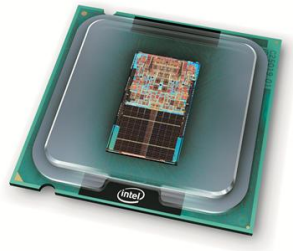


Organización del Computador II

Procesadores IA-32 e Intel® 64 - Protección

Alejandro Furfaro

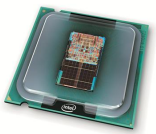
Departamento de Computación - FCEyN - UBA

4 de julio de 2019

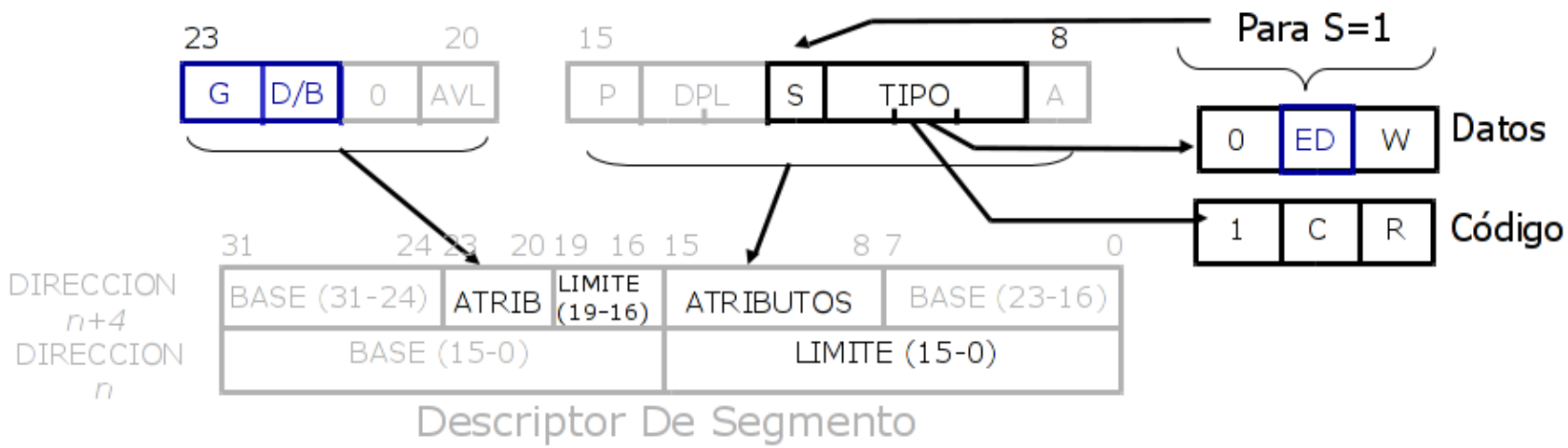
Temario

¿Que pasa cuando seteamos PE=1?

- Cuando el procesador pasa a Modo Protegido se pone en funcionamiento la Unidad de Protección.
- Esta Unidad supervisa las operaciones internas del procesador para la decodificación y ejecución de las instrucciones, comprobando el cumplimiento de una serie de reglas que constituyen el entorno de protección necesario para implementar de manera eficiente la multitarea.
- Estas reglas cubren los siguientes aspectos
  - Chequeo del Límite de los segmentos.
  - Chequeo del Tipo de los segmentos.
  - Chequeo de los niveles de privilegio de segmentos y páginas.
  - Restricción del dominio de direccionamiento a las tareas.
  - Restricción de los puntos de entrada a los procedimientos.
  - Restricción en el uso del set de instrucciones.



Elementos que el procesador toma en cuenta



Se apoya en algunos atributos:

- ***Bit G***: Determina si el valor de límite se mide en Bytes o en páginas de 4 Kbytes. En el primer caso el rango es 0 a 0xFFFFF, en el segundo es de 0xFFF a 0xFFFFFFFF.
- ***Bit D/B***: Determina si el segmento es de 16 o 32 bits
- ***Bit ED***: Expand Down. Indica el sentido de crecimiento del segmento. Importante en el manejo de pilas

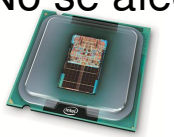
Chequeo de Límite si ED=0

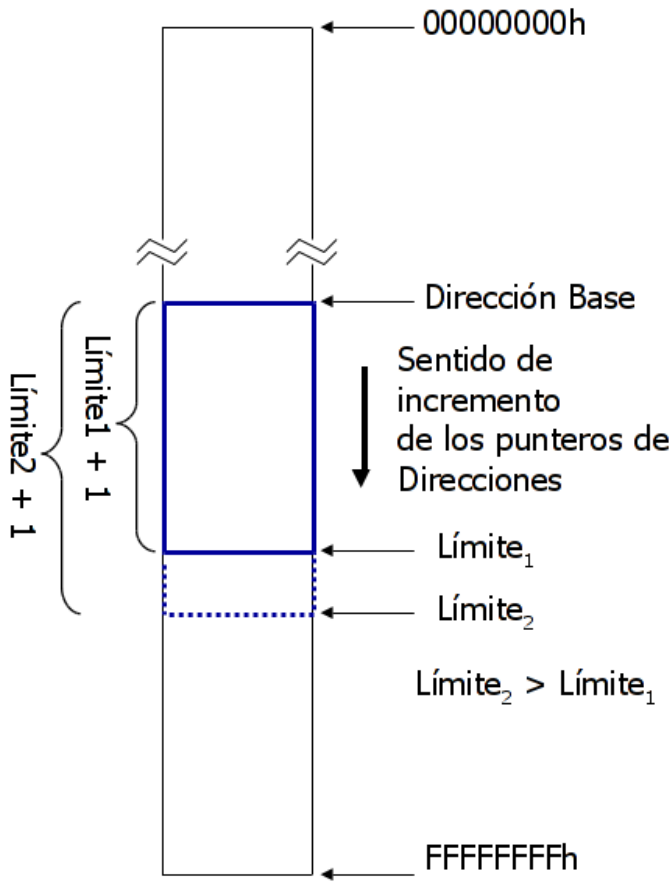
Límite Efectivo: Es la última dirección que se puede acceder dentro del segmento. Se generará una excepción 0x0D cada vez que se intente acceder a un offset mayor que el límite efectivo.

Ejemplos:

- Un byte cuyo offset sea mayor que el valor del Límite efectivo
- Una word cuyo offset sea mayor que (Límite efectivo - 1)
- Una doubleword cuyo offset sea mayor que (Límite efectivo - 3)
- Una quadword cuyo offset sea mayor que (Límite efectivo - 7)

Cuando se está por alcanzar el límite de un segmento se lo puede redimensionar aumentando el valor del Límite (en el gráfico, límite1 a límite2), sin necesidad de relocar segmentos. No se afecta al programa en ejecución.





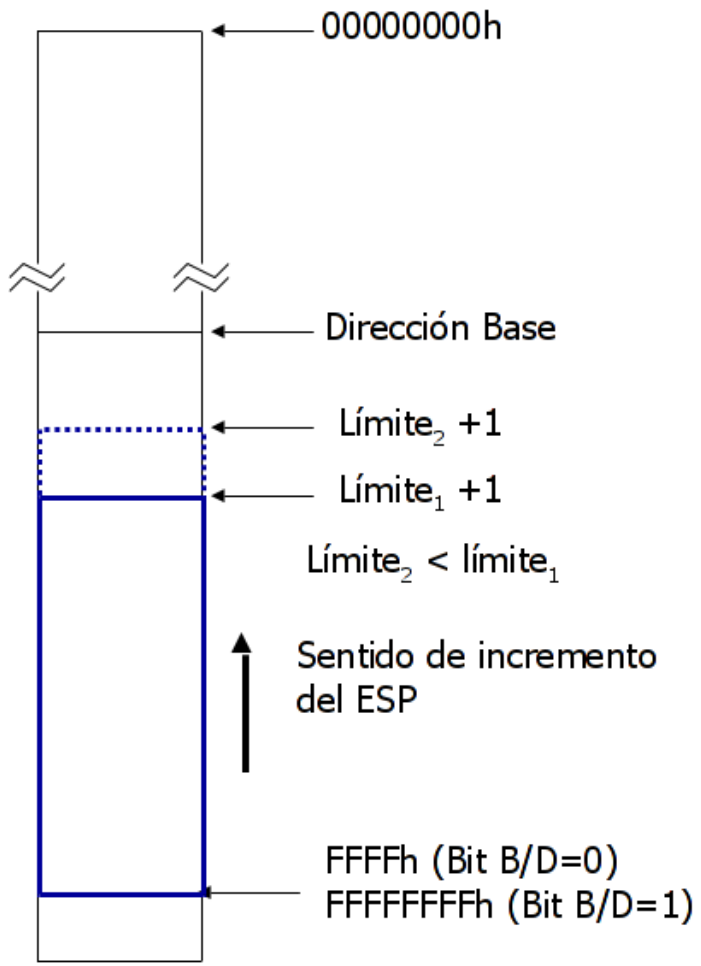
Chequeo de Límite si ED=1

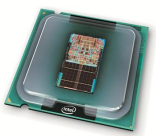
Límite Efectivo: Es el último offset que no puede ser accedido ya que generará la excepción #GP.

Para estos segmentos el rango de offsets válidos va desde:

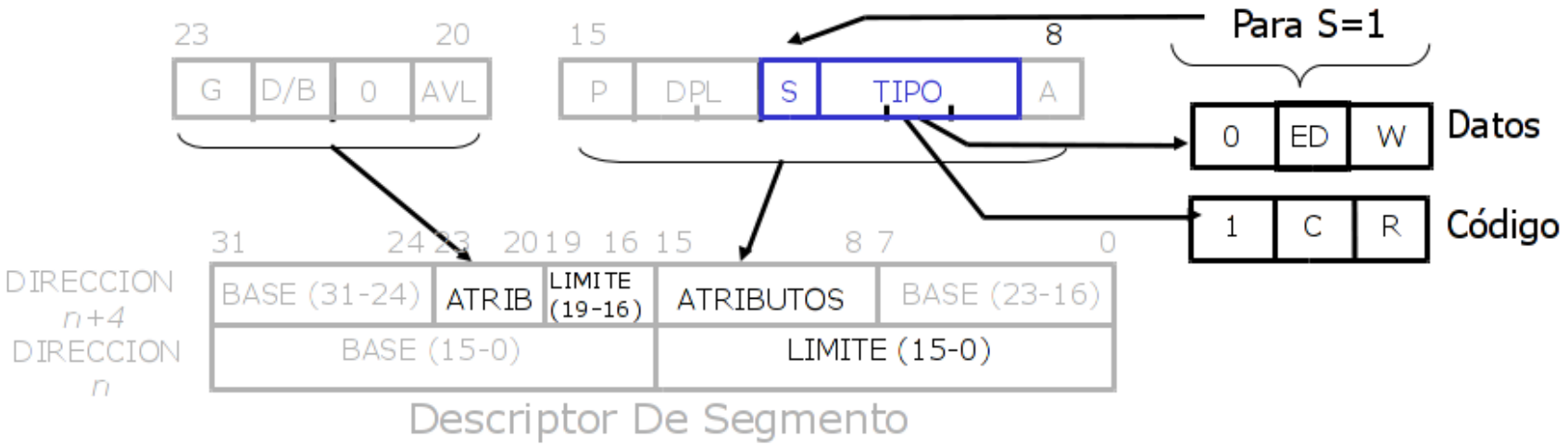
- (límite efectivo +1) hasta 0xFFFF (bit B/D=0)
- (límite efectivo +1) hasta 0xFFFFFFFF (bit B/D=1)

Este tipo de segmentos son útiles cuando contienen pilas. Los punteros de Pila en la arquitectura Intel decrementan su valor a medida que la pila se llena.

