

Resumen Practicas Orga 2

Booteo

Pasos de boot

1. El CPU comienza a ejecutar el BIOS (Basic Input Output System), que consiste de una memoria ROM en el mother con las primeras instrucciones para el CPU
2. El BIOS se encarga de correr una serie de diagnósticos llamados POST (Power On Self Test)
3. Busca un dispositivo "booteable" es decir, que en su sector de booteo los últimos dos bytes tengan la firma 0x55 y 0xAA respectivamente.
4. Se copia a memoria a partir de la dirección 0x7C00, el sector de booteo

Bootloader

510 bytes en la memoria auxiliar	En orga 2...
<ol style="list-style-type: none">1. Determinar el 'disco' y la partición a bootear2. Determinar donde esta la imagen del kernel en ese 'disco'3. Cargar la imagen del kernel en memoria4. Correr el "kernel"<ol style="list-style-type: none">1. Pasar a modo protegido2. Preparar las estructuras para administrar la memoria3. Preparar las estructuras del sistema	<ol style="list-style-type: none">1. Se copia el Bootloader en la posición 0x1000 de la memoria2. Se busca el archivo KERNEL.BIN en el diskette3. Se copia ese archivo en la posición 0x1200 de la memoria4. Se salta y se ejecuta la instrucción en la posición 0x1000 de la memoria

Modo Real	Direccionamiento a 16 bits: Modos						
<ol style="list-style-type: none">1. No hay protección de memoria2. AX, CX y DX no son de propósito general, no se pueden usar para acceder a memoria3. Podemos usar la BIOS, sus rutinas de acceso a dispositivos (por ejemplo, para imprimir por pantalla)4. Tenemos Registros de Segmento (CS, DS, SS, ...)	<table border="1"><tr><td>[BX + val] [val]</td><td>[BX + SI + val] [SI + val]</td></tr><tr><td>[BX + DI + val] [DI + val]</td><td>[BP + SI + val] [BP + val]</td></tr><tr><td>[BP + DI + val]</td><td></td></tr></table> <p>Cada dirección de memoria esta definida por un segmento y un offset (de 16 bits cada uno)</p> <p>0x : 0x segmento offset</p> <p>La forma de calcular a que dirección física que corresponde es: (segmento << 4) + offset</p>	[BX + val] [val]	[BX + SI + val] [SI + val]	[BX + DI + val] [DI + val]	[BP + SI + val] [BP + val]	[BP + DI + val]	
[BX + val] [val]	[BX + SI + val] [SI + val]						
[BX + DI + val] [DI + val]	[BP + SI + val] [BP + val]						
[BP + DI + val]							

Modo Real vs Modo Protegido

	Modo real	Modo protegido
<i>Memoria disponible</i>	1mb	4gb
<i>Privilegios</i>	nada	4 niveles de protección
<i>Manejo de interrupciones</i>	Rutinas de atención	Rutinas de atención con privilegios
<i>Acceso a instrucciones</i>	todas	Depende del nivel de protección

GDT: global descriptor table	LDT: local descriptor table
<p>Tabla en memoria donde cada entrada es de 8 bytes.</p> <p>Define alguno de los siguientes descriptores:</p> <ul style="list-style-type: none"> Descriptor de segmento de memoria (S=1) Descriptor de Task State Segment (TSS) (S=0) <ul style="list-style-type: none"> Guarda el estado de una tarea, sirve para intercambiar tareas Descriptor de call gate (S=0) <ul style="list-style-type: none"> Permite transferir control entre niveles de privilegios Actualmente no se usan en SO modernos Descriptor de LDT (S=0) <p>El primer descriptor de la tabla siempre es NULO</p>	<p>Tabla en memoria, igual que la GDT.</p> <p>Puede contener las mismas entradas que la GDT</p> <p>Se diferencia en:</p> <ul style="list-style-type: none"> La GDT tiene los descriptores globales y es única para todo el sistema. La LDT tiene los descriptores locales a una tarea y puede existir mas de una LDT en el sistema, una por cada tarea. <p>Tabla obsoleta por el uso del mecanismo de paginación</p>

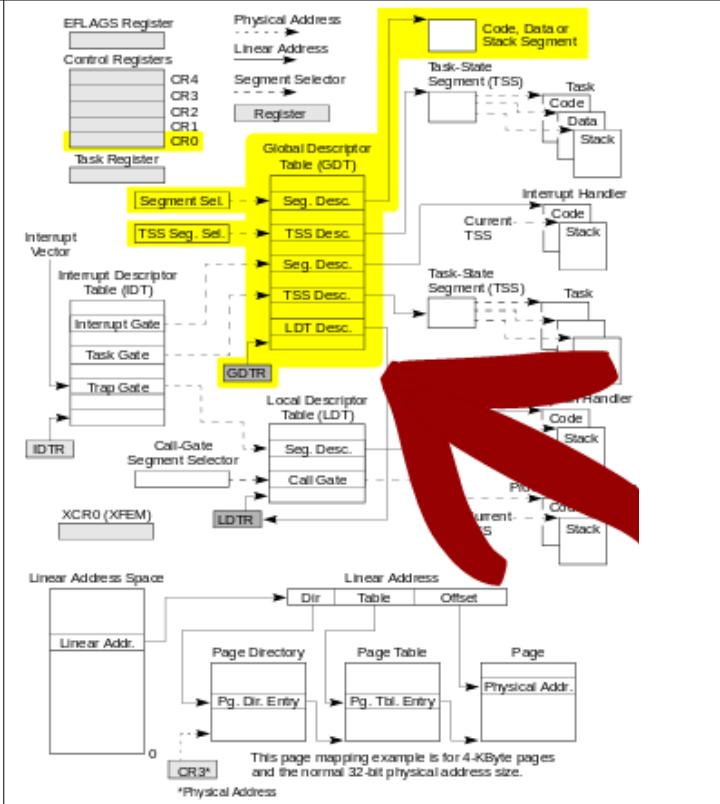
Unidad de segmentación	Selector de segmento
	<p>CS : Para acceder a código SS: Para acceder a pila DS: Para acceder a datos (default)</p> <p>ES: Para acceder a datos GS: Para acceder a datos FS: Para acceder a datos</p>

Descriptor de segmento (descriptor GDT)

	<ul style="list-style-type: none"> L — 64-bit code segment (IA-32e mode only) AVL — Available for use by system software BASE — Segment base address D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment) DPL — Descriptor privilege level G — Granularity LIMIT — Segment Limit P — Segment present S — Descriptor type (0 = system; 1 = code or data) TYPE — Segment type
--	--

Tipo de selector de segmento

Decimal	Type Field				Descriptor Type	Description	
	11	10	E	9	W	A	
0	0	0	0	0	0	Data	Read-Only
1	0	0	0	0	1	Data	Read-Only, accessed
2	0	0	1	0	0	Data	Read/Write
3	0	0	1	1	0	Data	Read/Write, accessed
4	0	1	0	0	0	Data	Read-Only, expand-down
5	0	1	0	0	1	Data	Read-Only, expand-down, accessed
6	0	1	1	0	0	Data	Read/Write, expand-down
7	0	1	1	1	0	Data	Read/Write, expand-down, accessed
	C R A						
8	1	0	0	0	0	Code	Execute-Only
9	1	0	0	0	1	Code	Execute-Only, accessed
10	1	0	1	0	0	Code	Execute/Read
11	1	0	1	1	0	Code	Execute/Read, accessed
12	1	1	0	0	0	Code	Execute-Only, conforming
13	1	1	0	1	0	Code	Execute-Only, conforming, accessed
14	1	1	1	0	0	Code	Execute/Read, conforming
15	1	1	1	1	0	Code	Execute/Read, conforming, accessed



