de 64 bits**.Para ingresar a este modo se deben establecer los siguientes valores: ***CR0.PG*** = 1, ***CR4.PAE*** = 1, y ***IA32\_EFER.LME*** = 1.



# Traducción en 64 bits

- Cuando se ingresa en el modo IA-32e el procesador genera direcciones lineales de 64 bits, partiendo del modelo de segmentación FLAT muy devaluado en términos de chequeos respecto del modo IA-32 legacy.
- Sin embargo para el sistema de traducción no es necesario trabajar con los 64 bits. En general los procesadores que soportan el modo IA-32e hasta el momento utilizan solo los 48 menos significativos. Con esto se tienen 256 Tbytes (TeraBytes) de espacio lineal de direccionamiento.
- Para prevenir futuras ampliaciones Intel introdujo entre las funciones largas de la instrucción CPUID, una que informa la cantidad de bits que componen una **dirección lineal**, que en la documentación de CPUID aparece como **dirección virtual**, y la cantidad de bits que componen una **dirección física**, que no es mas que el parámetro **MAXPHYADDR** ya mencionado.



## Comprobación de los bit de *dirección lineal* y *dirección física*

```
1 ; /////////////////////////////////
2 ; Llamar a CPUID con EAX = 0x80000008
3 ;
4 mov eax,0x80000008
5 cpuid
6 ; En este punto se tiene
7 ; EAX 15-8 = Cantidad de bits soportados que conforman la
     dirección virtual
8 ; EAX 7-0 = Cantidad de bits soportados que conforman la
     dirección física
```



# Estructuras de Paginación en 64 bits

## Raiz de la estructura

Partimos del registro **CR3**. En el modo 64 bits su contenido depende de la habilitación o no del bit **CR4.PCIDE**. Este bit se usa solo en el modo 64 bits y permite habilitar una funcionalidad en el sistema cache que se denomina **Process Context IDentifier (PCID)**, y permite a un procesador lógico cachear información desde múltiples direcciones lineales. Así el procesador lógico puede retener información en el cache aun cuando el software commute a otro espacio lineal de direccionamiento, por ejemplo cambiando el valor de **CR3**.

PCID es soportada por procesadores que responden a la función 0x01 de CPUID devolviendo el bit 17 del registro **ECX** seteado. De otro modo no se encuentra disponible. Una vez detectada su disponibilidad se habilita seteando el bit **CR4.PCIDE**, y ahora cada vez que el procesador genera una entrada en la estructura de la memoria cache y en la TLB, las asocia con el valor de PCID que figura en el campo homónimo del registro **CR3** según se puede ver a continuación.

# Formato de CR3 en modo IA-32e

**CR3**

63								$M_1^+$	M										39	38	37	36	35	34	33	32						
Reservados (deben ser 0)													Dirección física alineada a 4 Kbytes de la PML4																			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Dirección física alineada a 4 Kbytes de la PML4													Ignorados													P	P	C	W	D	T	Ignor