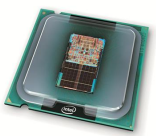




Elementos que el procesador toma en cuenta



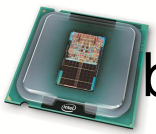
Se toma en cuenta el bit ***S*** (System bit) de los atributos del descriptor, y en función de su valor, se comprueba el campo de cuatro bits subsiguiente



Comprobaciones en selectores

En las instrucciones que cargan un valor de un selector en un registro de segmento no está permitido:

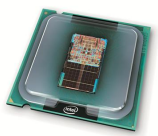
- Cargar CS un selector cuyo descriptor de segmentos corresponda a un segmento de datos (S=1 y tipo = 0XX).
- Cargar en SS, DS, ES, FS, o GS, un selector cuyo descriptor corresponde a un segmento de código que tiene R=0.
- Cargar en SS un selector cuyo descriptor de segmento corresponde a un segmento de datos (S=1 y tipo = 0XX) pero que tiene W=0.
- Cargar en LDTR un selector que no corresponda a un descriptor de LDT (S=0 y tipo = 0010b)
- Cargar en TR un selector que no corresponda a un descriptor de TSS (S=0 y tipo = 0001b, 0011b, 1001b, 1011b, o sea, TSS de 16 bits, disponible y ocupado, y TSS de 32 bits disponible y ocupado respectivamente)



Comprobaciones en selectores

Durante la ejecución de instrucciones que acceden a segmentos cuyo selector ya está cargado:

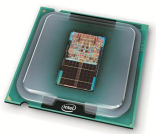
- No se puede escribir dentro de un segmento de código.
- No se puede escribir dentro de un segmento de datos con W=0.
- No se puede leer un segmento de código si R=0.



Comprobaciones en selectores

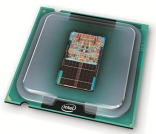
Merece un análisis particular el caso de las instrucciones ***CALL far*** y ***JMP far***

- Ambas contienen en su operando un selector. El descriptor correspondiente debe ser un segmento de código, una puerta de llamada, una puerta de tarea, o un TSS. En otro caso #GP.
- El tratamiento de éstas instrucciones depende del campo tipo en el descriptor.
  - Si corresponde a un segmento de código, se ejecuta la llamada o el salto incondicional al segmento de código indicado.
  - Si corresponde a una puerta de llamada, se ejecuta la llamada a un servicio de mayor nivel de privilegio como veremos mas adelante. La Unidad de Protección controlará que el segmento destino sea de código.
  - Si corresponde a un TSS o a una puerta de tarea, se ejecuta una conmutación de tarea. La Unidad de Protección controlará que el descriptor de segmento correspondiente al selector contenido en la puerta de tarea sea un descriptor de TSS.



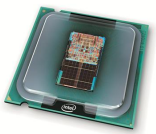
Chequeo del Tipo Comprobaciones Misceláneas

- Cada entrada de la IDT debe tener una puerta de interrupción, de excepción, o de tarea. De otro modo cualquier acceso a dicha entrada de la IDT por medio de una instrucción INT, Interrupción de hardware o Excepción causará una excepción #GP.
- Cuando se accede a una puerta de Trap o de interrupción la Unidad de Protección controla que el descriptor de segmento definido en el descriptor de la puerta corresponda a un segmento de código.
- Cuando se retorna de una tarea anidada (mediante una instrucción IRET), la unidad de Protección controla que el selector del campo previous task link en el TSS actual, corresponda a un descriptor de TSS.

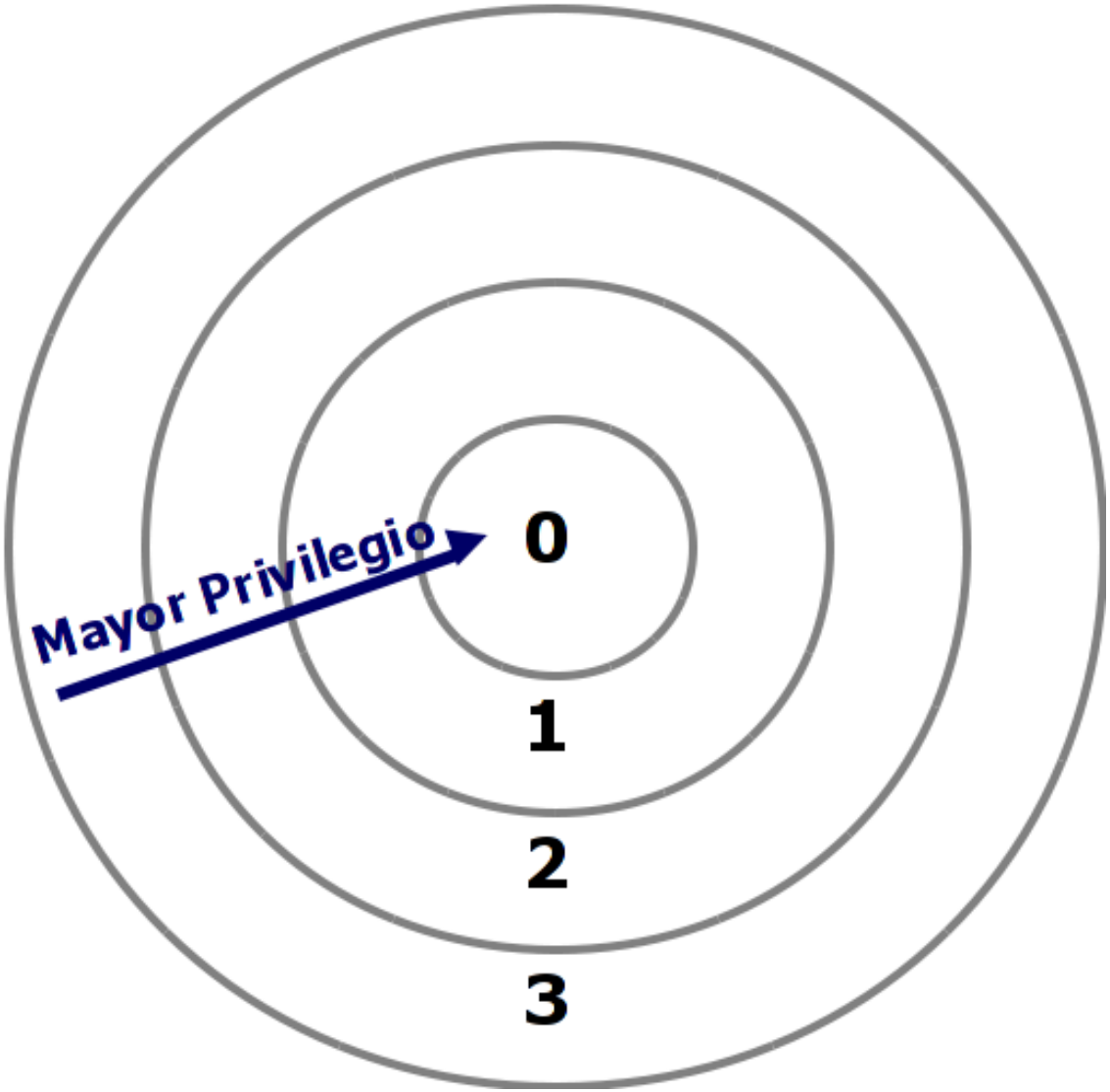


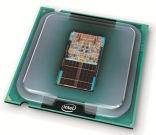
Selectores de segmento nulo

- Intentar asignar un selector de segmento Nulo en los registros de segmento CS or SS genera una excepción #GP de inmediato.
- No así cuando se asigna un selector de segmento nulo a los registros DS, ES, FS, or GS. Sin embargo, al intentar acceder al segmento por medio del registro cargado con el selector de segmento nulo generará una excepción #GP.
- Cargar un registro de segmento de datos con un selector de segmento nulo puede resultar un método útil para detectar accesos a registros de segmento no utilizados o para prevenir accesos no deseados a segmentos de datos.



Modelo de anillos





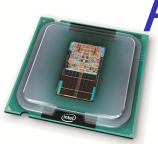
Modelo de anillos

- Intel suele representar el modelo de protección en forma de anillos concéntricos.
- Los mas internos tienen mayor nivel de privilegio
- Los segmentos se ubican en el anillo correspondiente de acuerdo con el valor del campo DPL de su descriptor.
- Uso propuesto por Intel:

  Anillo 0 Kernel del S.O.
  Anillos 1 y 2 Servicios del S.O.
  Anillo 3 Aplicaciones

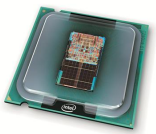
- Uso real en los Sistemas Operativos mas difundidos:

  Anillo 0 Kernel y servicios del S.O.
  Anillos 1 y 2 No utilizados
  Anillo 3 Aplicaciones



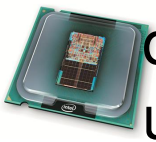
Mas sobre Niveles de Privilegio

- Los accesos entre segmentos están regidos por reglas que contemplan los niveles de privilegio.
- Los niveles de privilegio se chequean cuando se carga el selector en el registro de segmento.
- Se definen tres tipos de nivel de privilegio
  1. ***Descriptor Privilege Level (DPL)***
  2. ***Current Privilege Level (CPL)***
  3. ***Requested Privilege Level (RPL)***



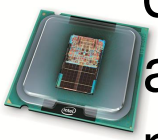
Mas sobre Niveles de Privilegio

1. ***Descriptor Privilege Level (DPL)***: Nivel de privilegio del segmento a ser accedido: Puerta de llamada, de tarea, TSS, o Descriptor de segmento de código o datos.
2. ***Current Privilege Level (CPL)***: Nivel de privilegio que tiene el código que está intentando acceder ya sea a un descriptor de segmento, una puerta de llamada o de tarea, etc. El procesador lo mantiene en el cache hidden del selector, ya que lo leyó directo de la tabla de descriptores.
3. ***Requested Privilege Level (RPL)***: Es el valor que se escribe en los bits 0 y 1 de los selectores. Es derogable, ya que puede sobreescribirse por programa. El procesador lo compara con el CPL y si es numéricamente menor, lo reemplaza por el CPL al momento de chequear el acceso. Si el RPL es mayor (numérico) que el CPL, el procesador usará el RPL. En suma, el procesador usa lo que se define como ***Effective Priviledge Level EPL=MAX(CPL,RPL)***