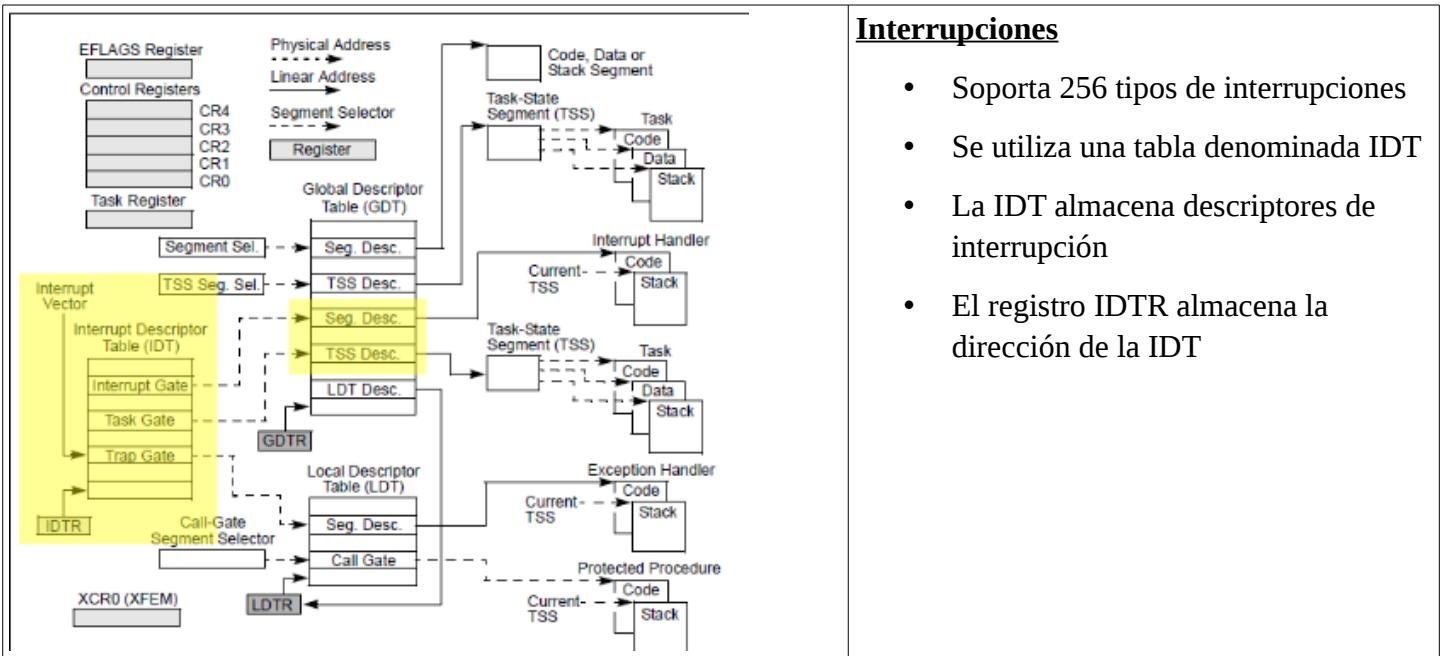
Pasar a modo protegido



Pasos:

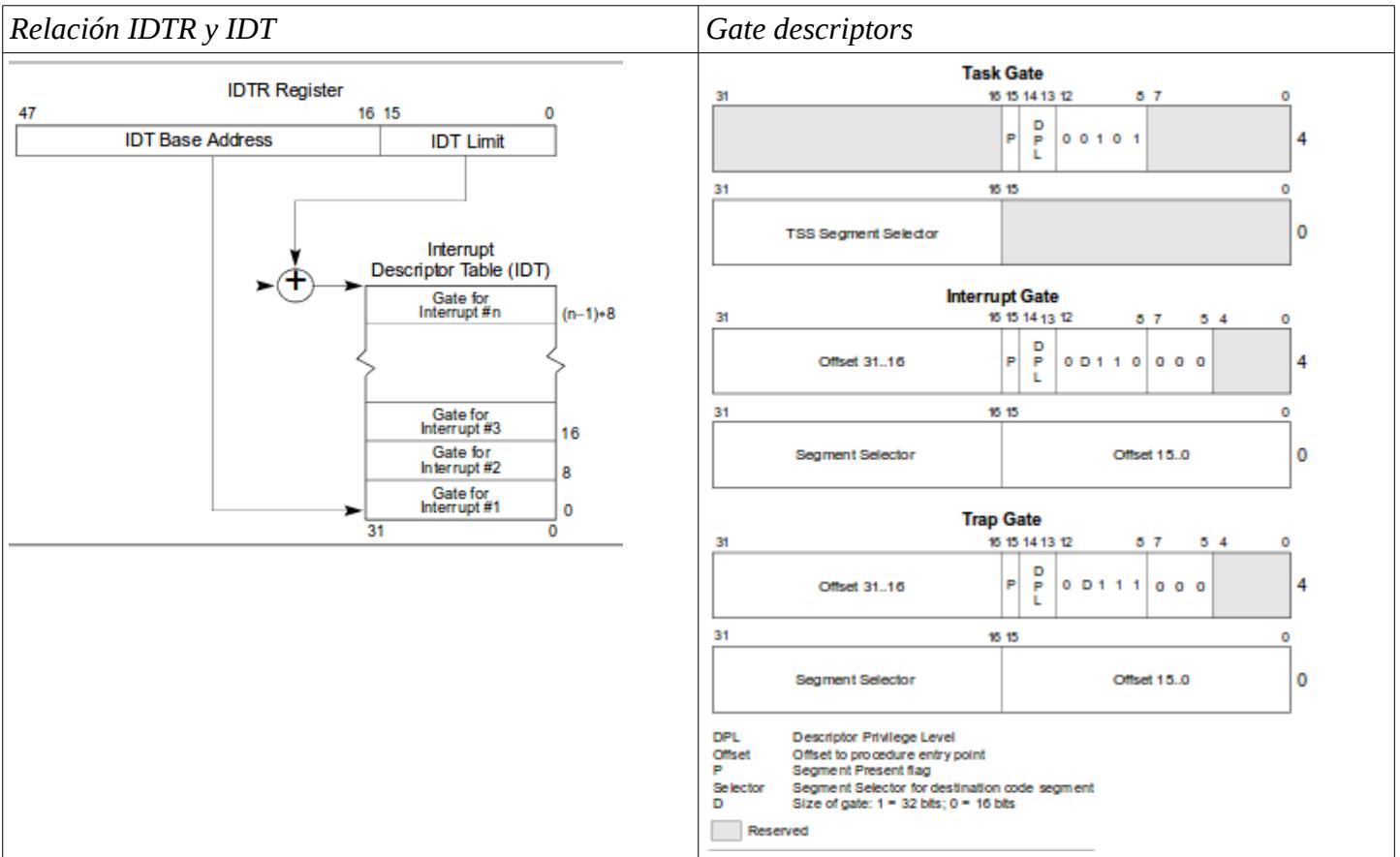
Completar la GDT

1. Deshabilitar interrupciones (CLI)
2. Cargar el registro GDTR con la dirección on base de la GDT:
 $LGDT <offset>$
3. Setear el bit PE del registro CR0
 $MOV eax,cr0$
 $OR eax,1$
 $MOV cr0,eax$
4. FAR JUMP a la siguiente instrucción
 $JMP <selector>:<offset>$
5. Cargar los registros de segmento (DS, ES, GS, FS y SS)



