Taller 4

1. Explorando el manual Intel Volumen 3: System Programming. Sección 2.2 Modes of Operation.
¿A qué nos referimos con modo real y con modo protegido en un procesador Intel?
¿Qué particularidades tiene cada modo?
Modo real: Este modo de operación provee el entorno de programación de un procesador Intel 8086. Además incluye unas extensiones que nos permiten pasar a modo protegido.
Modo protegido: Este es el modo nativo del procesador. Este modo nos provee acceso a un conjunto de opciones de arquitectura ,flexibilidad, alto rendimiento y *backward compatibility* con el software base.

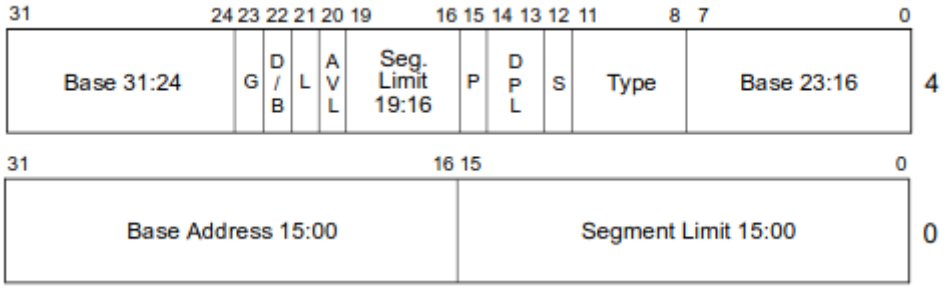
2. Comenten en su equipo,
¿Por qué debemos hacer el pasaje de modo real a modo protegido?
Es importante para poder operar correctamente con el procesador y sobre el sistema.
Es importante notar que el modo protegido es el modo nativo de funcionamiento del procesador.
¿No podríamos simplemente tener un sistema operativo en modo real?
Sí podríamos tener un S.O. en modo real, pero no es recomendable por la cantidad de limitaciones.

¿Qué desventajas tendría?
No podemos usar nada del modo protegido y otras cosas más. Ej.: 1MB de direccionamiento, no hay privilegios, seguridad, etc. En la siguiente tabla se pueden ver todas las desventajas del modo real frente al modo protegido.

	Modo real	Modo Protegido
Memoria Disponible	$\sim$ 1 MB	4 GB
Privilegios	:(	4 niveles de protección
Interrupciones	Rutinas de atención	Rutinas de atención con privilegios
Set de instrucciones	Todas	Depende el nivel de privilegio

3. Busquen el manual volumen 3 de Intel en la sección 3.4.5 Segment Descriptors.
¿Qué es la GDT?
La GDT o Global Descriptor Table. Es la tabla que contiene a los descriptores de segmento los cuales nos permiten movernos por el espacio en memoria de los segmentos que tengamos disponibles para manejar tanto datos como código de manera global.

¿Cómo es el formato de un descriptor de segmento, bit a bit?
El formato de un descriptor de segmento bit a bit es el siguiente:



Expliquen para qué sirven los campos Limit, Base, G, P, DPL, S. También puede referirse a la teórica slide 30. Manejo de memoria

- **Dirección base:** Es la dirección física a partir de la cual se despliega el segmento.
- **Límite:** Especifica el desplazamiento máximo offset que puede tomar un registro de desplazamiento en ese segmento. Si el valor que tiene ese registro supera el límite, hay Segmentation Fault.
- **G:** Bit 23: Granularidad. Si es 1 se expresa el máximo desplazamiento en 4K, si es 0 se expresa el máximo límite de 1 byte dentro del segmento.
- **P:** Bit 15. Presente. Si el segmento está presente o no en la memoria. Presente = 1, No Presente = 0. Esto significa que el segmento está en la RAM o no. El aviso para poder hacer el swap de disco a memoria es una interrupción del S.O. y el hardware.
- **DPL:** Bit 13-14. Descriptor Privilege Level. Le indico al descriptor el nivel de privilegio que tiene. Son 2 bits por ende son 4 privilegios. Van de Más a menos privilegiados.
- **S:** Bit 12. System. Define si el descriptor contiene código o datos o si es un descriptor de sistema.