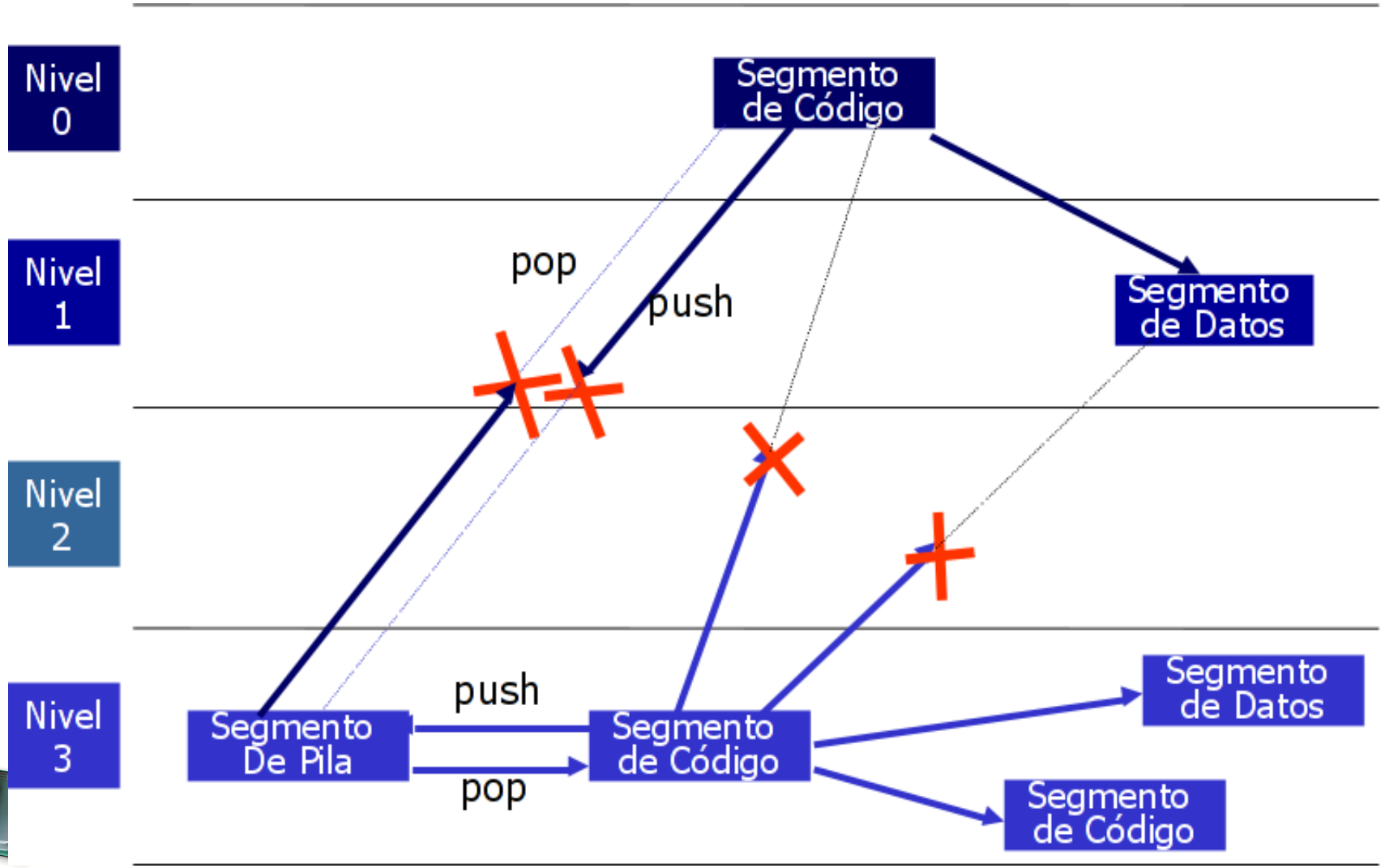


Significado del campo DPL del descriptor

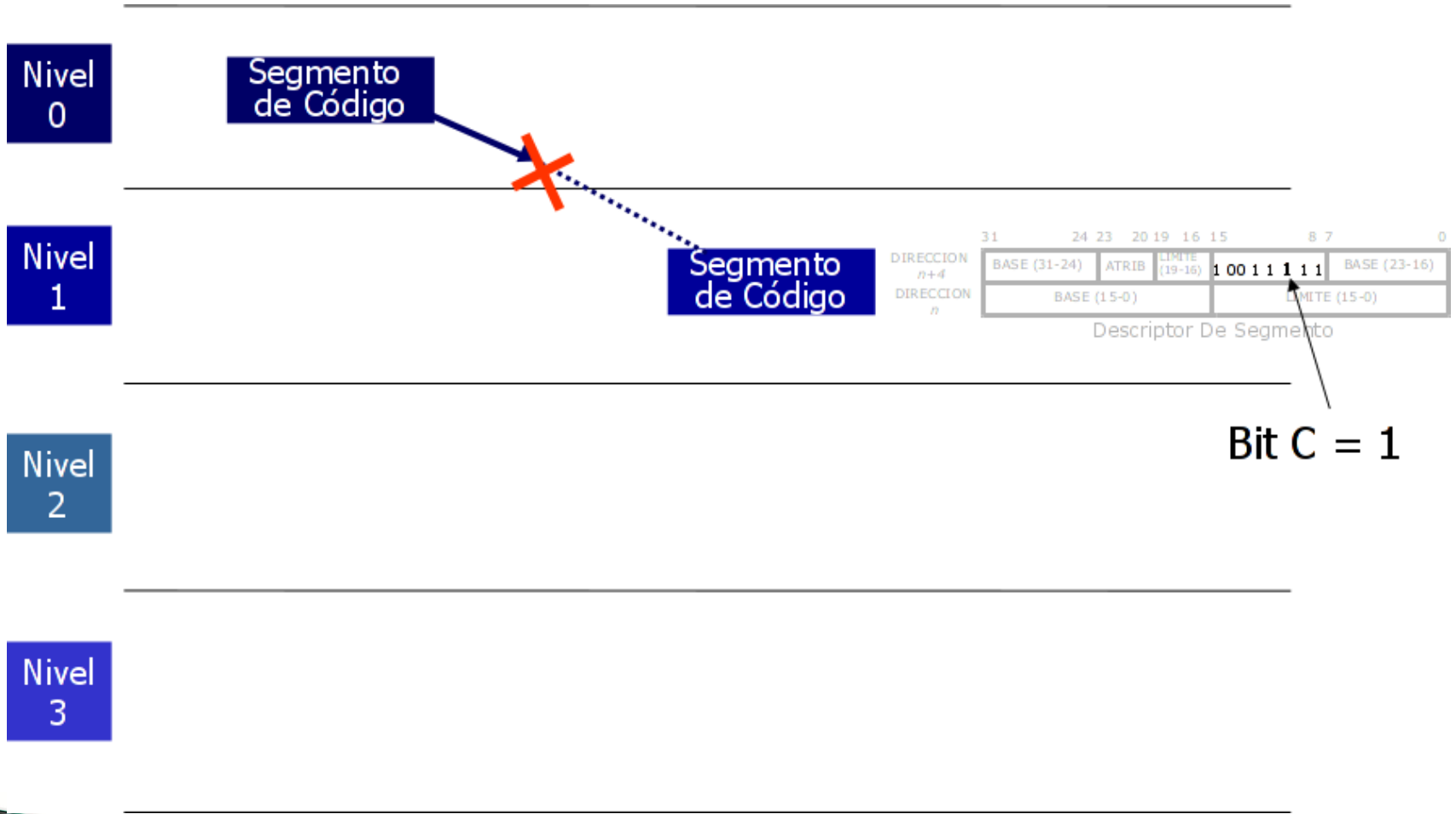
- ***Segmento de Datos / Puerta de Llamada / TSS***: Indica el máximo valor numérico (mínimo nivel de privilegio) que debe tener el código de una tarea para acceder a este segmento. Ej.: DPL=01 implica que se puede acceder solamente desde segmentos de código con CPL=00 o CPL=01.
- ***Segmento de Código No Conforming (sin utilizar puerta de llamada)***: Es el nivel de privilegio que debe tener el código de una tarea para accederlo. Ej.: DPL=00 implica que solo los programas que ejecuten en segmentos con CPL=00.
- ***Segmento de código Conforming, o segmento de código no Conforming accedido a través de una puerta de llamada***: Indica el mínimo valor numérico (máximo nivel de privilegio) que debe tener el código que llamó a la puerta de llamada para poder acceder al código apuntado por ésta. Ej.: DPL=10, indica que los programas de anillo 0 y 1 no pueden acceder a este segmento.