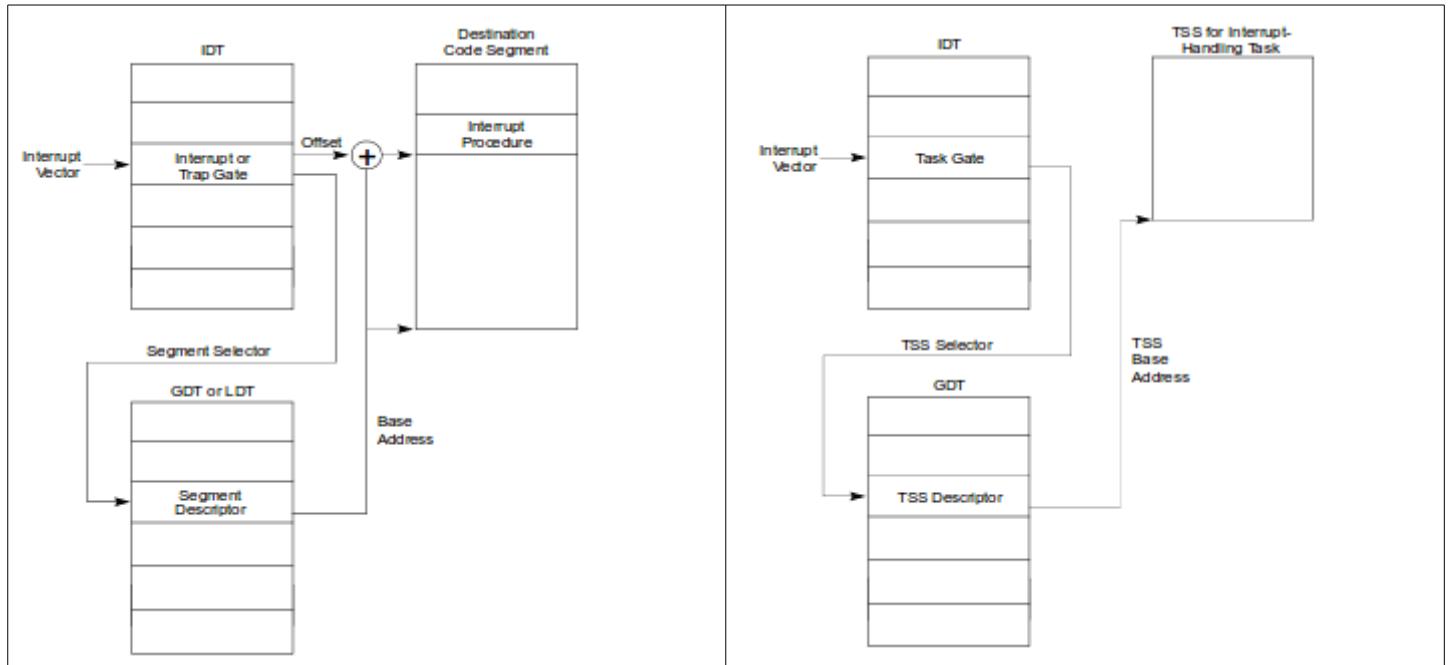
Interrupciones

- Soporta 256 tipos de interrupciones
- Se utiliza una tabla denominada IDT
- La IDT almacena descriptores de interrupción
- El registro IDTR almacena la dirección de la IDT



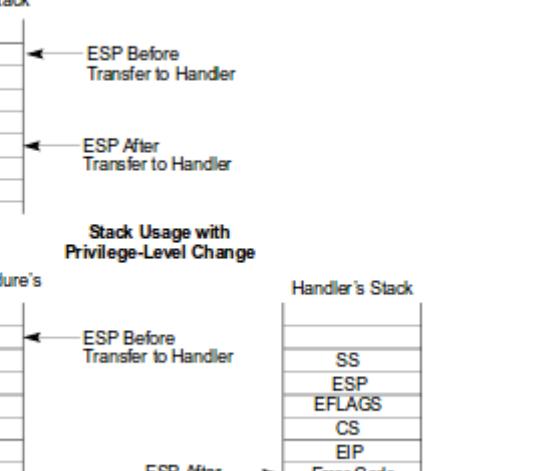
Interrupt procedure call

Interrupt task switch



Tipos de interrupciones

- **Fault:** Excepción que puede corregirse permitiendo al programa retomar la ejecución de esa instrucción sin perder continuidad. El procesador guarda en la pila la dirección de la instrucción que produjo la falla.
 - **Traps:** Excepción producida inmediatamente a continuación de una instrucción de trap. Algunas permiten al procesador retomar la ejecución sin perder continuidad. Otras no. El procesador guarda en la pila la dirección de la instrucción a ejecutarse luego de la instrucción trapeada.
 - **Aborts:** Excepción que no siempre puede determinar la instrucción que la causó, ni permite recuperar la ejecución de la tarea que la causó. Reporta errores severos de hardware o inconsistencias en tablas del sistema.

Stack	Error code			
<p>Stack Usage with No Privilege-Level Change</p>  <p>Interrupted Procedure's Stack</p> <p>Handler's Stack</p> <p>Stack Usage with No Privilege-Level Change</p> <p>ESP Before Transfer to Handler</p> <p>ESP After Transfer to Handler</p> <p>ESP Before Transfer to Handler</p> <p>ESP After Transfer to Handler</p>	<p>31 3 2 1 0</p> <table border="1"> <tr> <td data-bbox="847 1354 1079 1390">Reserved</td> <td data-bbox="1079 1354 1310 1390">Segment Selector Index</td> <td data-bbox="1310 1354 1426 1390">T I D T E X T</td> </tr> </table> <p>EXT: (External Event) Se setea para indicar que la excepción ha sido causada por un evento externo al procesador</p> <p>IDT: (Descriptor Location) Cuando está seteado indica que el campo Segment Selector Index se refiere a un descriptor de puerta en la IDT. Cuando está en cero indica que dicho campo se refiere a un descriptor en la GDT o en la LDT de la tarea actual.</p> <p>TI: (GDT/LDT) Tiene significado cuando el bit anterior esta en cero. Indica a que tabla de descriptores corresponde el selector del campo Indice. (GDT=0 , LDT=1)</p>	Reserved	Segment Selector Index	T I D T E X T
Reserved	Segment Selector Index	T I D T E X T		