4. La tabla de la sección 3.4.5.1 Code- and Data-Segment Descriptor Types del volumen 3 del manual del Intel nos permite completar el Type, los bits 11, 10, 9, 8. ¿Qué combinación de bits tendríamos que usar si queremos especificar un segmento para ejecución y lectura de código?

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

- **0|1:** Bit 11. Si S es 1 y este bit es 0, es un descriptor de sistema, es de datos, puede ser pila. Si es 1 tengo un segmento de código siempre que S sea 1. Si el descriptor es de código va abajo. C, R, si es 0 va ED, W
- **ED|C:** Bit 10. C = Conforme, ajustable. ED = Expand Down => El segmento decrece hacia direcciones numéricamente inferiores.
- **W|R:** Bit 9. R = Readable. Lo podemos poner en 0 o 1.
- **A:** Bit 8 . Accedido
    - Tanto P como A dan soporte a la memoria virtual. Algo virtual es algo que en un principio no existe pero funciona todo como si existiese.
    - Es un mecanismo de memoria para mostrarle al usuario un espacio de memoria mucho mayor del que tienen disponible.
    - Cuando se termina la RAM se usa la memoria del siguiente escalón para usarlo, esto es, del disco duro. Se toma y organiza para hacerlo en tiempo y forma.
    - El aviso para poder hacer el swap de disco a memoria es una interrupción del S.O. y el hardware.

Si queremos un segmento para ejecución es:

Bit 11 = 1
Bit 10 => C = 0
Bit 9 => R = 1
Bit 8 => A = 0

Ejercicio 5

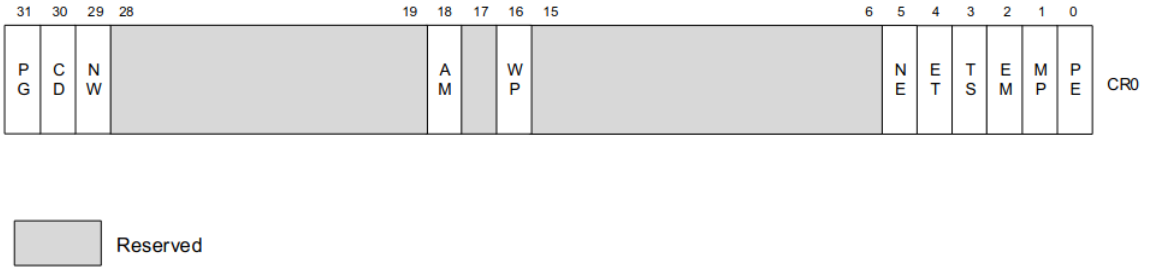
https://docs.google.com/spreadsheets/d/1ijpQq5vxm5ZWf3qUbPYv6CeYJNzWG5t1woTz5mYZhDI/edit?usp=sharing

6. En el archivo gdt.h observen las estructuras: struct gdt_descriptor_t y el struct gdt_entry_t
¿Qué creen que contiene la variable extern gdt_entry_t gdt[]; y extern gdt_descriptor_t GDT_DESC; ?
La variable gdt va a contener un arreglo de estructuras gdt_entry_t.
GDT_DESC es una variable que contiene una estructura gdt_descriptor_t

10. Busquen qué hace la instrucción LGDT en el Volumen 2 del manual de Intel. Expliquen con sus palabras para qué sirve esta instrucción. En el código, ¿qué estructura indica donde está almacenada la dirección desde la cual se carga la GDT y su tamaño? ¿dónde se inicializa en el código?

Esta instruccion carga en el registro gdtr el valor de 6 bytes que le pasemos, el cual va a contener el tamaño total de la gdt y la direccion de memoria donde empieza.
La estructura que almacena la dirección desde la cual se carga la GDT es el str_gdt_descriptor definido en gdt.h.
La mísma se inicializa en el gdt.c con la variable GDT_DESC

13. Investiguen en el manual de Intel sección 2.5 Control Registers, el registro CR0. ¿Qué formato tiene?
¿Qué hace el bit CR0.PE?
Para modificar CR0, no pueden hacerlo directamente. Sólo mediante un MOV desde/hacía los registros de control (pueden leerlo en el manual en la sección citada).
Es un registro de control de 32 bits. Tiene flags de control del sistema que controlan el modo de operación y el estado del procesador.