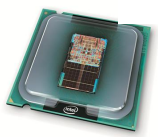


Reglas de Protección entre segmentos

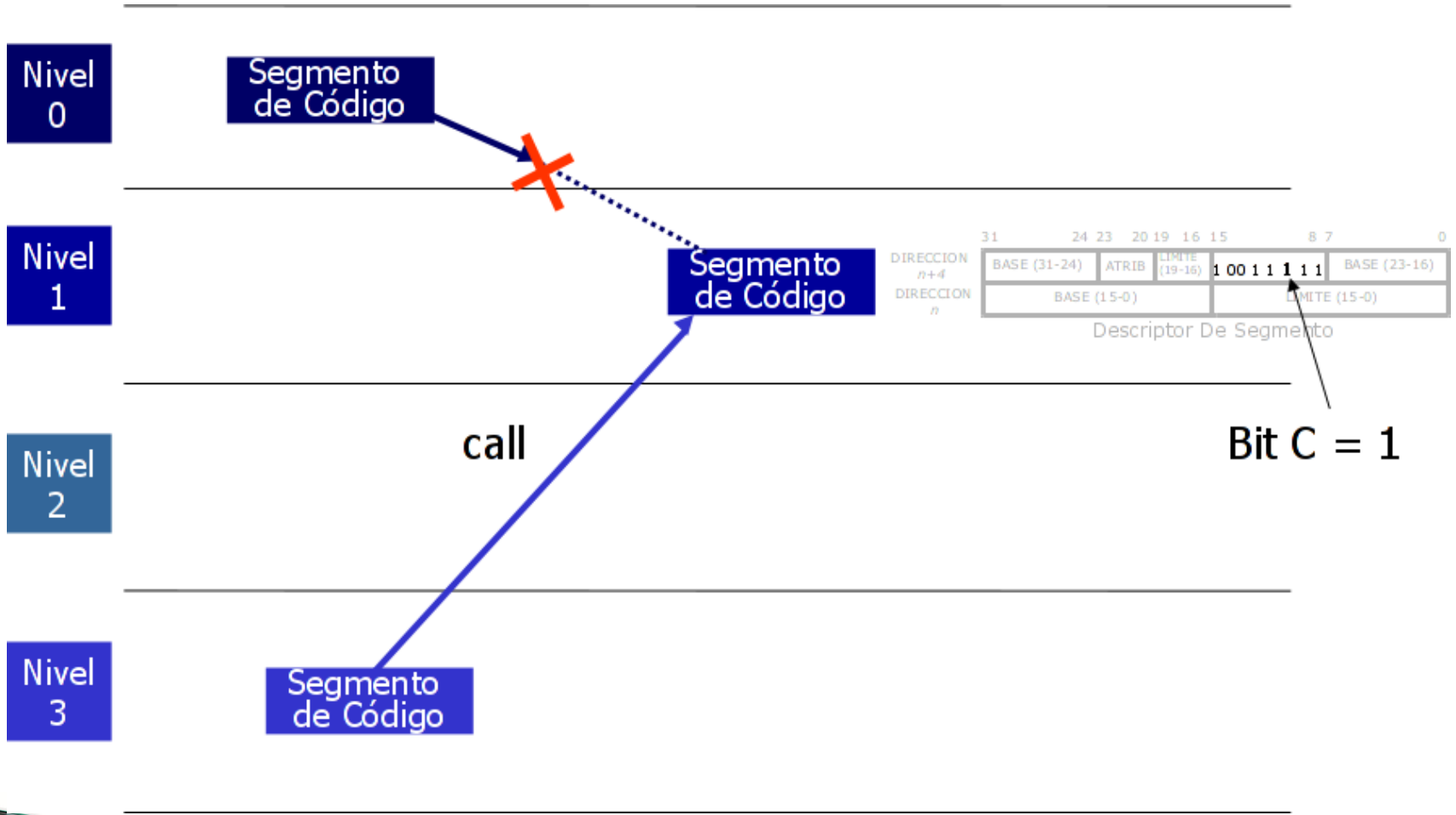


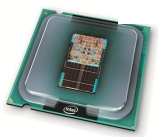
Segmentos Conforming



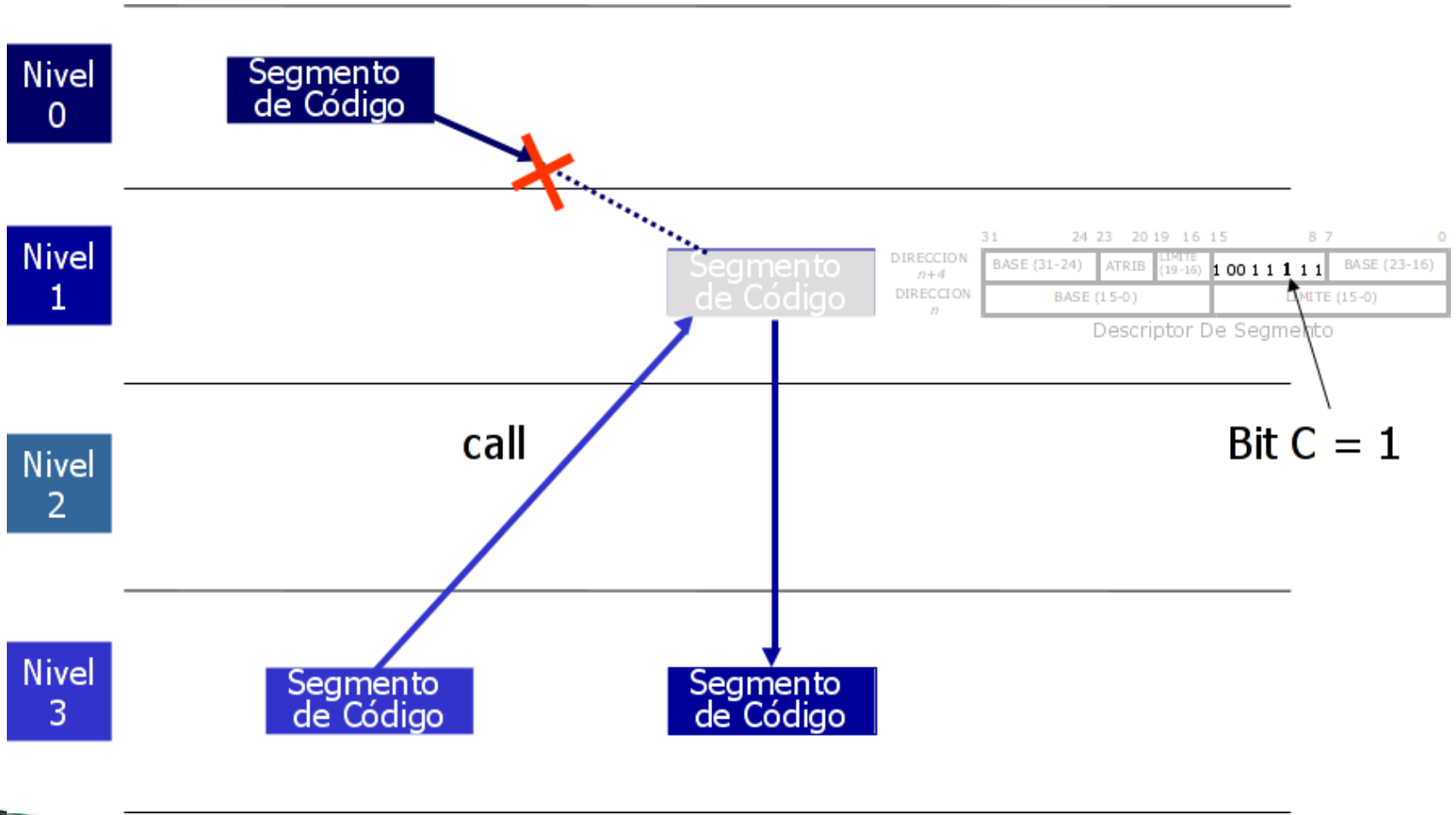


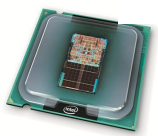
Segmentos Conforming





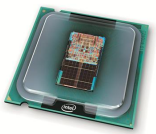
Segmentos Conforming





Puertas de Llamada

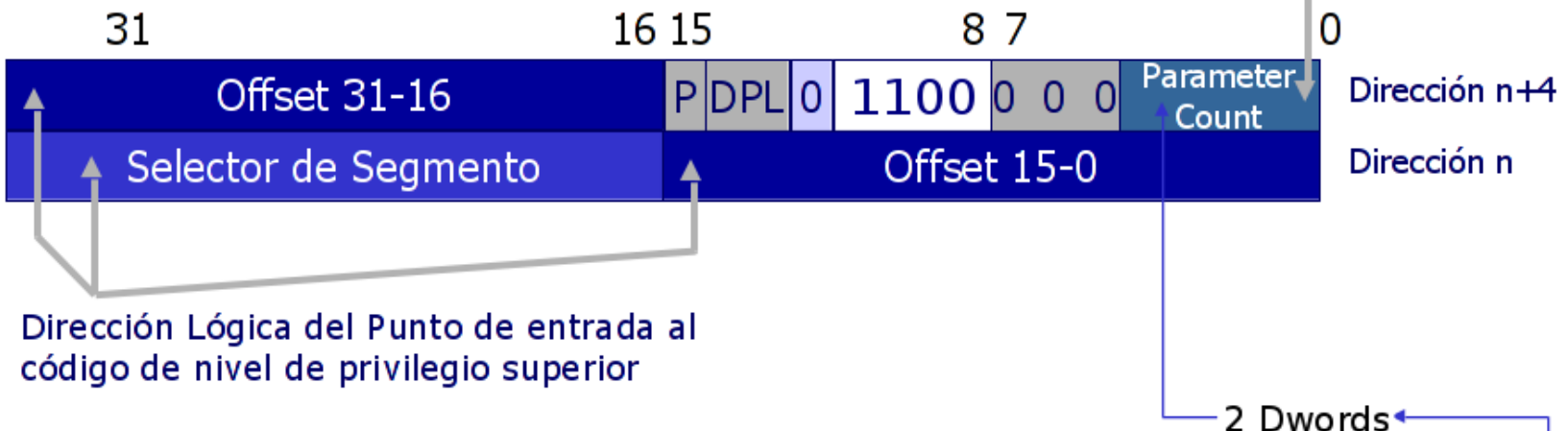
- Son un mecanismo especial para transferir el control desde un segmento de código a otro de mayor nivel de privilegio, cuando se encuentra habilitado el Sistema de Protección del procesador.
- Se basan en un descriptor de sistema (bit S=0), que debe residir en la GDT o en la LDT (nunca en la IDT)
- Se acceden efectuando un CALL o un JMP a una dirección far (dirección lógica), en la que el selector corresponde a un descriptor de Puerta de Llamada, y el offset es ignorado por el procesador.



Descriptor de Puerta de Llamada

Cantidad de parámetros a transferir por la pila (doble word implica segmentos de 32 bits)

Descriptor de Puerta de llamada de 32 bits

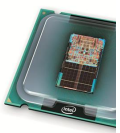


Llamada en un programa C

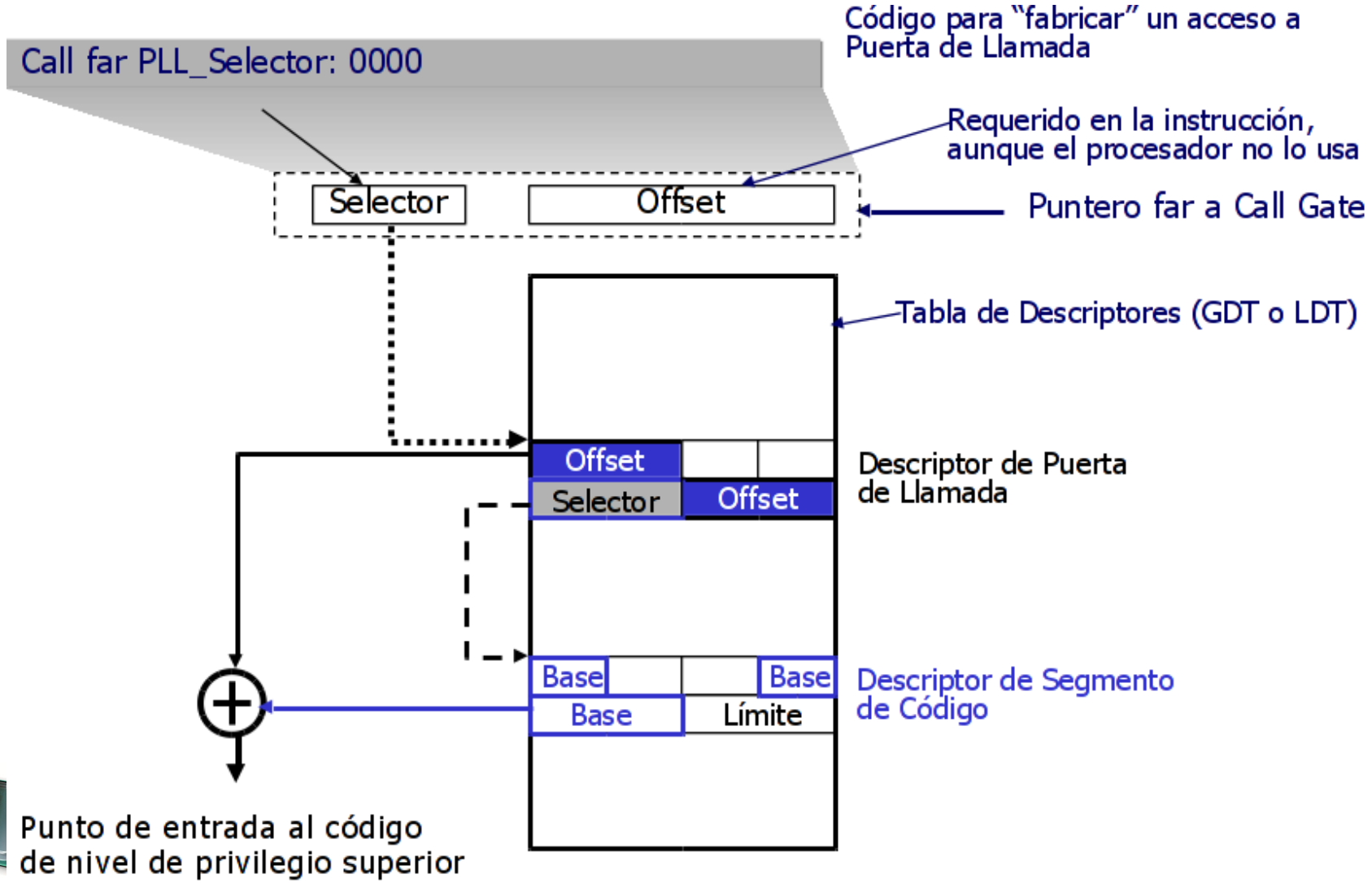
servicio (arg1,arg2)

Assembly generado por el compilador

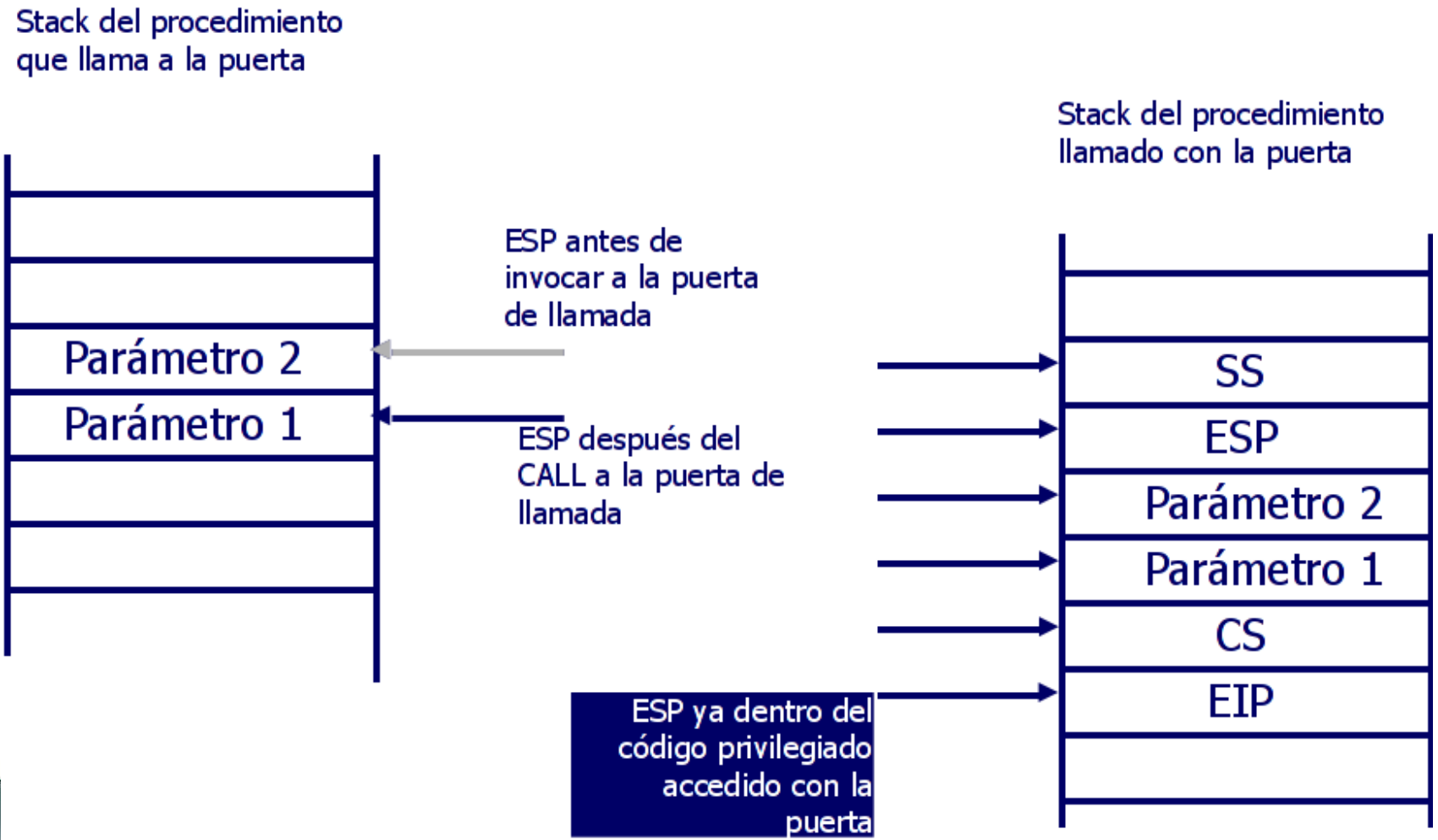
```
push arg2
push arg1
call servicio
```