Ayudas *IDT*

<p>idt.h: Descripción de las estructuras.</p> <p>idt.c: Estructura de la IDT con cada una de sus entradas</p> <p>isr.h: Descripción de las funciones handlers</p> <p>isr.asm: Código de los handlers</p> <p>IDT_ENTRY(numero): Permite declarar una entrada en la IDT, para la utilizar el handler de nombre _isrX</p> <p>void inicializar_idt(): Función llamada desde el kernel para inicializar las entradas en la idt</p>	<p><u>Struct de descriptor de IDT</u></p> <pre>typedef struct str_idt_descriptor { unsigned short idt_length; unsigned int idt_addr; } __attribute__((__packed__)) idt_descriptor;</pre> <p><u>Struct de una entrada de la IDT</u></p> <pre>typedef struct str_idt_entry_fld { unsigned short offset_0_15; unsigned short segsel; unsigned short attr; unsigned short offset_16_31; } __attribute__((__packed__, aligned (8))) idt_entry;</pre>
---	---

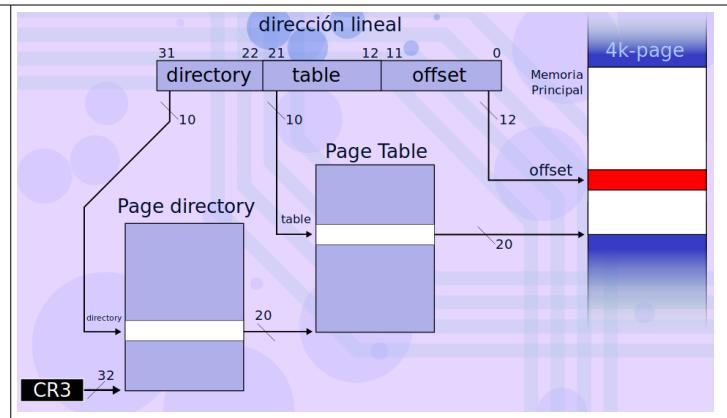
Interrupciones externas (enmascarables)	PIC
<p><u>Como las manejamos?</u></p> <p>Igual que manejamos las excepciones:</p> <ol style="list-style-type: none"> 1. Se definen las rutinas de atención para cada interrupción. <i>void _isrXX()</i> 2. Se declaran en la IDT.. <i>CALL clase anterior</i> 3. Se remapea el PIC. 4. Se activan las interrupciones. Se activa el flag IF del registro EFLAGS 	<ul style="list-style-type: none"> • Es un chip al que le llegan las interrupciones de los demás dispositivos de la máquina. • Administra las interrupciones por prioridad y las manda al procesador • El PIC puede atender 15 interrupciones, y las interrupciones se pisan. Hay un conflicto, y para solucionarlo hay que “remapearlas” a partir de la primera interrupción no reservada. <p>pic.h: resetear_pic (remapeo), habilitar_pic y deshabilitar_pic.</p>

Rutinas de atención	Unidad de paginación
<ol style="list-style-type: none"> 1. Preservar los registros que vayamos a romper 2. Comunicar al PIC que ya se atendió la interrupción (EOI) 3. Realizar la tarea correspondiente a la interrupción. 4. Restaurar los registros. 5. Retornar de la interrupción. (iret) <p>NOTA: No es necesario deshabilitar las interrupciones al comienzo de la rutina, ya que el procesador lo hace</p>	<p><u>Unidades de administración de memoria</u></p> <p><u>Mecanismo de paginacion</u></p>

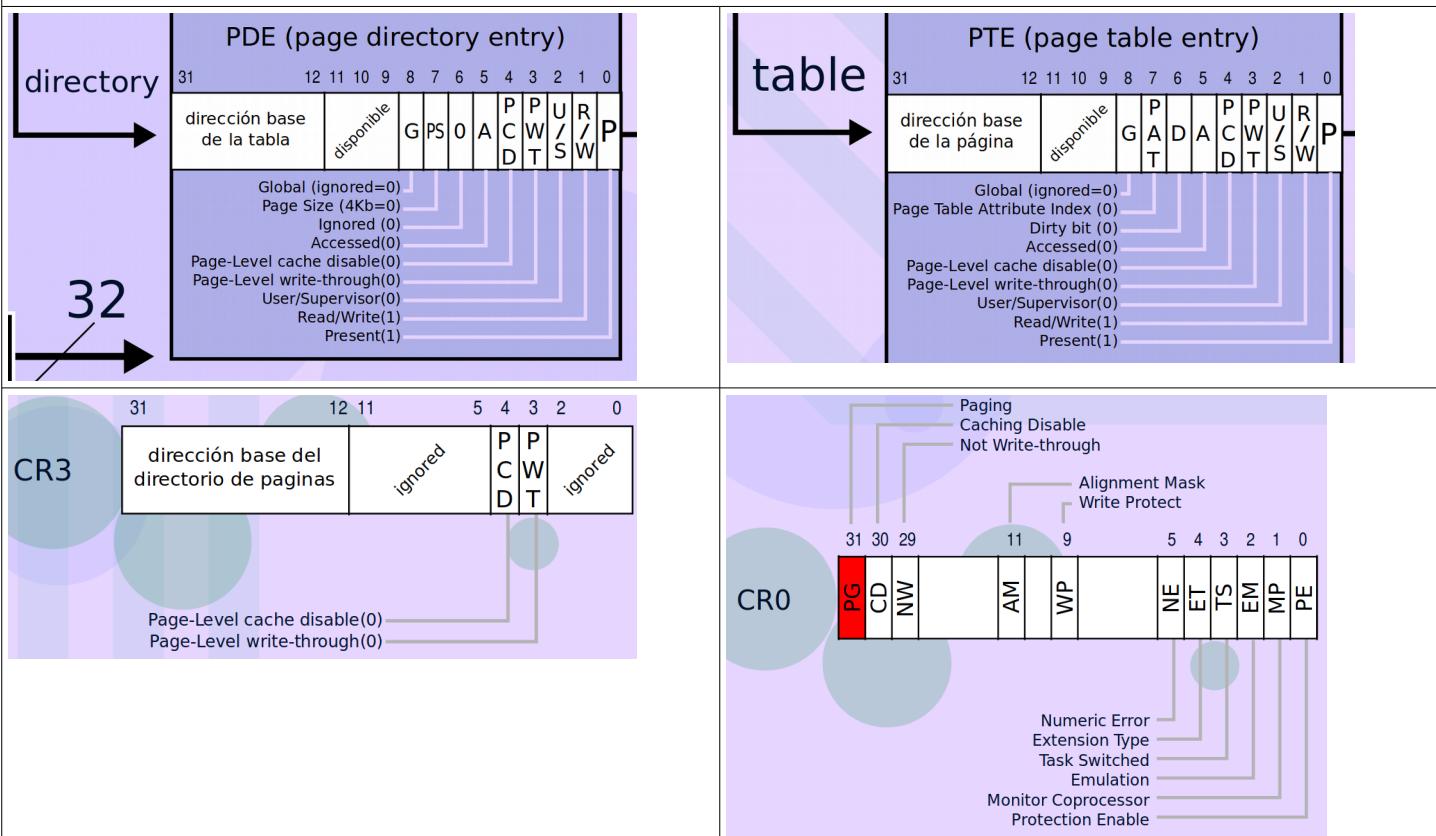
automáticamente (si definimos el descriptor como un interrupt gate). Tampoco es necesario volver a habilitarlas al finalizar.