El bit 0 de CR0 es el *protection enable*.Cuando se setea establece el modo protegido.
Tener en cuenta que para establecer la paginación es necesario setear el bit 31 PG.

21. Observen el método screen_draw_box en screen.c y la estructura en screen.h . ¿Qué creen que hace el método screen_draw_box ? ¿Cómo hace para acceder a la pantalla? ¿Qué estructura usa para
representar cada carácter de la pantalla y cuanto ocupa en memoria?
El método se utiliza para escribir en la pantalla. Cada carácter ocupa 1 byte.
PREGUNTAR

El metodo screen_draw_box se supone que dibuja un cuadrado de tamaño fSize y cSize inicializado en posiciones fInit y cInit, el cuadrado es del tipo character y los atributos indican su color del caracter (Idea)

Taller 5

1)

a) Observen que la macro IDT_ENTRY0 corresponde a cada entrada de la IDT de nivel 0

¿A qué se refiere cada campo? ¿Qué valores toma el campo offset?

- ➔ **Offset** que va a ser la dirección de memoria donde comienza la rutina de atención de interrupción.
- ➔ **Segment selector** que indica qué selector debe utilizarse al ejecutar el código de la rutina.
- ➔ **P,DPL** que indican si la rutina se encuentra en memoria o no y el nivel de privilegio, respectivamente.
- ➔ **Bits 8 a 12 de los bytes 4 a 7** indican el tipo específico de la compuerta de interrupción, el bit D indica si es una compuerta de 32 o 16 bits.

ISR = Interrupt Service Routine
El campo offset va tomar la dirección de memoria de **ISR** concatenada con el número pasado por parámetro. Es decir, la dirección de memoria de la rutina de atención a la interrupción correspondiente al número dado.
(Entendemos que ## concatena)

b) Completar los campos de Selector de Segmento (segsel) y los atributos (attr) de manera que al usarse la macro defina una Interrupt Gate de nivel 0. Para el Selector de Segmento, recuerden que la rutina de atención de interrupción es un código que corre en el nivel del kernel.

¿Cuál sería un selector de segmento apropiado acorde a los índices definidos en la GDT[segsel]?

```
GDT_CODE_0_SEL
```

¿Y el valor de los atributos si usamos Gate Size de 32 bits?
Si utilizamos un gate size de 32 bits el bit d=1 .
Por lo tanto type = 0xE.

Prologo: pushad
Epilogo: popad
Iret

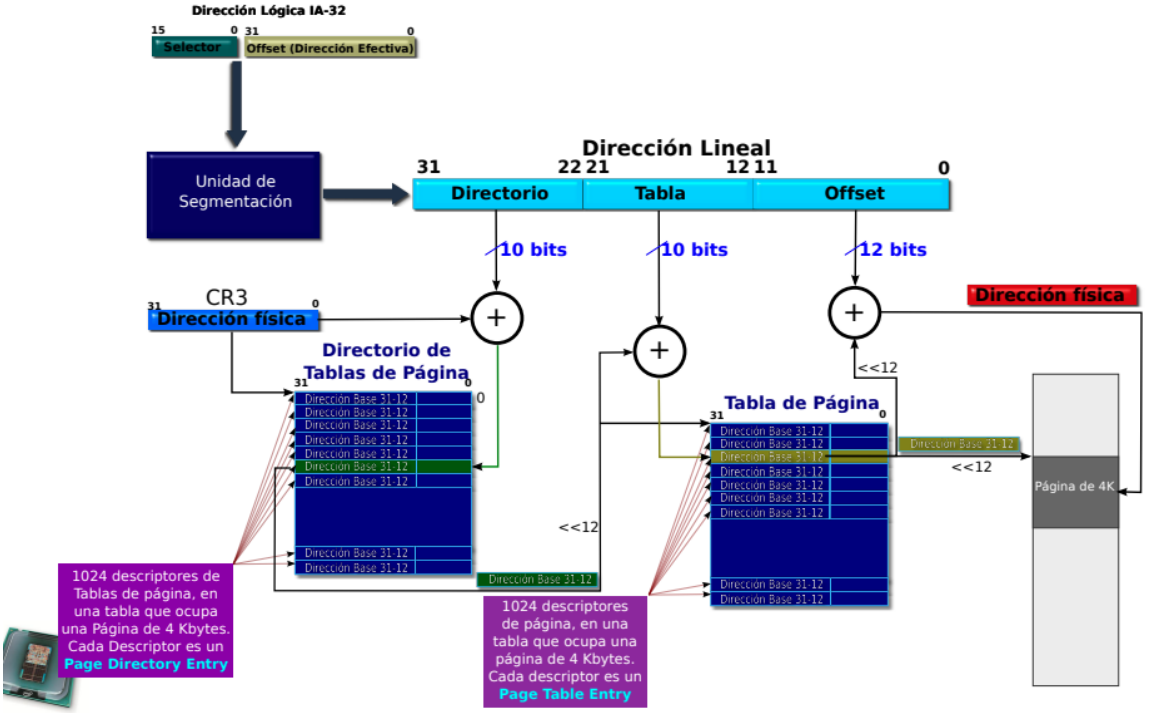
- Es necesario porque necesitamos sacar más cosas de la pila. Registros eflags, registro CS , registro eip y en algunas excepciones también un error code.
- Habilita las interrupciones. CLI