Figura: Layout del registro CR3 si CR4.PCIDE = 0

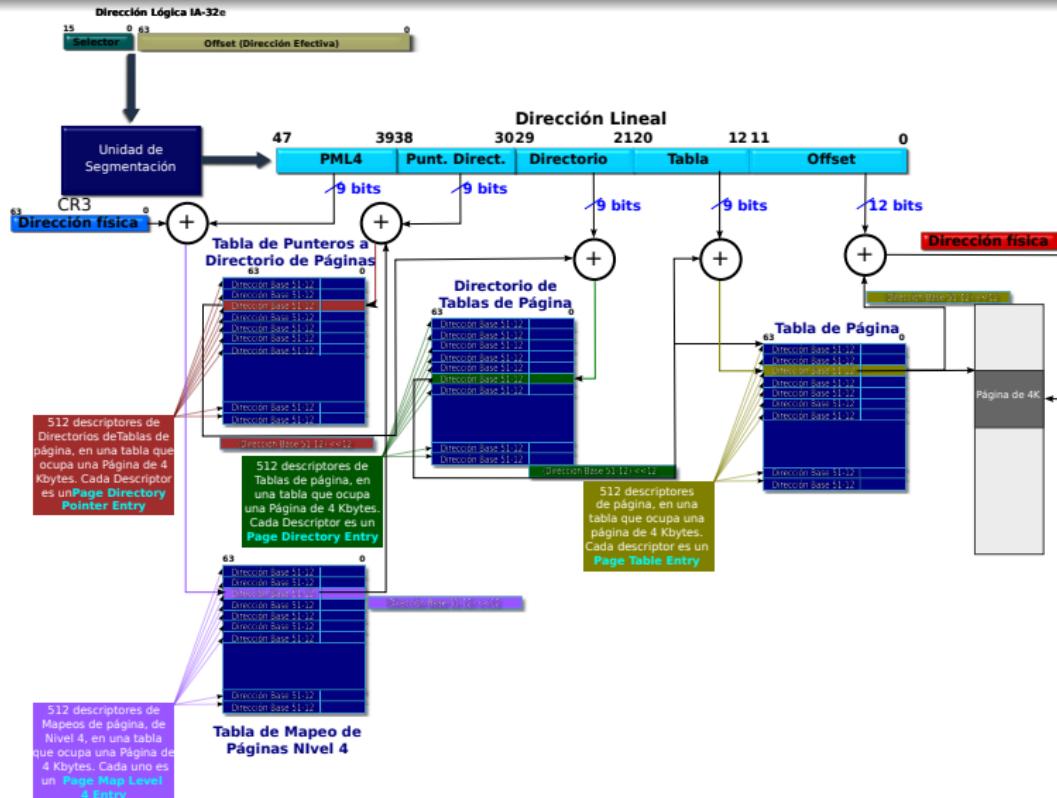
**CR3**

63								$M_1^+$	M										39	38	37	36	35	34	33	32					
Reservados (deben ser 0)													Dirección física alineada a 4 Kbytes de la PML4																		
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dirección física alineada a 4 Kbytes de la PML4													PCID																		

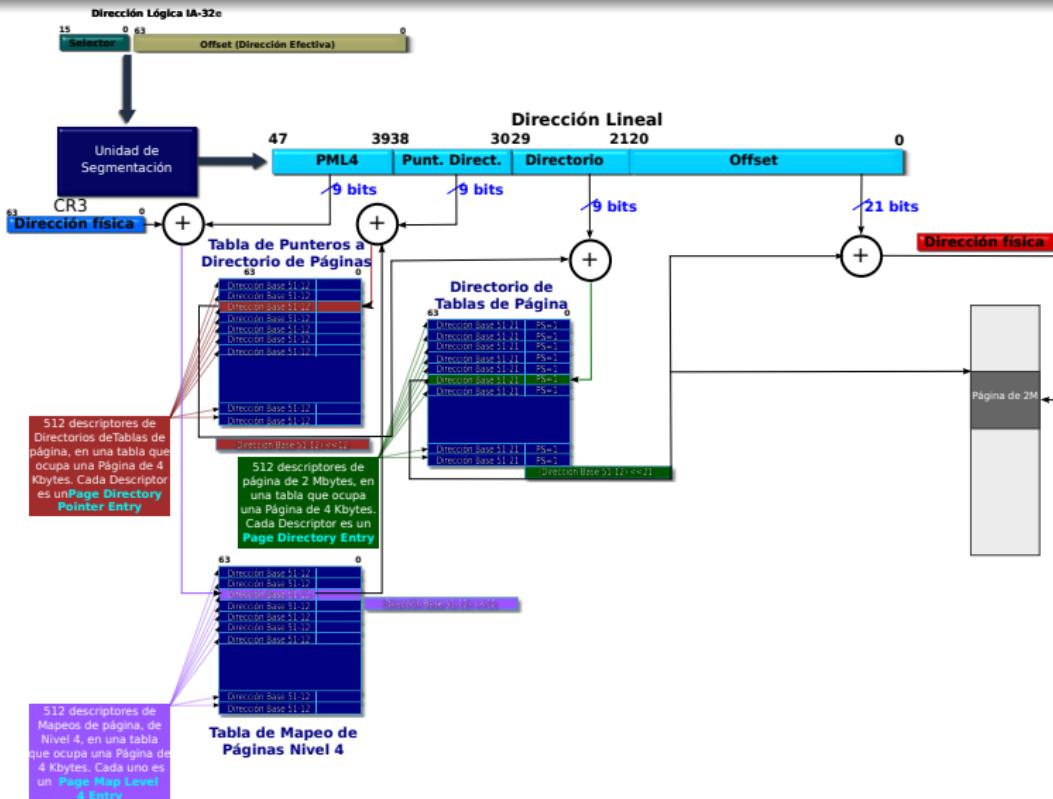
Figura: Layout del registro CR3 si CR4.PCIDE = 0