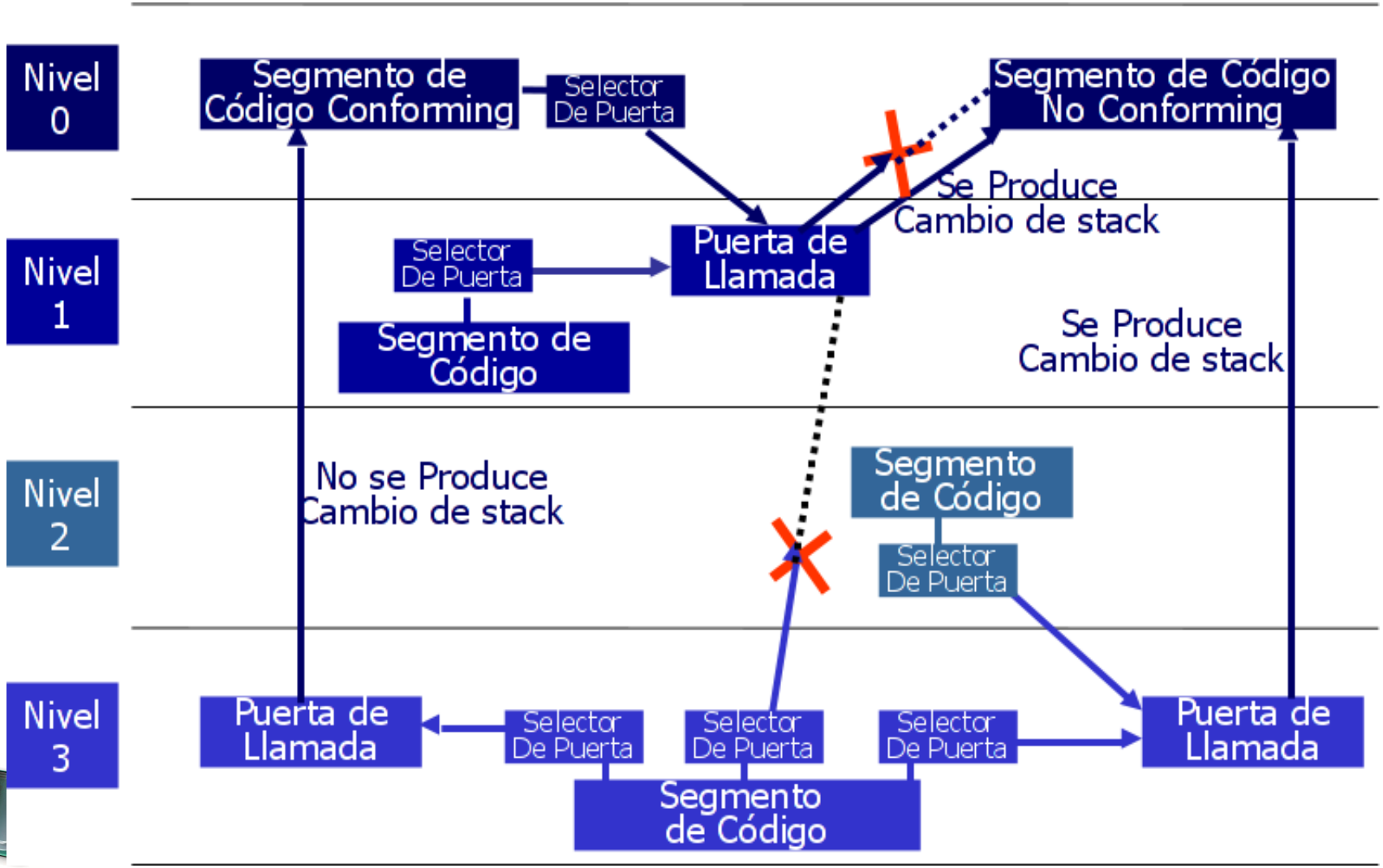
Acceso al código mediante Puerta de Llamada



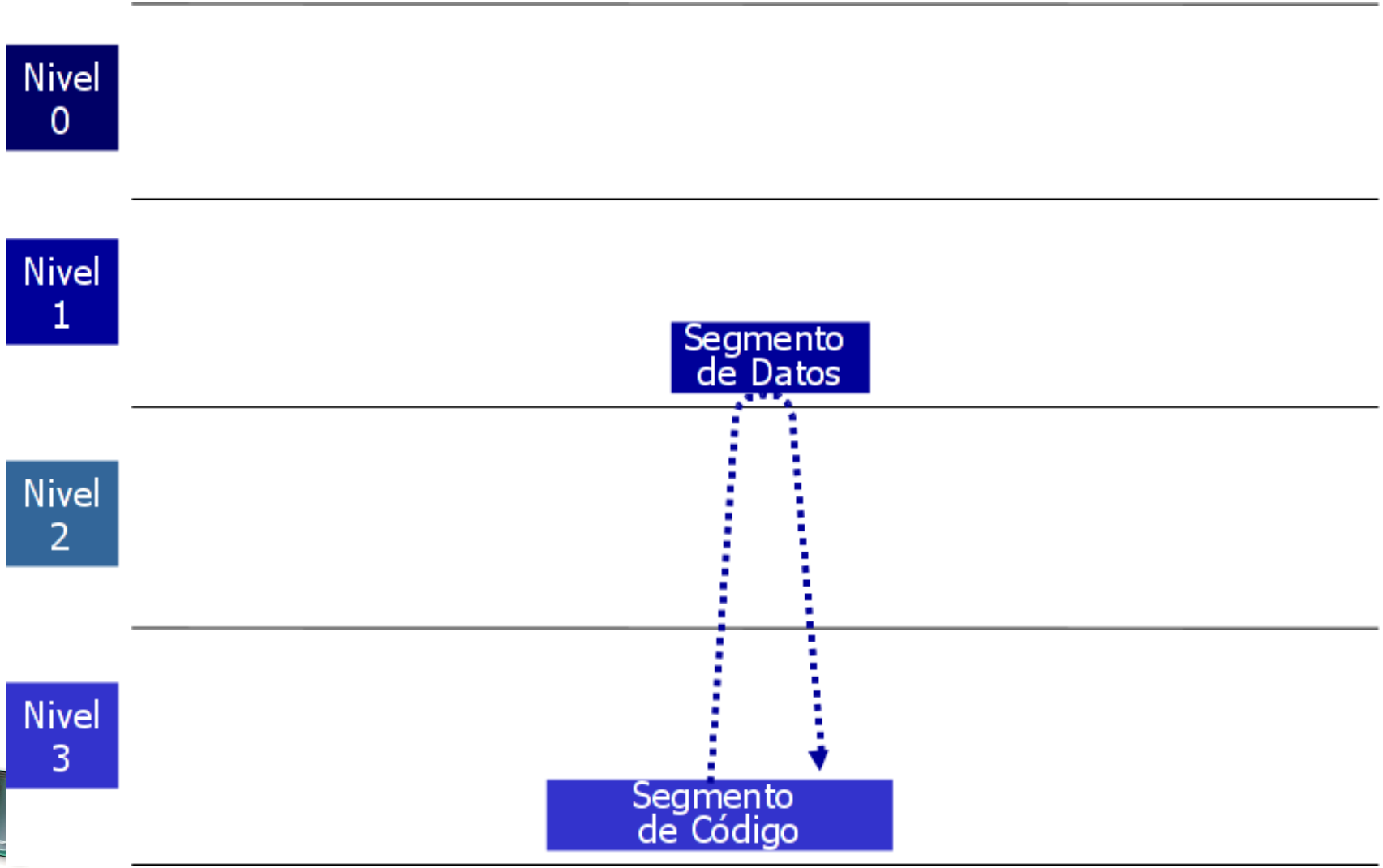
Pila en un cambio de nivel de privilegio