Taller 6: Paginación

Checkpoint 1

a) ¿Cuántos niveles de privilegio podemos definir en las estructuras de paginación?
Solo tenemos dos niveles de privilegio supervisor o
kernel (que se indica con el ‘0’) y usuarix (que se indica con el ‘1’).

b) ¿Cómo se traduce una dirección lógica en una dirección física?
¿Cómo participan el selector de segmento, el registro de control CR3, el directorio y la tabla de páginas?



Para cada tarea el procesador necesita conocer la dirección física en donde se inicia la estructura de Descriptores de página. En estos procesadores **el Registro de Control CR3 es el que contiene esta dirección (dirección física).**
Luego el procesador toma la **dirección lineal** y la ‘divide’ en tres campos de bits que serán utilizados de la siguiente forma:

1. Utilizo los 10 bits más significativos en conjunto con el CR3 para entrar en el directorio de tablas de página.
2. **Directorio** me dice cual de los 1024 directorios de tabla de página es. De ahí obtengo una dirección de memoria de 32 bits y descarta los 12 menos significativos.
3. Luego ese dir de memoria nos dice donde comienza la tabla de páginas donde uno de esos 1024 descriptores (de la tabla de páginas) es la página que buscamos.
4. Para encontrar el descriptor le suma los bits de tabla (del 12 al 21) a la dirección de memoria que obtuve en 3.
5. Luego al descriptor de página que obtuve le sumó el offset y obtengo la dirección física. Es decir, llegue al dato.

c) ¿Cuál es el efecto de los siguientes atributos en las entradas de la tabla de página?

- ➔ D: Dirty. Indica que la página ha sido modificada. El SO lo inicializa en '0'
- ➔ A: Accedido. Se setea cada vez que la página es accedida. Le sirve al SO para llevar contabilizar los accesos. Lo utiliza para llevar a cabo su política de desalojo LRU.
- ➔ PCD: Page-Level Cache Disable. Establece que una página integre el tipo de memoria no cacheable.
- ➔ PWT: Page-Level Write Through. Establece el modo de escritura que tendrá la página en el Cache.
- ➔ U/S: User / Supervisor. Establece si es usuario o supervisor (0 - mayor privilegio)
- ➔ R/W: Read / Write: SI se puede leer o escribir el segmento. 0 - writable
- ➔ P: Presente. Indica si la página está en memoria.

d) Suponiendo que el código de la tarea ocupa dos páginas y utilizaramos una página para la pila de la tarea.
¿Cuántas páginas hace falta pedir a la unidad de manejo de memoria para el directorio, tablas de páginas y la memoria de una tarea?
5 páginas. 1 por el PD , 1 por el PT y 3, 2 por las páginas de código y 1 por la pila.

f) ¿Qué es el buffer auxiliar de traducción (translation lookaside buffer o TLB) y por qué es necesario purgarlo (tlbflush) al introducir modificaciones a nuestras estructuras de paginación (directorio, tabla de páginas o CR3)?