# Mecanismo de traducción de 4 niveles y páginas de 4 Kbytes en IA-32e

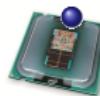


# Mecanismo de traducción de 3 niveles y páginas de 2 Mbytes en IA-32e

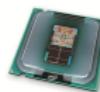
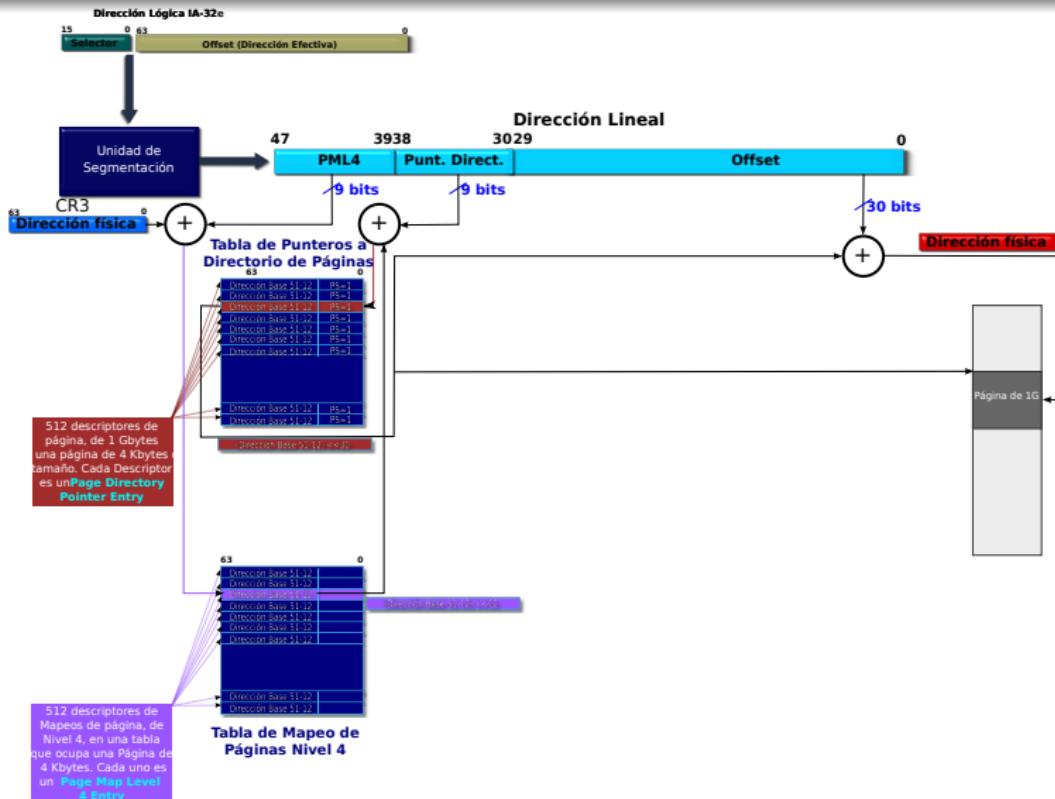


# Páginas de 1 Gbyte de tamaño

- El modo IA-32e tiene la posibilidad de trabajar con páginas de tamaño 1 Gbyte, superando funcionalmente al PAE.
- Para ello emplea una estructura jerárquica de dos niveles, eliminándose el nivel de Directorio de Tablas de Página. Los 30 bits menos significativos de la dirección lineal son entonces el offset dentro de la página.
- Se tiene una primera tabla de descriptores (PML4) que permite direccionar hasta 512 tablas de Punteros a Directorios de Página, cada una de cuyas entradas contiene un descriptor de página cuyo atributo PS=1, y direcciona directamente una página alineada a 1 Gbyte y con este mismo tamaño.
- Tener 250 Mega Páginas de 1 Gbyte, nos lleva a un espacio físico de 256 Peta Bytes subdividido en páginas de 1 Gbytes.
- Para activar páginas de este tamaño es necesario antes chequear si el procesador lo soporta. La función 0x80000001 de CPUID, devuelva el bit 26 de EDX seteado en tal caso.