Rutinas de atención del teclado

Leemos del teclado a través del puerto 0x60 (in al, 0x60) y obtenemos un scan code.



Paginación (cont.)



Ejemplo

Queremos resolver la dirección lineal 0x4a125515

directory	table	offset
31 0100101000	22 21 0100100101	12 11 010100010101
0x128	0x125	0x515

Pasos del CPU:

- Buscar el CR3

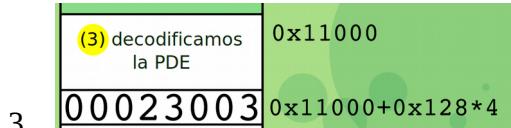
CR3
00011000

- Buscar el CR3
- Buscamos una entrada dentro del page directory (directory = 128)

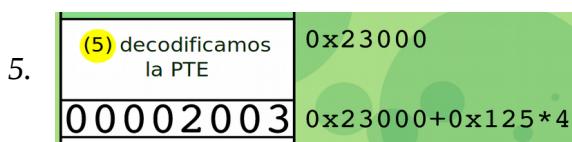
Pasos de paginación

- Armar un directorio de páginas y tablas de páginas
- Poner en CR3 la dirección base del directorio de páginas
- Limpiar bits PCD y PWT de CR3
- Setear el bit PG de CR0

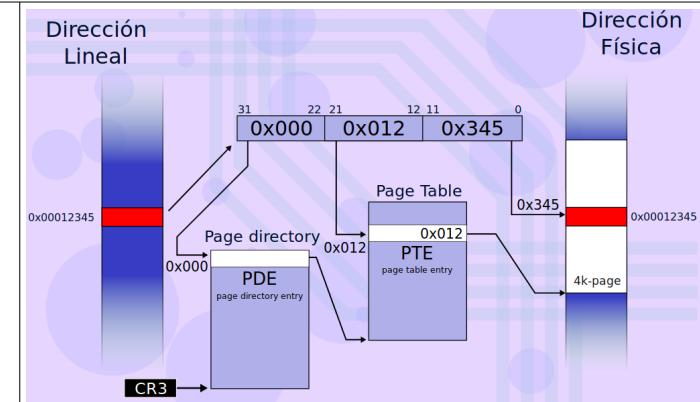
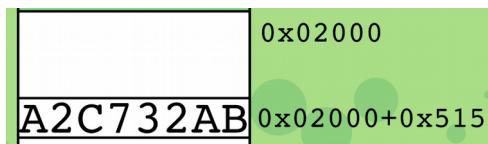
Identity mapping



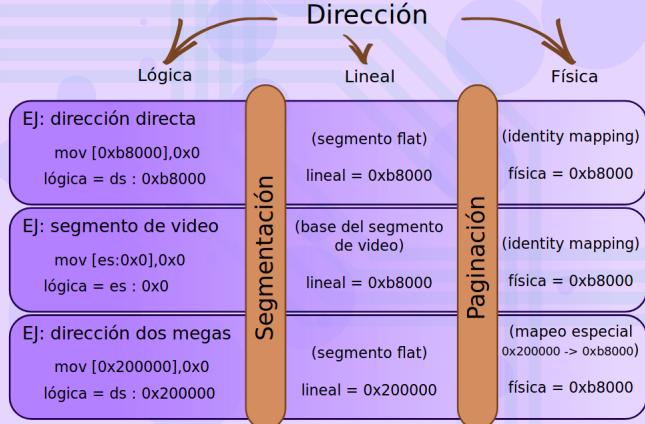
4. Buscamos una entrada dentro del page table (table = 125)



6. Buscamos la entrada de memoria que queremos (offset = 515)



Resolver direcciones - ejemplos



Memory management unit (MMU)

Obtención de páginas físicas libres

- Vamos a tener un entero que guarde la dirección de una página física libre.
- Cada vez que querremos una nueva página física libre, la obtenemos desde nuestro entero y luego incrementamos el mismo.
- Necesitamos las funciones para pedir las páginas y un entero global.

Desmappeo de páginas

Tenemos que hacer una función que desmappee una dirección virtual

Necesitamos:

- La dirección virtual
- La dirección en donde se encuentra el page directory donde vamos a hacer el desmappeo.

Procedimiento:

- Descomponemos la dirección virtual en índice del page directory y en índice del page table
- Utilizar la dirección del page directory y el índice del page directory para encontrar el page directory entry asociado al índice.
- Utilizar el índice del page table para obtener la page table entry correspondiente.
- Establecer el bit de presente en 0.

Mappeo de páginas

Tenemos que hacer una función que mappee páginas de una dirección virtual a otra física

Necesitamos:

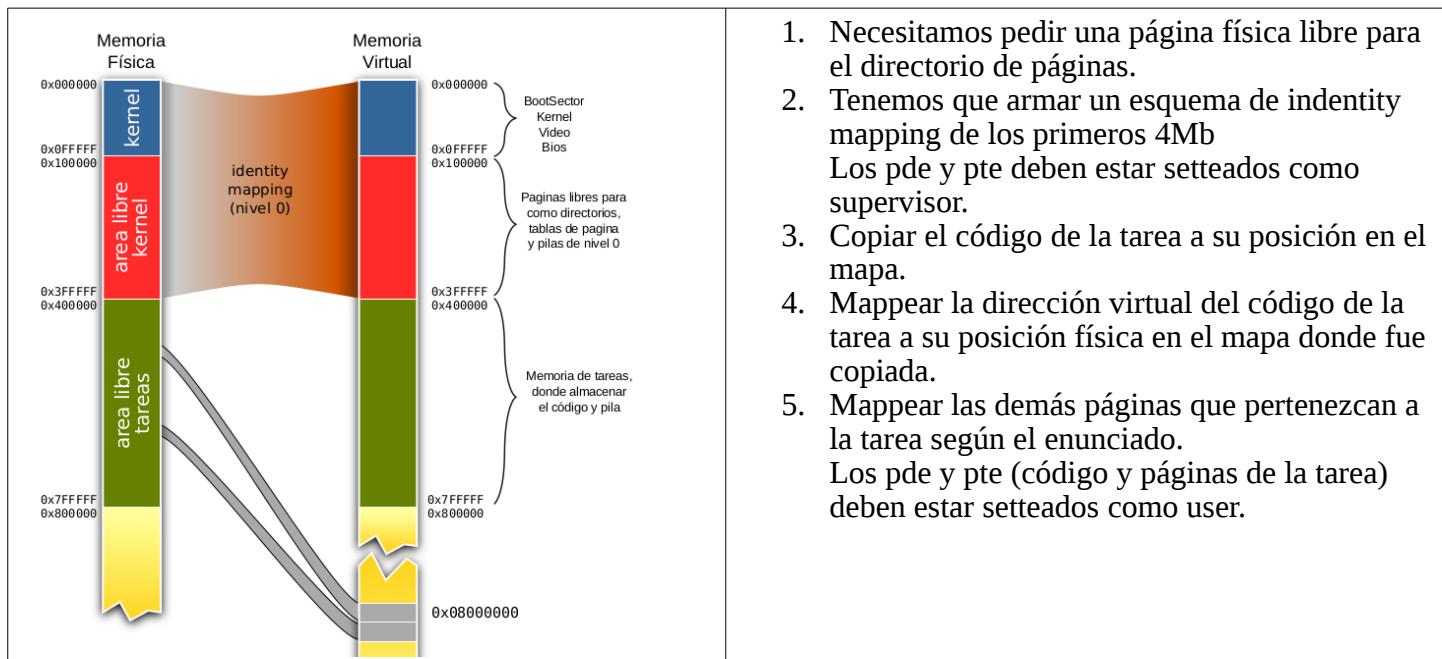
- La dirección virtual.
- La dirección en donde se encuentra el page directory donde vamos a hacer el mappeo.
- La dirección física

Procedimiento:

- Descomponemos la dirección virtual en índice del page directory y en índice del page table.
- Utilizar la dirección del page directory y el índice del page directory para encontrar el page directory entry asociado al índice.
 - Hay que chequear que la page table a la que referencia el pde exista.
 - Si no existe, hay que crearla y settear correctamente (bits de propiedades) la pde para que pueda ser accedida posteriormente.