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 caché de traducciones: el Translation Lookaside Buffer (o **TLB**)

- ➔ Es una memoria **caché pequeña** donde guardamos las traducciones.
- ➔ **Guardo SOLO el par directorio-tabla y lo pone al lado del descriptor de página que encontré** gracias a la dirección lineal (directorio,tabla y offset).Es decir, guarda la traducción.
- ➔ La idea es que primero busque en la TLB la página y si no lo encuentro hago todo los pasos de ir y buscarlo con en las tablas.
- ➔ Al ser una memoria tipo caché los accesos se hacen muy rápidos. La **política de desalojo en la TLB es LRU**. Para esto utiliza algunos bits de control.
- ➔ Es necesario purgarlo para poder tener la versión más actualizada de las páginas y, además, puede llegar a ocurrir que la TLB se llene por completo y nunca encuentre después las demás páginas que vaya a usar ocurriendo así una pérdida de acceso a las páginas.

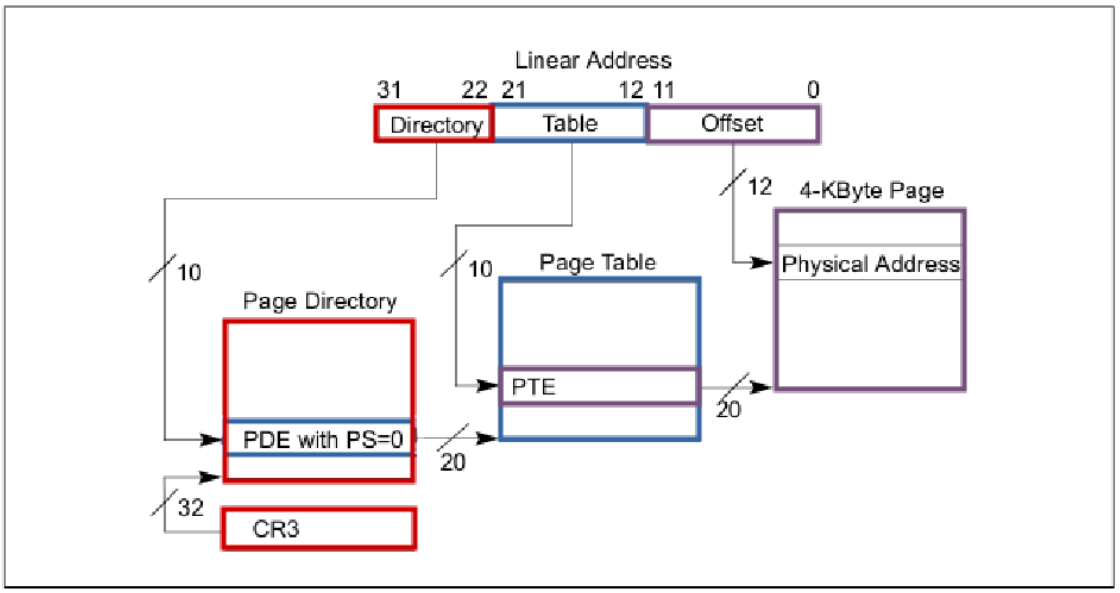


Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

Recuerden que el procesador cuenta con una tabla de traducciones pre-computadas a la que llamaremos buffer auxiliar de traducción o translation lookaside buffer, que básicamente almacena las últimas traducciones realizadas para no tener que volver a computarlas. Cuando realicemos un cambio en nuestras estructuras de paginación es necesario forzar una limpieza del mismo para evitar que las direcciones pre-computadas que ya no son válidas se sigan empleando, para esto realizamos un intercambio del registro CR3 con un valor temporal y luego lo restauramos.

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				PS	IGN	A	PCD	PWT	U/S	R/W	P

Figura: Descriptor de la Tabla de Páginas de 4 Kbytes

Taller 7: tareas

1. Si queremos definir un sistema que utilice sólo dos tareas, ¿qué estructuras, cantidad de nuevas entradas por estructura y registros tenemos que configurar? Indicar el detalle de los bits de cada una, ¿qué formato tienen? y ¿dónde se encuentran almacenadas dichas estructuras?

- ➔ Necesitamos configurar el registro TR. Para que tenga el selector de segmento de la primera tarea.
- ➔ Vamos a necesitar dos entradas en la gdt para poder cargar los descriptores de tss.
- ➔ Dos tss. Una para cada tarea.
- ➔ Cada una de 100 bytes.