# Mecanismo de traducción de 2 niveles y páginas de 1 Gbytes en IA-32e



# Formato de retorno de la Instrucción CPUID Función 0x80000001

Bit	Name	Description when Flag = 1	Comments
31:30		<b>Reserved</b>	Do not count on the value.
29	Intel® 64	Intel® 64 Instruction Set Architecture	The processor supports Intel® 64 Architecture extensions to the IA-32 Architecture. For additional information refer to <a href="http://developer.intel.com/technology/architecture-silicon/intel64/index.htm">http://developer.intel.com/technology/architecture-silicon/intel64/index.htm</a>
28		<b>Reserved</b>	Do not count on the value.
27	RDTSCP	RDTSCP and IA32_TSC_AUX	The processor supports RDTSCP and IA32_TSC_AUX.
26	1 GB Pages	1 GB Pages	The processor supports 1-GB pages.
25:21		<b>Reserved</b>	Do not count on the value.
20	XD Bit	Execution Disable Bit	The processor supports the XD Bit when PAE mode paging is enabled.
19:12		<b>Reserved</b>	Do not count on the value.
11	SYSCALL	SYSCALL/SYSRET	The processor supports the SYSCALL and SYSRET instructions.
10:0		<b>Reserved</b>	Do not count on the value.



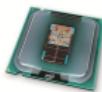
# Comparación entre los modos de paginación

La estructura General de Paginación varía de acuerdo al Modo de Paginación empleado. La Tabla de la figura, muestra las características de los diferentes modos de paginación del procesador organizada por cada nivel de paginación de la estructura.

Paging Structure	Entry Name	Paging Mode	Physical Address of Structure	Bits Selecting Entry	Page Mapping
PML4 table	PML4E	32-bit, PAE	N/A		
		IA-32e	CR3	47:39	N/A (PS must be 0)
Page-directory-pointer table	PDPT	32-bit	N/A		
		PAE	CR3	31:30	N/A (PS must be 0)
		IA-32e	PML4E	38:30	1-GByte page if PS=1 <sup>1</sup>
Page directory	PDE	32-bit	CR3	31:22	4-MByte page if PS=1 <sup>2</sup>
		PAE, IA-32e	PDPT	29:21	2-MByte page if PS=1
Page table	PTE	32-bit	PDE	21:12	4-KByte page
		PAE, IA-32e		20:12	4-KByte page

## NOTES:

- Not all processors allow the PS flag to be 1 in PDPTEs; see Section 4.1.4 for how to determine whether 1-GByte pages are supported.



# Resumen de características de cada Modo de Paginación

Paging Mode	PG in CR0	PAE in CR4	LME in IA32_EFER	Lin.-Addr. Width	Phys.-Addr. Width <sup>1</sup>	Page Sizes	Supports Execute-Disable?	Supports PCIDs?
None	0	N/A	N/A	32	32	N/A	No	No
32-bit	1	0	0 <sup>2</sup>	32	Up to 40 <sup>3</sup>	4 KB 4 MB <sup>4</sup>	No	No
PAE	1	1	0	32	Up to 52	4 KB 2 MB	Yes <sup>5</sup>	No
IA-32e	1	1	2	48	Up to 52	4 KB 2 MB 1 GB <sup>6</sup>	Yes <sup>5</sup>	Yes <sup>7</sup>

## NOTES:

1. The physical-address width is always bounded by MAXPHYADDR; see Section 4.1.4.
2. The processor ensures that IA32\_EFER.LME must be 0 if CR0.PG = 1 and CR4.PAE = 0.
3. 32-bit paging supports physical-address widths of more than 32 bits only for 4-MByte pages and only if the PSE-36 mechanism is supported; see Section 4.1.4 and Section 4.3.
4. 4-MByte pages are used with 32-bit paging only if CR4.PSE = 1; see Section 4.3.
5. Execute-disable access rights are applied only if IA32\_EFER.NXE = 1; see Section 4.6.
6. Not all processors that support IA-32e paging support 1-GByte pages; see Section 4.1.4.
7. PCIDs are used only if CR4.PCIDE = 1; see Section 4.10.1.



# Generalidades

- Linux es un sistema operativo multiplataforma, lo cual supone compila para diferentes microprocesadores.
- Esto crea a obligación en los desarrolladores de código de emplear estructuras de programación flexibles y portables.
- No basta para ello simplemente escribir código en C, sino hacerlo para que este resulte suficiente para cumplir estos objetivos.
- Implementar un código suficientemente flexible para un procesador x86 en lo que a paginación se refiere por si solo se hace un interesante problema, que implica detenerse a pensar de que forma definir una única estructura que sirva para implementar Paginación de 32, bits, PAE, o IA-32e.
- En general todos los procesadores tienen una estructura de paginación que básicamente se encarga de traducir una **dirección virtual** (en os procesadores x86 llamada **dirección lineal**), en una **dirección física**, apta para enviar al exterior por el bus de direcciones.