3. Utilizar el indice del page table para obtener la page table entry correspondiente.
4. Completar la pte para que refiera a la dirección física.
5. Ejecutar tlbflush() para invalidar la cache de traducciones.

Construcción de esquema de paginación de tareas



Manejo de Tareas/Tasks

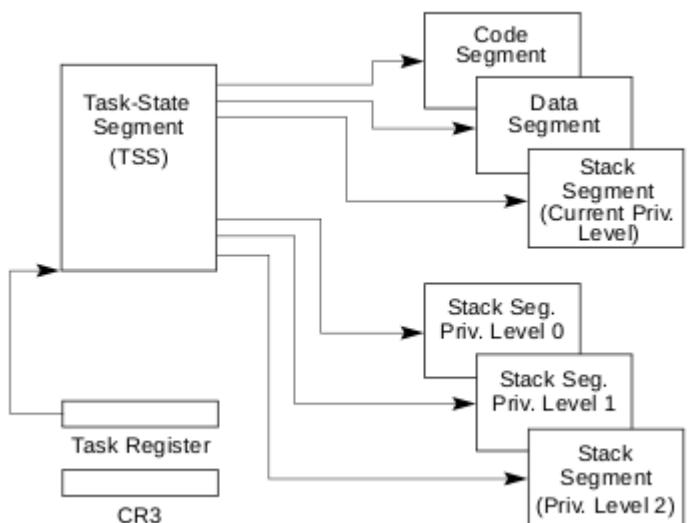
Tarea/task

Es una unidad de trabajo que el procesador puede despachar, ejecutar y suspender. La arquitectura provee mecanismos para salvar el estado de una tarea, para despachar una tarea para su ejecución y para conmutar tareas.

Esta compuesta por:

- Espacio de ejecución:
 - Segmento de código.
 - Segmento de datos/pila (uno o varios).
- Segmento de estado (TSS):
 - Almacena el estado de la tarea (su contexto) para poder reanudarla desde el mismo lugar.

Estructura de una tarea



Identificación de tareas

- Una tarea está identificada por el selector de segmento de su TSS.
- La TSS es un segmento. Debe estar descripto en la GDT del mismo modo que se describen los segmentos de código y datos.
- Además, el selector de segmento de la tarea que se está ejecutando actualmente se encuentra en el registro Task Register (TR).

TSS Descriptor

TSS Descriptor															
31	24	23	22	21	20	19	16	15	14	13	12	11	8	7	0
Base 31:24	G	0	0	A	V	L	Limit 19:16	P	D	P	L	Type	Base 23:16	4	
31	16 15												0		
	Base Address 15:00						Segment Limit 15:00								

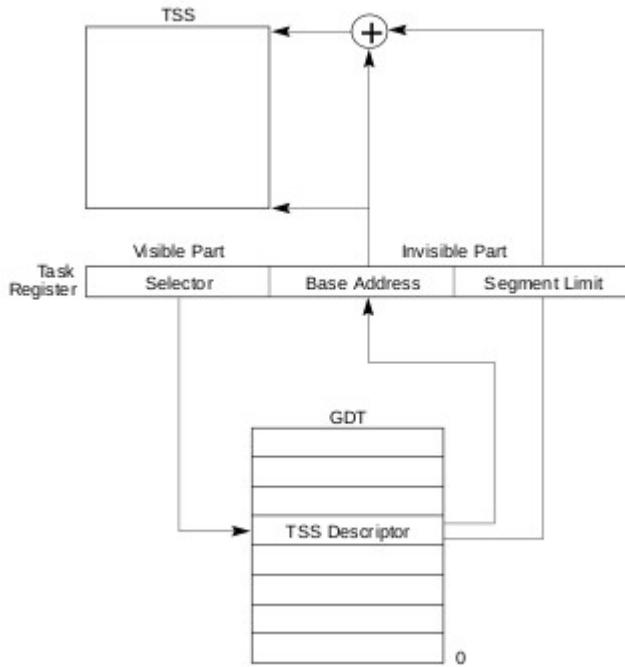
AVL Available for use by system software
 B Busy flag
 BASE Segment Base Address
 DPL Descriptor Privilege Level
 G Granularity
 LIMIT Segment Limit
 P Segment Present
 TYPE Segment Type

TSS (Task State Segment)

31	15	0
I/O Map Base Address	Reserved	T
Reserved	LDT Segment Selector	100
Reserved	GS	92
Reserved	FS	88
Reserved	DS	84
Reserved	SS	80
Reserved	CS	76
Reserved	ES	72
	EDI	68
	ESI	64
	EBP	60
	ESP	56
	EBX	52
	EDX	48
	ECX	44
	EAX	40
	EFLAGS	36
	EIP	32
	CR3 (PDOR)	28
Reserved	SS2	24
	ESP2	20
Reserved	SS1	16
	ESP1	12
Reserved	SS0	8
	ESP0	4
Reserved	Previous Task Link	0

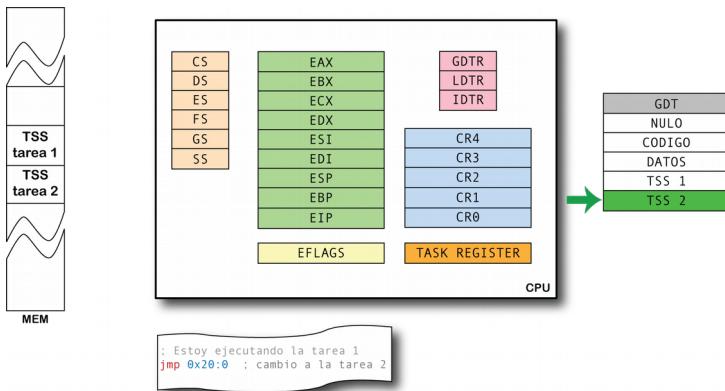
■ Reserved bits. Set to 0.