2. ¿A qué llamamos cambio de contexto, cuándo se produce y qué efecto tiene sobre los registros del procesador?
Cada tarea tiene su propio contexto de ejecución. Por lo tanto, el cambio de contexto se produce cuando se quiere cambiar de una tarea a otra. Para eso el procesador se encarga de “salvar” todos los registros actuales del procesado (su contexto) , para empezar a restaurar el contexto de la nueva tarea a ejecutar. De esta forma los registros del procesador van a tener valores totalmente distintos porque se estará ejecutando **otro programa.**
Recordar que el **contexto de ejecución es** el conjunto de valores de los registros internos del procesador en cada momento.

31	15	0	
I/O Map Base Address	Reserved	T	100
Reserved	LDT Segment Selector		96
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 (PDBR)			28
Reserved	SS2		24
ESP2			20
Reserved	SS1		16
ESP1			12
Reserved	SS0		8
ESP0			4
Reserved	Previous Task Link		0

Expliquen en sus palabras que almacena el registro TR y cómo obtiene la información necesaria para ejecutar la tarea después de un cambio de contexto.
El registro **Task Register** (TR) almacena el selector de segmento de la tarea en ejecución. Se utiliza para encontrar la **TSS** de la tarea actual. Cuando se restaure el contexto de la nueva tarea el valor del registro TR se actualiza con el selector de la nueva tarea.

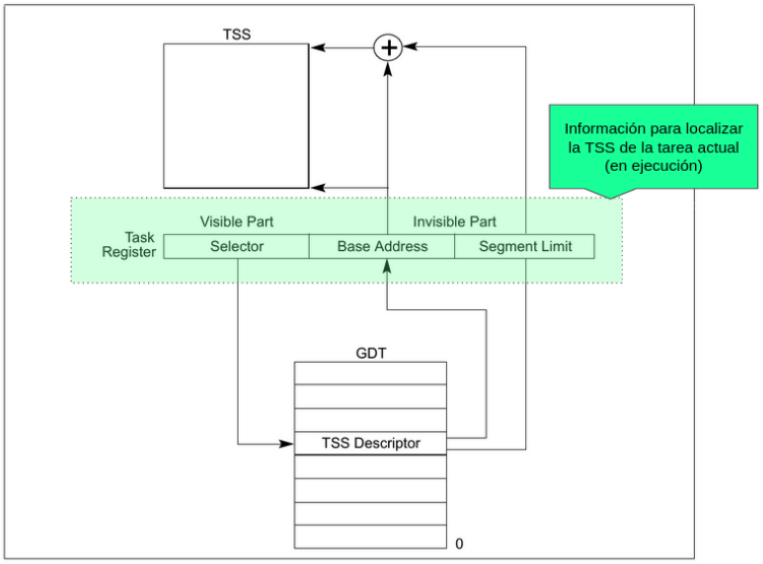


Figure 7-5. Task Register

3. Al momento de realizar un cambio de contexto el procesador va almacenar el estado actual de acuerdo al selector indicado en el registro TR y ha de restaurar aquel almacenado en la tss cuyo selector se asigna en el jmp far.
¿Qué consideraciones deberíamos tener para poder realizar el primer cambio de contexto? ¿Y cuáles cuando no tenemos tareas que ejecutar o se encuentran todas suspendidas?

1. El scheduler pide cambiar la tarea en ejecución haciendo un JMP al segmento de la tss de la nueva tarea a ejecutar en la GDT.
2. Antes de cambiar de contexto, debemos copiar el estado actual de los registros , en la tss de la tarea en ejecución.
3. Para eso con el registro TR buscan en la GDT el TSS Descriptor asociado a la tarea.
4. Una vez que encontró la TSS correspondiente a esa tarea copió todos los registros.
5. Con el contexto actual “salvado” debemos continuar la ejecución de la nueva tarea donde se la había dejado. Para eso copiamos los valores de la TSS en la memoria hacia la CPU y luego actualizamos la TR con la información de la nueva tarea en ejecución.

El procesador siempre precisa estar ejecutando una tarea, aunque no haga nada. Para esto definimos dos tareas especiales: la tarea Inicial y la tarea Idle.

Necesitamos dos pasos para dejar al kernel listo para ejecutar las tareas que querramos:

1. Apenas inicia el kernel hay que cargar la **tarea Inicial**. Para hacerlo, vamos a usar la instrucción LTR que toma como parámetro un registro de 16 bits con el selector de la tarea en la GDT.