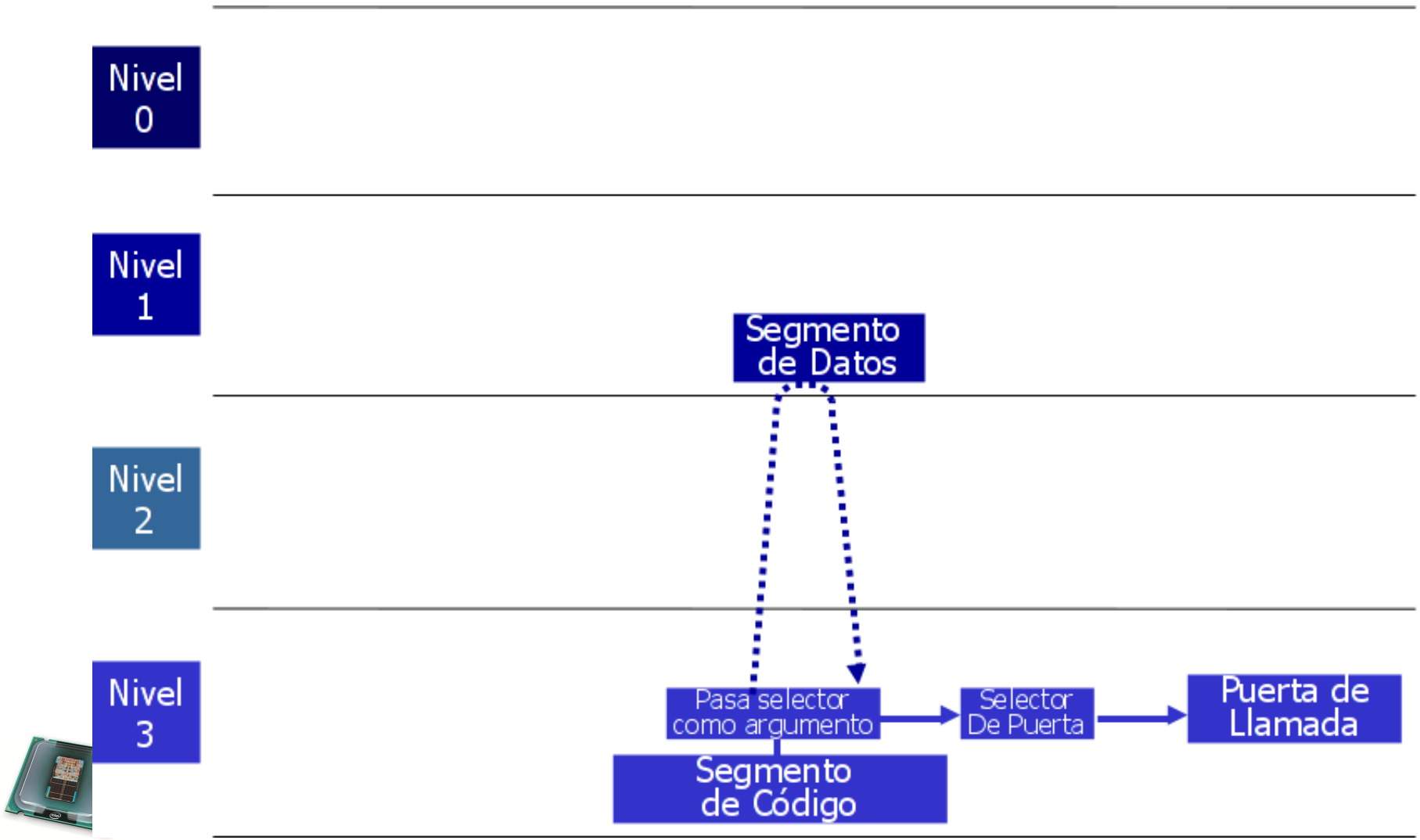
Puertas de Llamada: resumen



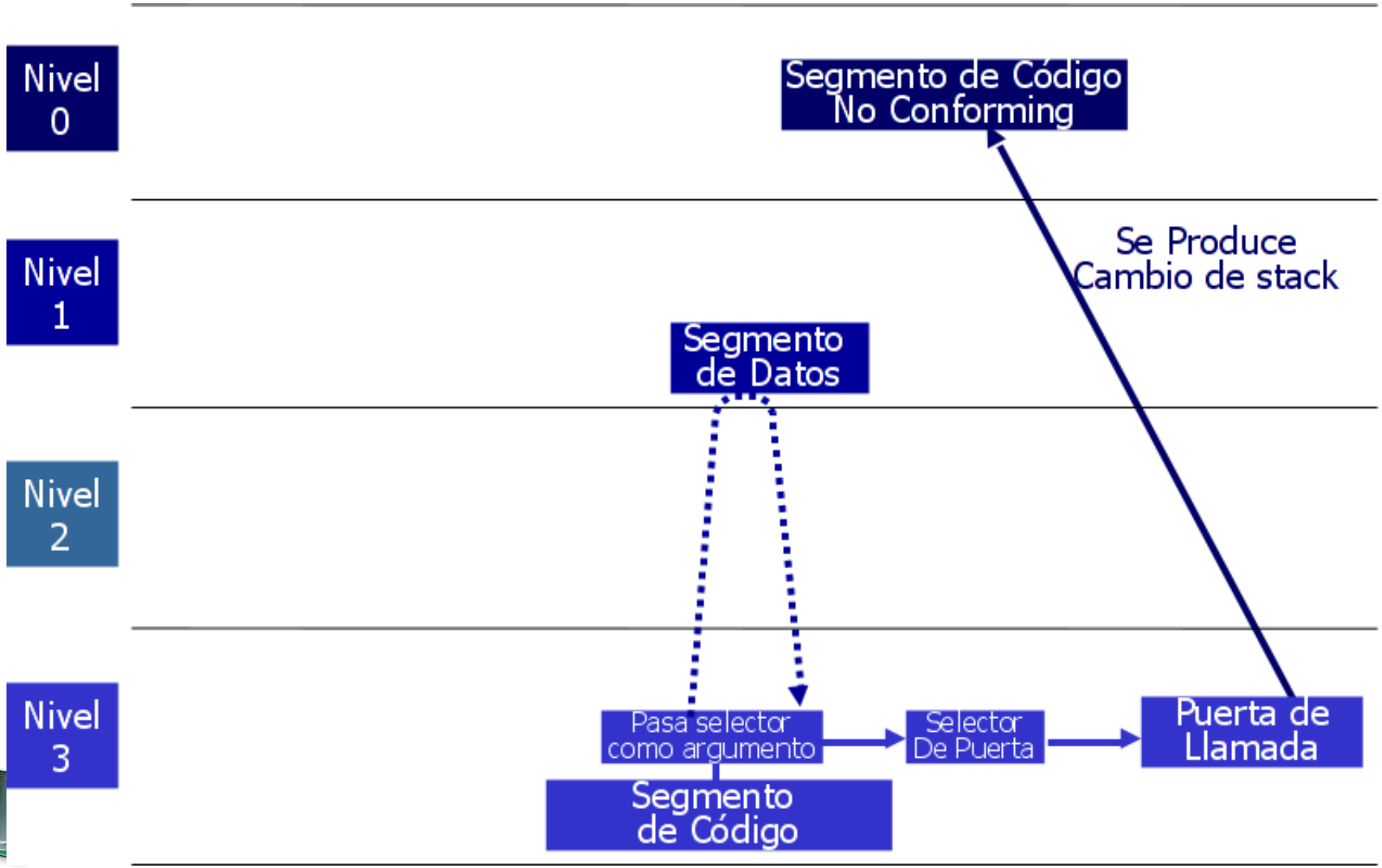
Escenario del Caballo de Troya

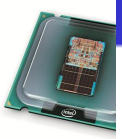


Escenario del Caballo de Troya

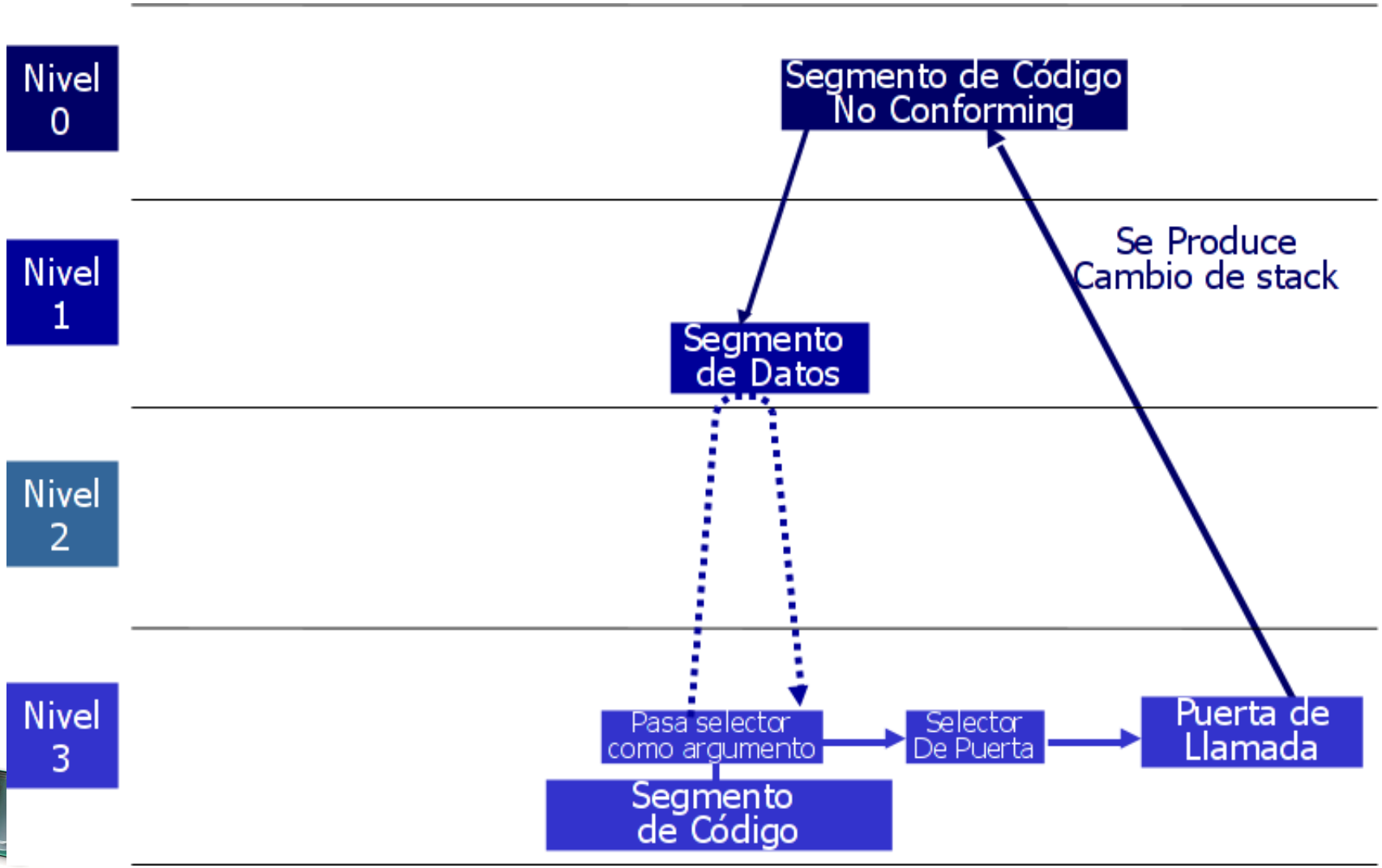


Escenario del Caballo de Troya

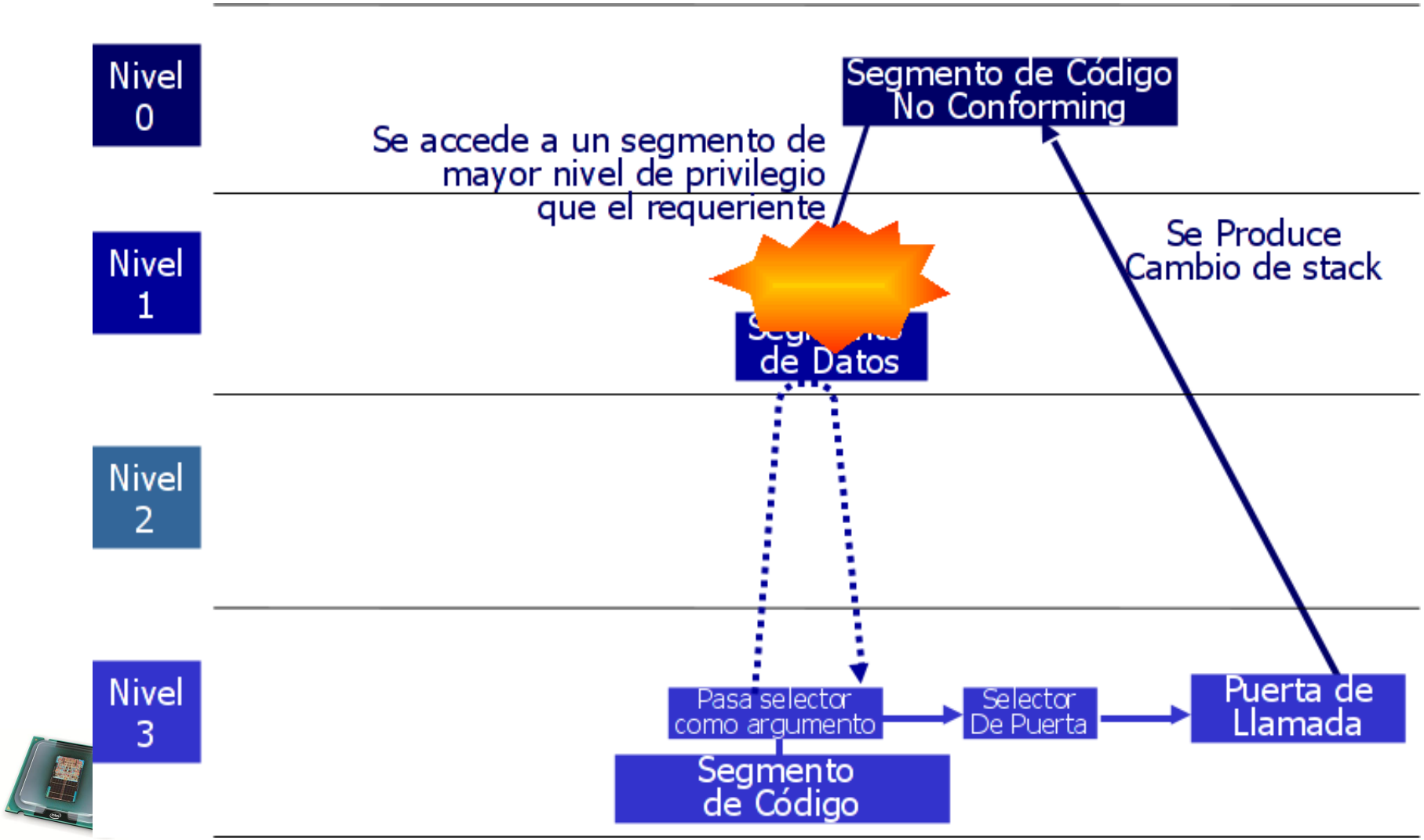




Escenario del Caballo de Troya

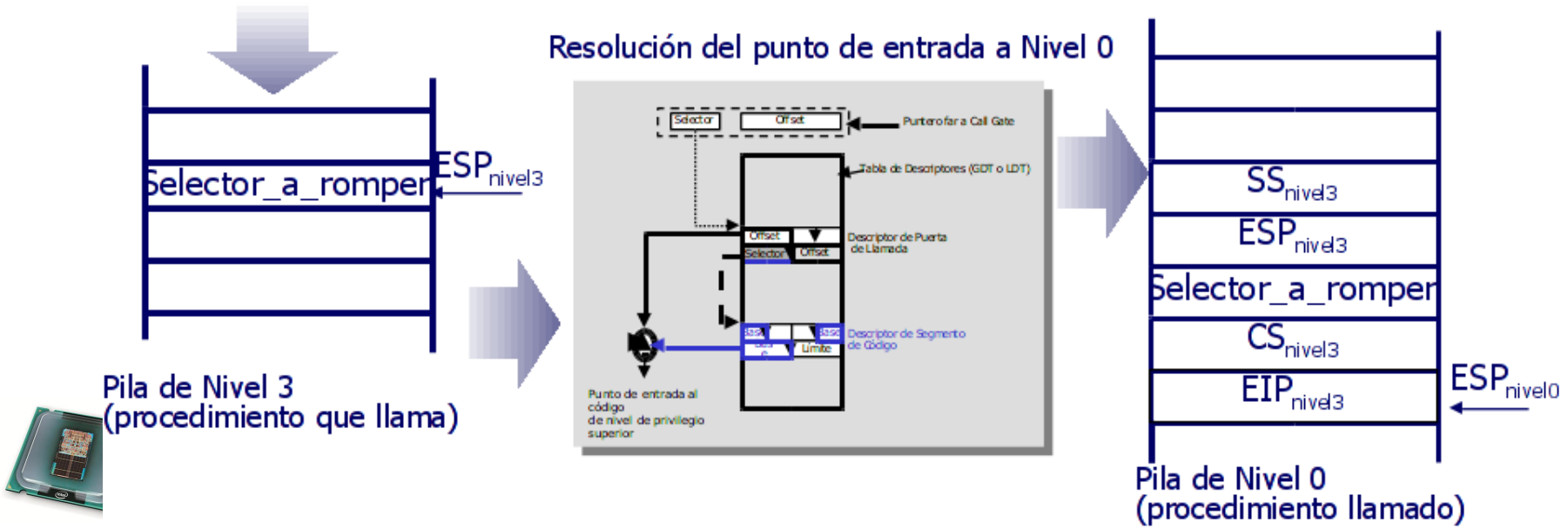


Escenario del Caballo de Troya



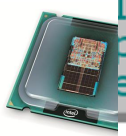
En código un inocente troyano

```
;Este programa ejecuta en un segmento de código con DPL = 11
mov     ax,219h     ;ax=0000 0010 0001 1001 ← RPL
;el programador intenta acceder al segmento de Nivel de Privilegio 1
push    ax
call    PLL_Selector:0 ;Llama al servicio del kernel
```