Task register

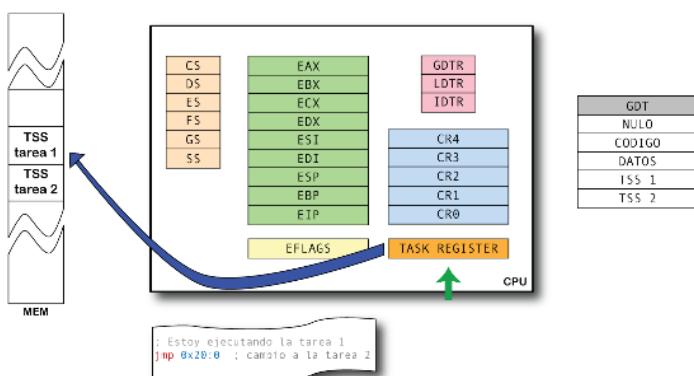


Conmutacion de tareas

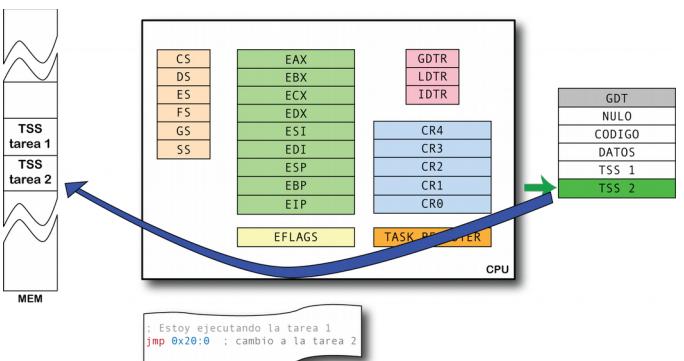
1. Ejecutamos la instrucción jmp 0x20:0
2. Se busca el descriptor correspondiente en la GDT.



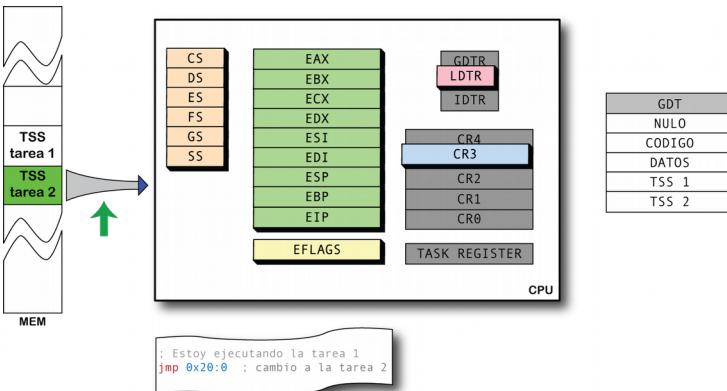
3. Como es un cambio de tarea, se lee el TR.
4. Se busca el TSS apuntado por el TR.



5. Se guarda el contexto actual.
6. Se busca TSS apuntado por descriptor de tarea a ejecutar.

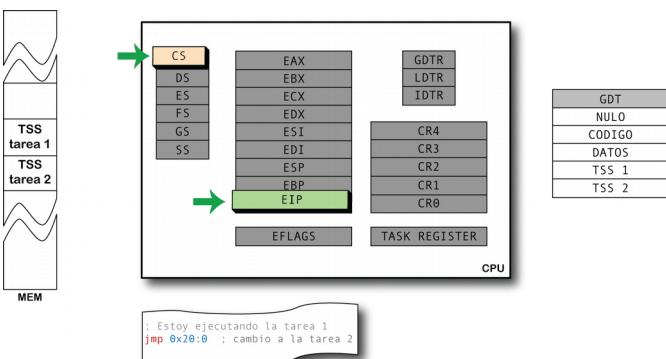


7. Se obtiene el nuevo contexto.



8. Se actualiza el TR.

9. Se continua la ejecución con el nuevo contexto.



Conmutación para la tarea inicial

Siempre que se salta a una tarea, hay un cambio de contexto. El procesador guarda el contexto actual de la tarea (identificado en TR) y carga el contexto de la tarea a la cual se está saltando.

Hay que crear una tarea inicial para proveer una TSS en donde el procesador pueda guardar el contexto actual. Esta tarea inicial tiene este único propósito.

Checklist:

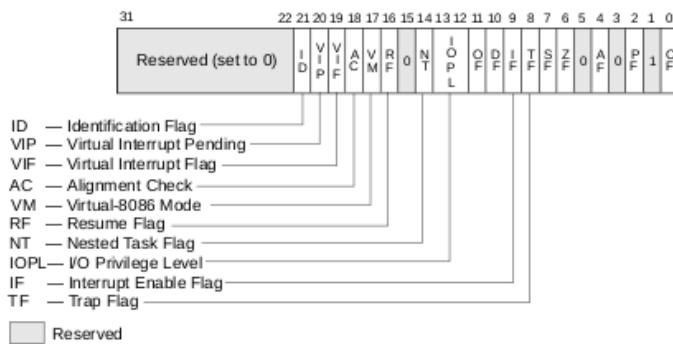
Para iniciar tareas, completar:

- EIP
- ESP y EBP
- selectores de segmento
- CR3
- EFLAGS

Al saltar por primera vez a una tarea:

- tener un descriptor en la GDT de la tarea inicial
- tener un descriptor en la GDT de la tarea a saltar
- tener en TR algun valor válido (tarea inicial)

Eflags



Chequeo de permisos/privilegios/protección

Accediendo a direcciones virtuales con paginación y segmentación:

- ¿Cómo se resuelven las direcciones virtuales?



- Estructuras necesarias para segmentación:

GDT / LDT

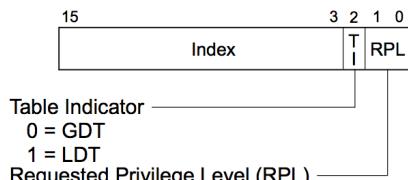
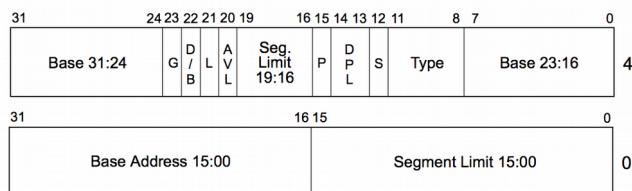
- Estructuras necesarias para paginación:

Page Directory (PD) y Page Table (PT).

Selector de segmento

- ¿Cómo sé qué segmento usar?