   LDTR ax ; (con ax = selector segmento tarea inicial)

2. Luego, hay que saltar a la **tarea Idle**. La forma de hacerlo es saltar al selector con un JMP y el valor que pongamos en offset es ignorado (podemos poner 0).

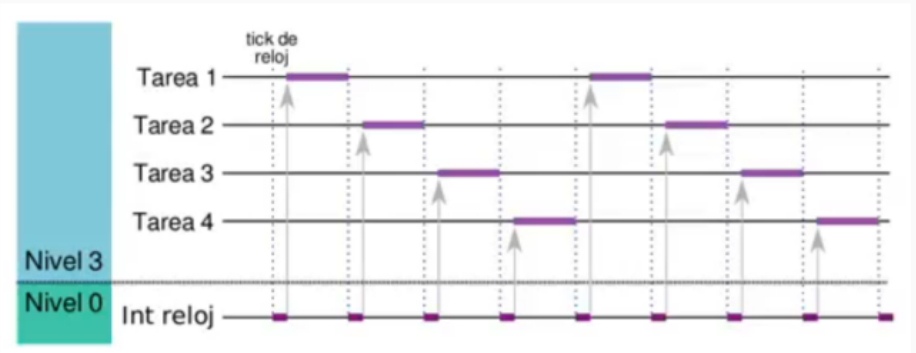
   JMP SELECTOR_TAREA_IDLE:0

Esto va a cambiar el valor del registro TR apuntando a la TSS de la tarea Idle y producir el cambio de contexto. Saltar a una tarea es algo que lo va a hacer el Sistema Operativo en nivel 0.

4. ¿Qué hace el scheduler de un Sistema Operativo? ¿A qué nos referimos con que usa una política?

El scheduler es un módulo de software que administra la ejecución de tareas/procesos. Se encarga de definir un intervalo de tiempo (time frame) a cada tarea. Que a su vez con ayuda de un temporizador (tics del reloj) es dividido en intervalos más pequeños. De esta forma el scheduler asigna distintas prioridades para cada tarea. Así cada tarea tiene unos milisegundos para progresar (ejecutarse), expirado ese tiempo se suspende la tarea y despacha para ejecutar la siguiente de la lista.

Utiliza una política o criterio para decir cuál es la próxima tarea a ejecutar y lo hace en cada **tic del reloj**



Cuando decimos que el scheduler utiliza una política nos referimos al criterio que lleva a cabo para decidir cuál es la próxima tarea a ejecutar. Esto significa la prioridad, el timeframe asignado para cada tarea,etc.

5. En un sistema de una única CPU, ¿cómo se hace para que lxs usuarixs vean todos los programas corriendo en simultáneo?
El procesador solo puede correr una única tarea por vez. Sin embargo gracias al scheduler nosotros como usuarios tenemos la sensación de que la ejecución de las distintas tareas se realiza en paralelo pero en realidad se hace en serie. Conmutando de una a otra a gran velocidad. Esto significa que se realizan muchos context switches por segundo, creándose así una sensación de simultaneidad. En otras palabras, es una ilusión.

11.
a) Expliquen con sus palabras que se estaría ejecutando en cada tic del reloj línea por línea

1. Define el selector de y el offset de la tarea a usar.
2. Pushea todos los registros.
3. Le indica al pic que la interrupción fue atendida.
4. Va a buscar el código de la siguiente tarea a ejecutar.
5. Lee el registro TR y realiza un compare con el registro ax para saber si es una tarea distinta. En caso de que la siguiente tarea sea igual a la actual salta a .fin. Caso contrario cambia de tarea.
6. Mueve el valor de ax a la pos de memoria del selector. sched_task_selector.
7. JMP far al contenido de la dirección indicada por shed_task_offset.

PREGUNTAR LOS TIC DE RELOJ.

12)
d) En la línea de la RAI del reloj que dice:
jmp far [sched_task_offset]
¿De qué tamaño es el dato que estaría leyendo desde la memoria? ¿Qué indica cada uno de estos valores? ¿Tiene algún efecto el offset elegido?
El jump recibe una dirección lógica de 48 bits. 32 bits del selector y 16 bits del offset.