ARPL: La forma de prevenirlo

```
;Punto de entrada del programa de Nivel 0
(Offset ;que debe figurar en el descriptor de ;puerta
de ;llamada)

pushebp
mov ebp,esp

;extrae selector de CS del requeriente.
mov dx, word ptr [ebp+8]

;ajusta el nivel de privilegio del selector pasado
;como parámetro  (con RPL=01), con el del CS
de ;nivel 3, que es el privilegio del requeriente.
arpl  word ptr [ebp+12], dx

;lee selector ajustado (ahora RPL = 11)
mov ds, word ptr [ebp+12]

;En cuanto realice el acceso al segmento, excepción
;0Dh. El acceso no es posible.
;Solo debe manejarse el handler de la
Interrupción ;adecuadamente y regresar al
programa origen en ;el nivel 3, con un código de
error.
```

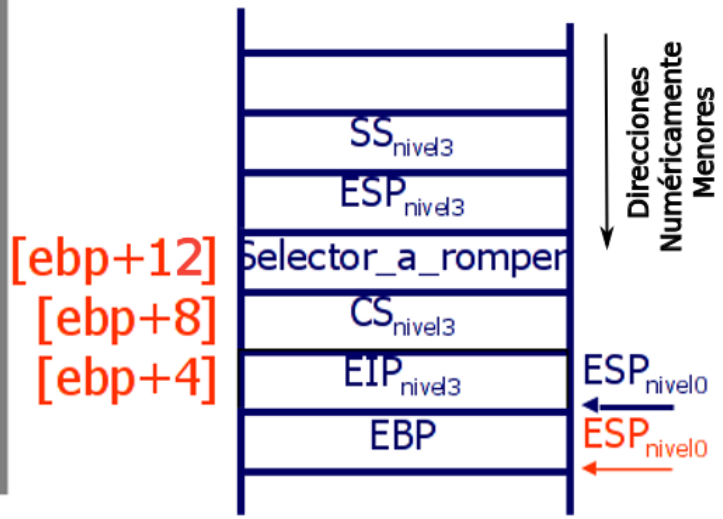


Se resuelve con al instrucción **ARPL**. Al igual que el bit conforming resuelve al acceso trabajando con el Nivel de Privilegio Efectivo

$$EPL = MAX(CPL, RPL)$$

ARPL r/m16, r16

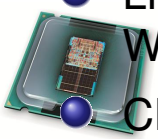
Resultado: El operando destino adquiere el RPL del operando fuente, si es mayor numéricamente (menos privilegiado), caso contrario conserva el que tenía.



Que se puede hacer y que no...

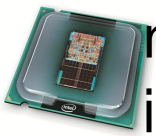
El Registro ***EFLAGS*** contiene el campo IOPL (bits 12 y 13) que determina el nivel de privilegio que debe tener la tarea en curso para acceder a la E/S. Por lo tanto, pueden ejecutarse IN, OUT, INS, y OUTS, desde esta tarea solo si CPL = IOPL.
Existen instrucciones que, una vez en modo protegido, solo pueden ejecutarse en nivel de privilegio 0. Estas son:

- LGDT - Cargar registro GDTR
- LLDT - Cargar registro LDTR
- LTR - Cargar registro TR
- LIDT - Cargar Registro IDTR
- MOV - si destino es un Registro de Control
- MOV - si destino es un Registro de Debug
- LMSW - Escribir en el Machine Status Word (parte baja de CR0)
- CLTS - Clear Flag Task-Switched en CR0
- INVD - Invalidar Caché sin Write Back
- WBINVD - Invalidar Caché con Write Back
- INVLPG - Invalidar entrada de la TLB
- HLT - Parar el procesador
- RDMSR - Leer Model Specific Register
- WRMSR - Escribir Model Specific Register
- RDPMC - Leer Contador de Monitoreo de Performance
- RDTSC - Leer Time Stamp Counter



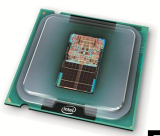
Una vez que pasamos la Unidad de Segmentación...

- Se combina con la Protección a nivel de segmentos, aportando granularidad al sistema de protección dentro de un mismo segmento.
- Al igual que en los segmentos los chequeos se realizan dentro de la ventana de decodificación y ejecución de la instrucción.
- De acuerdo al sistema de protección de páginas (reflejado en la estructura del descriptor de página) hay dos niveles: Usuario (bit U/S=0) y Supervisor (U/S=1), y el tipo de página puede ser de Lectura o Lectura/Escritura.
- Cualquier violación al sistema de protección de páginas genera una Excepción typo 0x0E definida por Intel como Page Fault (#PF), y rebautizada bajo el misterioso y tristemente familiar nombre de "**`Error grave 0Eh`**" en algunas ya antiguas implementaciones...;−)



Hilando fino con los permisos en Paginación

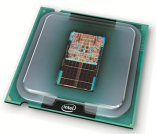
- Los CPL's 0, 1, y 2 del sistema de protección de segmentos mapean en el nivel Supervisor del esquema de páginas.
- El CPL3, se corresponde con el nivel Usuario.
- En el modo supervisor se accede a todas las páginas y en modo usuario solo a las que tienen en su descriptor el bit U/S = 1
- En el caso particular de la escritura. Un código supervisor escribe en cualquier página, aunque el descriptor de ;a misma tenga el atributo R/W=0 (es decir, read only)
- A partir del 80486, el bit 16 del registro CR0, se utiliza bajo el nombre WP (Write Protect).
  - Cuando el procesador esta en Modo Supervisor, por default, accede a cualquier página con permiso de lectura escritura. (Se ignora la protección de escritura)
  - Si es 1 impide al procesador escribir una página Read Only de nivel usuario desde código que ejecuta en una página en Modo Supervisor.



- El procesador chequea la protección en el PD, y en cada PT.

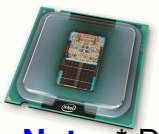
Combinando protección de Segmentos y Páginas

- El procesador evaluará siempre en primer lugar la protección de segmentos, ya que la Unidad de Paginación puede o no estar habilitada.
- Si genera una excepción por segmentación no se genera la #PF.
- La protección a nivel de página no pisa a la protección a nivel de segmento. Paginar un segmento de código fijando permisos de escritura en las páginas, no permitirá escribir, ya que lo impedirá el mecanismo de protección de segmentos.
- En el caso de un segmento de datos con permiso de escritura, la paginación permite definir diferentes permisos para cada página con lo cual podremos dividirlo en áreas de lectura solamente y otras de lectura/escritura.



Combinando protección de Segmentos y Páginas

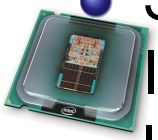
Page-Directory Entry		Page-Table Entry		Combined Effect	
Privilege	**Access Type**	**Privilege**	**Access Type**	**Privilege**	**Access Type**
User	Read-Only	User	Read-Only	User	Read-Only
User	Read-Only	User	Read-Write	User	Read-Only
User	Read-Write	User	Read-Only	User	Read-Only
User	Read-Write	User	Read-Write	User	Read/Write
User	Read-Only	Supervisor	Read-Only	Supervisor	Read/Write*
User	Read-Only	Supervisor	Read-Write	Supervisor	Read/Write*
User	Read-Write	Supervisor	Read-Only	Supervisor	Read/Write*
User	Read-Write	Supervisor	Read-Write	Supervisor	Read/Write
Supervisor	Read-Only	User	Read-Only	Supervisor	Read/Write*
Supervisor	Read-Only	User	Read-Write	Supervisor	Read/Write*
Supervisor	Read-Write	User	Read-Only	Supervisor	Read/Write*
Supervisor	Read-Write	User	Read-Write	Supervisor	Read/Write
Supervisor	Read-Only	Supervisor	Read-Only	Supervisor	Read/Write*
Supervisor	Read-Only	Supervisor	Read-Write	Supervisor	Read/Write*
Supervisor	Read-Write	Supervisor	Read-Only	Supervisor	Read/Write*
Supervisor	Read-Write	Supervisor	Read-Write	Supervisor	Read/Write



Nota: * Depende de CRO.WP.
Si CR0.WP=1, depende de la combinación de la tabla, Si CR0.WP=0, el privilegio de Supervisor habilita la escritura

Execute Disable

- Si se trabaja con PAE habilitada, o en modo IA-32e, el procesador habilita en la unidad de paginación una función para proteger páginas de datos, evitando que se pueda depositar código, digamos, malicioso, en una página en forma de datos y luego utilizarlo, en detrimento de la integridad del sistema.
- No requiere nuevas instrucciones. Simplemente utiliza el bit 63 del descriptor de página en la estructura de paginación.
- Una página con esta facilidad habilitada, solo puede ser utilizada como datos. Cualquier intento de ejecutar código en ella generará inevitablemente una excepción #PF.
- Para detectar si el procesador tiene esta función disponible: CPUID.80000001H:EDX.NX [bit 20] = 1.
- Si está disponible esta capacidad, se la habilita poniendo IA32_EFER.NXE[bit 11] = 1. Escribir directamente este registro si la Execute Disable no está disponible, genera una excepción #GP.

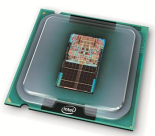


Execute Disable: como hacer

Execute Disable Bit Value (Bit 63)				Valid Usage
PML4	PDP	PDE	PTE	
Bit 63 = 1	*	*	*	Data
*	Bit 63 = 1	*	*	Data
*	*	Bit 63 = 1	*	Data
*	*	*	Bit 63 = 1	Data
Bit 63 = 0	Bit 63 = 0	Bit 63 = 0	Bit 63 = 0	Data/Code

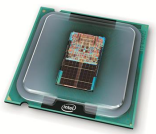
Execute Disable Bit Value (Bit 63)		Valid Usage
PDE	PTE	
Bit 63 = 1	*	Data
*	Bit 63 = 1	Data
Bit 63 = 0	Bit 63 = 0	Data/Code

Execute Disable Bit Value (Bit 63)	Valid Usage
PDE	
Bit 63 = 1	Data
Bit 63 = 0	Data/Code



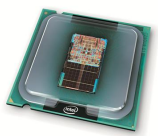
Check de bits reservados

- El procesador fuerza el chequeo de los bits que figuran reservados en las estructuras de paginación en función del modo de paginación en el que esté trabajando y de la cantidad de bits de memoria física que es capaz de manejar.
- Para saber cuantos bits de dirección física se dispone en cada modelo de procesador CPUID.80000008H:EAX[bits 7-0] retornan el número binario correspondiente a la cantidad de bits de memoria física disponibles.