Todo acceso a memoria está compuesto por:
→ (selector-segmento : offset)



Legend:

- L — 64-bit code segment (IA-32e mode only)
- AVL — Available for use by system software
- BASE — Segment base address
- D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
- DPL — Descriptor privilege level
- G — Granularity
- LIMIT — Segment Limit
- P — Segment present
- S — Descriptor type (0 = system; 1 = code or data)
- TYPE — Segment type

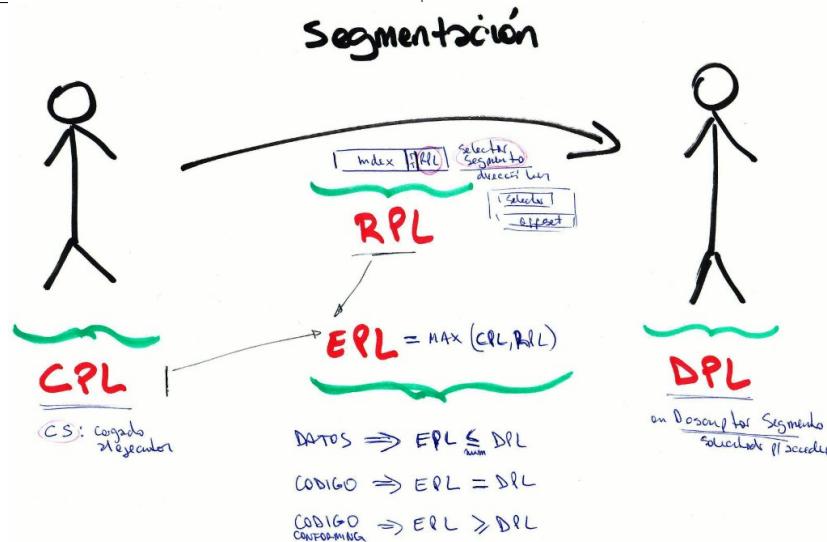
Cosas a chequear en el GDT descriptor para segmentación:

- P: tiene que estar presente la entrada
- Límite:
 - G = 0: $offset + bytes\ a\ leer - 1 \leq limite$
 - G = 1: alguno de los siguientes
 - $offset + bytes\ a\ leer - 1 \leq ((limite + 1) * 4\ kb) - 1$
 - $offset + bytes\ a\ leer - 1 \leq ((limite + 1) \ll 12) - 1$
 - $offset + bytes\ a\ leer - 1 \leq (limite \ll 12) + 0\ xFFF$
- Chequear permisos:

- Si es un segmento de datos: $EPL \leq DPL$.
- Si es un segmento de código:
 - Non-conforming: $EPL = DPL$
 - Conforming: $CPL \geq DPL$ (Ojo: no se usa EPL)

Definiciones:

- CPL: Current privilege level. El CPL es el DPL del segmento de código que estoy ejecutando
- EPL: Effective Privilege Level. $EPL = \max(CPL, RPL)$



- El tipo de entrada es correcto? (S)
 - Si quiero leer, escribir o ejecutar y $S = 0 \rightarrow$ Error.
 - Quiero hacer una lectura
 - Código con lectura inhabilitada? \rightarrow Error.
 - Código con lectura habilitada? \rightarrow Bien
 - Datos? \rightarrow Bien
 - Estoy realizando una escritura
 - Código? \rightarrow Error
 - Datos con escritura inhabilitada? \rightarrow Error
 - Datos con escritura habilitada? \rightarrow Bien
 - Estoy ejecutando código
 - Datos? \rightarrow Error
 - Código? \rightarrow Bien.

Type Field					Descriptor Type	Description							
Decimal	11	10 E	9 W	8 A									
0	0	0	0	0	Data	Read-Only							
1	0	0	0	1	Data	Read-Only, accessed							
2	0	0	1	0	Data	Read/Write							
3	0	0	1	1	Data	Read/Write, accessed							
4	0	1	0	0	Data	Read-Only, expand-down							
5	0	1	0	1	Data	Read-Only, expand-down, accessed							
6	0	1	1	0	Data	Read/Write, expand-down							
7	0	1	1	1	Data	Read/Write, expand-down, accessed							
		C	R	A									
8	1	0	0	0	Code	Execute-Only							
9	1	0	0	1	Code	Execute-Only, accessed							
10	1	0	1	0	Code	Execute/Read							
11	1	0	1	1	Code	Execute/Read, accessed							
12	1	1	0	0	Code	Execute-Only, conforming							
13	1	1	0	1	Code	Execute-Only, conforming, accessed							
14	1	1	1	0	Code	Execute/Read, conforming							
15	1	1	1	1	Code	Execute/Read, conforming, accessed							

Paginacion

Chequeo de privilegios y atributos

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	
Address of 4KB page frame												Ignored	G	P A T	D	A	P C D	P W T	U / S	R / W	1	PTE: 4KB page										
Address of page table												Ignored	Q	I g n	A	P C D	P W T	U / S	R / W	1	PDE: page table											

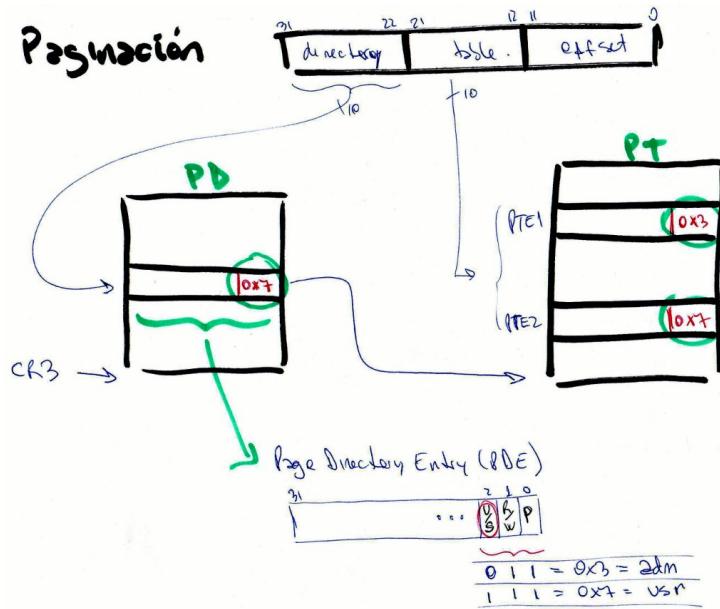
- ¿Están presentes las entradas del PD y PT? Chequeo bit de presente
- ¿Puedo leer y/o escribir? Bit R/W
 - 0 → Read Only
 - 1 → Read Write
- ¿Soy usuario o supervisor? Bit U/S.
 - 0 → Supervisor
 - 1 → User

El resultado del acceso se calcula en relación a los permisos de ambas tablas (PD y PT). Generalmente:

- Si la entrada del PD o PT tiene Supervisor → sólo es accesible por supervisor.
- Sólo si la entrada del PD y PT tiene User → es accesible por user.
- Sólo si ambas entradas tienen Read/Write → es posible escribir.

Ejemplo: PD con la que se debe acceder a 2 PTs con distintos privilegios cada una, una debería ser de usr y la otra de adm

Paginación



Interrupciones

Interrupt Gate		Trap Gate	
31	Offset 31..16	31	Offset 31..16
31	16 15	31	16 15
Segment Selector	Offset 15..0	Segment Selector	Offset 15..0
	0		0

Chequeo de privilegios en interrupciones

- Bit de presente: P = 1
- Selector de segmento: Determina qué entrada de la GDT será utilizada para correr la interrupción.
- DPL:
 - Nivel de privilegio necesario para llamar a la interrupción (con la instrucción int).
 - El CPL ≤ DPL de la interrupción.
 - En las interrupciones por hardware se ignora el DPL.

Intercambio de tareas

TSS Descriptor										Descriptor Fields									
31	24	23	22	21	20	19	16	15	14	13	12	11	8	7	0	AVL	B	Available for use by system software	
Base 31:24	G	0	0	A	V	L	Limit 19:16	P	D	P	L	0	1	0	B	1	Base 23:16	4	
31	16	15																	0
Base Address 15:00																			0

Chequeo de privilegios

- Bit de presente: P = 1
- DPL: Sólo es posible saltar a una tarea si el CPL ≤ DPL.