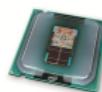
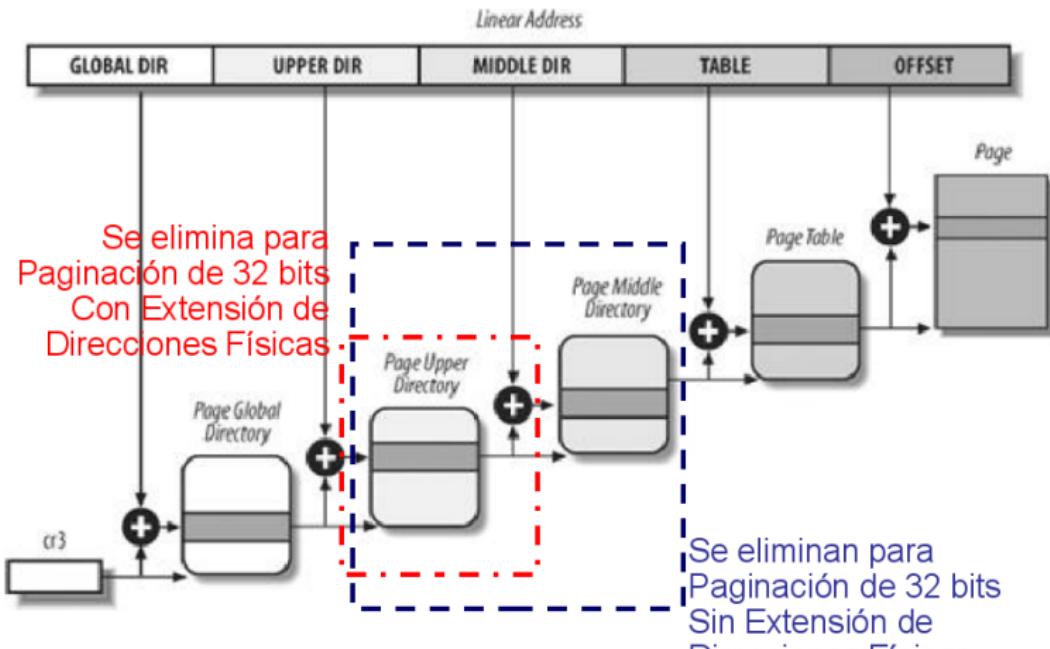
# Generalidades

- A diferencia de la administración de memoria por segmentación, que es prácticamente una marca registrada de los Procesadores x86, la administración de memoria por paginación es común a todos los procesadores que implementan multitarea y lidian con algún mínimo esquema de protección de memoria entre tareas.
- Hemos analizado la implementación de la paginación en los procesadores x86, en sus diferentes modos.
- Esto es lo que hace atractivo el estudio de esta familia de procesadores: No hay ninguna más compleja, y prácticamente en si misma presenta variantes para las que se requiere reunir otras varias arquitecturas diferentes.
- Habrá diferencias en la cantidad de bits de la **dirección virtual**, el tamaño de página, y otros detalles de implementación, en el mecanismo de traducción de cada arquitectura.



Para implementar una estructura suficientemente flexible de programación, Linux se basa en un modelo genérico.

# Estructura de Paginación General de Linux



# Parámetros de Paginación para diferentes procesadores en Linux 32-bit

Arquitectura	PageSize	BitAddr	Niveles	Bits de cada campo
alpha	8 Kbytes	43	3	10 + 10 + 10 + 13
IA-64 (Itanium)	4 Kbytes	39	3	9 + 9 + 9 + 12
ppc64	4 Kbytes	41	3	10 + 10 + 9 + 12
sh64	4 Kbytes	41	3	10 + 10 + 9 + 12
Intel® 64	4 Kbytes	48	4	9 + 9 + 9 + 9 + 12

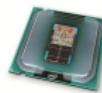
Así es que para cada arquitectura un conjunto de macros permiten definir la cantidad de bits de cada acampo de la **dirección virtual** particionada en 4 niveles en la Figura anterior, pudiendo colapsar uno o hasta dos campos intermedios simplemente poniendo 0 como su valor a la marco que define la cantidad de bits de ancho de ese campo.



# Lineamientos para Administración de la memoria

Se trata de optimizar el uso de la memoria asignando a cada proceso y al mismo kernel solo la memoria necesaria, y liberándola ni bien haya finalizado su uso.