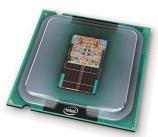
Check de bits reservados con Execute Disable Habilitado

Mode	Paging Mode	Check Bits
32-bit	4-KByte paging (non-PAE)	No reserved bits checked
	PSE36 - PDE, 4-MByte page	Bit [21]
	PSE36 - PDE, 4-KByte page	No reserved bits checked
	PSE36 - PTE	No reserved bits checked
	PAE - PDP table entry	Bits [63:MAXPHYADDR] & [8:5] & [2:1] *
	PAE - PDE, 2-MByte page	Bits [62:MAXPHYADDR] & [20:13] *
	PAE - PDE, 4-KByte page	Bits [62:MAXPHYADDR] *
	PAE - PTE	Bits [62:MAXPHYADDR] *
64-bit	PML4E	Bits [51:MAXPHYADDR] *
	PDPTE	Bits [51:MAXPHYADDR] *
	PDE, 2-MByte page	Bits [51:MAXPHYADDR] & [20:13] *
	PDE, 4-KByte page	Bits [51:MAXPHYADDR] *
	PTE	Bits [51:MAXPHYADDR] *



Check de bits reservados con Execute Disable Deshabilitado

Mode	Paging Mode	Check Bits
32-bit	KByte paging (non-PAE)	No reserved bits checked
	PSE36 - PDE, 4-MByte page	Bit [21]
	PSE36 - PDE, 4-KByte page	No reserved bits checked
	PSE36 - PTE	No reserved bits checked
	PAE - PDP table entry	Bits [63:MAXPHYADDR] & [8:5] & [2:1]*
	PAE - PDE, 2-MByte page	Bits [63:MAXPHYADDR] & [20:13]*
	PAE - PDE, 4-KByte page	Bits [63:MAXPHYADDR]*
	PAE - PTE	Bits [63:MAXPHYADDR]*
64-bit	PML4E	Bit [63], bits [51:MAXPHYADDR]*
	PDPTE	Bit [63], bits [51:MAXPHYADDR]*
	PDE, 2-MByte page	Bit [63], bits [51:MAXPHYADDR] & [20:13]*
	PDE, 4-KByte page	Bit [63], bits [51:MAXPHYADDR]*
	PTE	Bit [63], bits [51:MAXPHYADDR]*

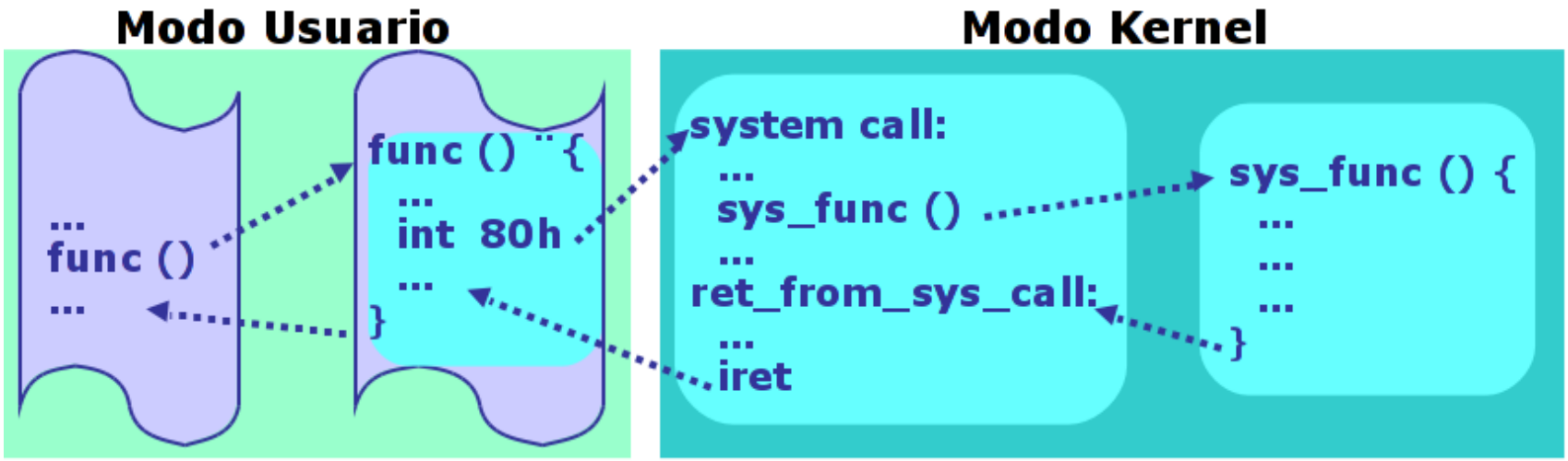


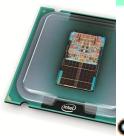
Estructura de una System Call

API : Formato de una llamada para obtener un servicio del kernel

System Call : Requerimiento explícito cursado al kernel para resolver un servicio.

La inversa no es necesariamente cierta. Algunas API resuelven directamente en Modo Usuario si pasar a Modo kernel para resolver el pedido.





Aplicación Invoca System Call

Accede a una rutina Wrapper en la librería standard ***libc***

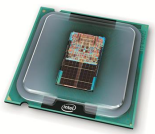
handler de System Call del kernel

rutina de servicio de la System Call

Rol de los segmentos en Modo IA-32e

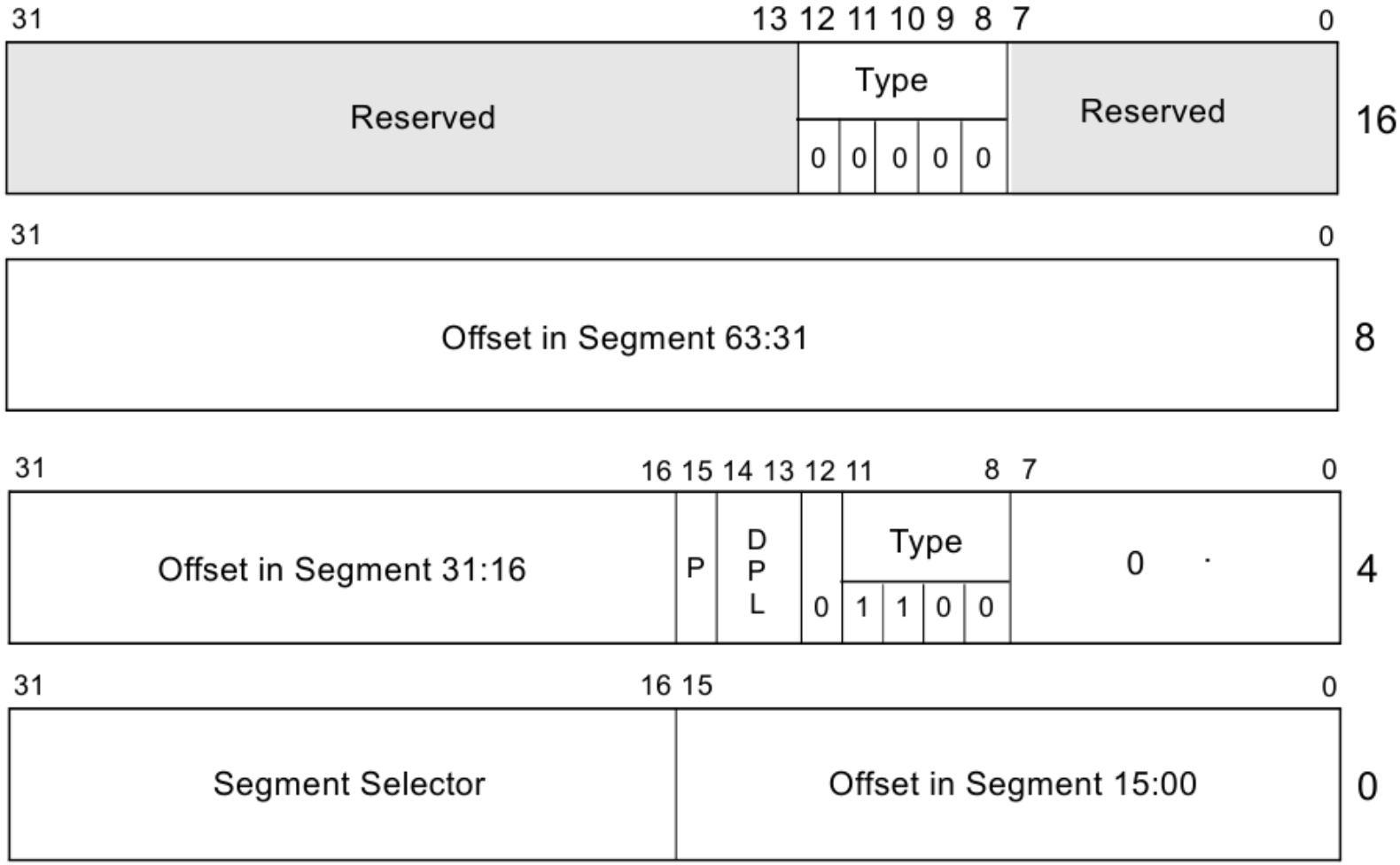
Si bien los segmentos en 64 bits están sumamente devaluados en el Modo IA-32e, caben algunas consideraciones, siempre asumiendo que IA-32_EFER.LME = 1:

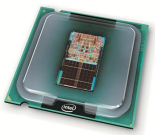
- Los segmentos en 64 bits se tratan con base 0 y no se chequea el campo límite del descriptor.
- En particular los segmentos de código mantienen algunas propiedades esenciales:
  1. El bit ***L*** de los atributos determina si el código que contiene este segmento se ejecuta en Modo 64 bits (***L*** = 1), o en modo compatibilidad (***L*** = 0).
  2. En caso de ***L*** = 0, el Bit de atributos ***D/B***, que está a continuación en los atributos determina, cuando vale '1', que el tamaño de datos y direcciones es de 32 bits, o cuando vale '0', que el tamaño de datos y direcciones es de 16 bits.
  3. En caso de ***L*** = 1, el Bit de atributos ***D/B***, debe valer '0', y el tamaño de datos default es 32 bits y el tamaño default de direcciones es 64 bits.
  4. la combinación ***L*** = 1, ***D/B*** = 1, está reservada para usos futuros y en la actualidad el procesador genera una regia #GP Fault si la detecta.
  5. El DPL del descriptor al igual que en el modo protegido legacy se utiliza para determinar el nivel de privilegio de ejecución.



- No se genera #GP Fault ante la carga o uso de un Null selector.

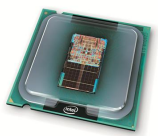
Puertas de llamada de 64 bits





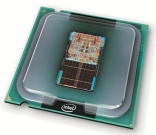
Puertas de llamada de 64 bits

- Se ha removido el campo “Parameter Count”
- Los 32 bits mas significativos de offset que se adosan en el descriptor deben estar en formato canónico, de otro modo se generará una excepción #GP.
- Este descriptor de 16 bytes coexiste en la GDT o LDT con cualquier otro descriptor de 8 bytes, inclusive con segmentos de código y datos de 16 y 32 bits que pueden usarse en modo compatibilidad. No obstante a los efectos del uso el descriptor de Call Gate de 64 bits ocupa dos lugares en la tabla de descriptores.
- El segmento de código destino debe ser de 64 bits. De otro modo se generará una excepción #GP.



Conmutación de stack

- En Modo 64 bits cuando se invoca por ejemplo una call gate y se cambia el nivel de privilegio de ejecución no se carga el stack de la manera en que lo hace el procesador en el modo protegido legacy.
- No se carga un nuevo SS, sino que se lo fuerza a NULL, pero seteando el campo RPL al nivel de privilegio del código al que se accede. El RSP se busca en un nivel mas interno de la IST.
- El par de registros SS:RSP actuales se almacenan en la nueva pila, desde donde se recuperarán al ejecutar RETF.
- La pila en 64 bits almacena elementos de ese mismo tamaño.



32-bit Mode	IA-32e mode

Conmutación de stack

- En el Modo 64 bits, RETF puede cargar un SS NULL bajo ciertas condiciones.
- Si se retorna a un código de 64 bits con CPL!= 3, SS puede cargar un selector Nulo
- Si el procedimiento llamado a su vez es interrumpido, el valor de SS que se guarda en el stack es NULL. De este modo al generarse RETF el selector Nulo le indica al procesador que no es necesario cargar un nuevo descriptor de SS.