- ① El kernel se asigna de modo permanente un área de memoria para su código y sus estructuras de datos estáticas
- ② El resto se denomina *memoria dinámica*, y consiste en un recurso fundamental, tanto para los procesos como para el mismo kernel.
- ③ Como consecuencia de los dos ítems anteriores, la correcta administración de la *memoria dinámica* impacta directamente en la performance del sistema.



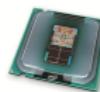
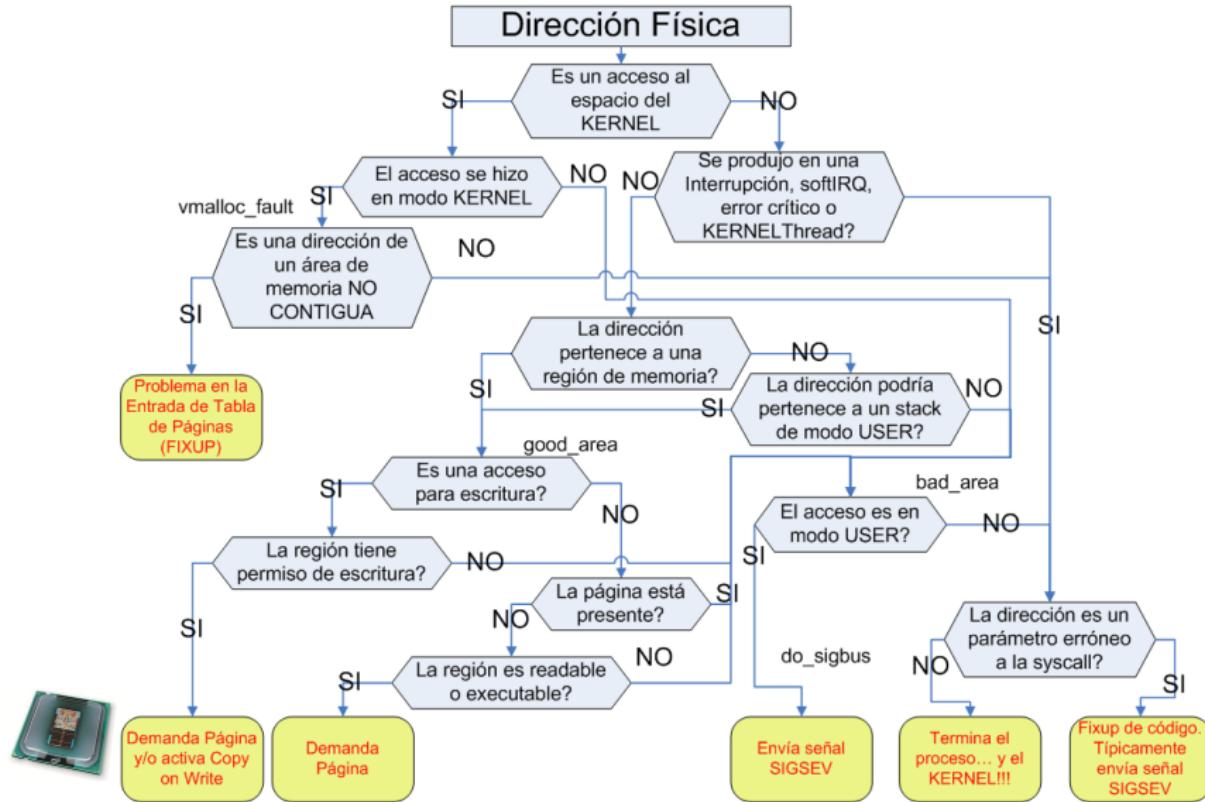
# Lineamientos para Administración de la memoria

Las distribuciones de Linux de 32 bits trabajan con páginas de 4 Kbytes solamente (aún las que implementan PAE). Pesaron en la decisión los siguientes factores que se señalan como cruciales.

- El algoritmo de swaping es mas simple
- Si bien las transferencias de disco se hacen en múltiplos enteros de los tamaños de páginas de 4 Kbytes, lo cual permitiría emplear tamaños de páginas grandes, las transferencias de disco a memoria de menor tamaño son mas eficientes.



# Detalle del Page Fault handler



Demandar Página y/o activa Copy on Write

Demandar Página

Envía señal SIGSEV

Termina el proceso... y el KERNEL!!!

Fixup de código.  
Tipicamente envía señal SIGSEV