Manejo Básico de Tareas y Scheduler

Organización del Computador II

5 de Noviembre de 2019

Introducción: Tareas

- Una tarea/task es una unidad de trabajo que el procesador puede despachar, ejecutar y suspender.
- La tarea ejecuta una instancia de un programa y se puede ver como un contexto de ejecución.
- La arquitectura provee un mecanismo para salvar el contexto de una tarea, comenzarla a ejecutar o conmutarla con otra.

Introducción: Tareas

Una tarea está compuesta por:

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

Introducción: Tareas

Estructura de una tarea

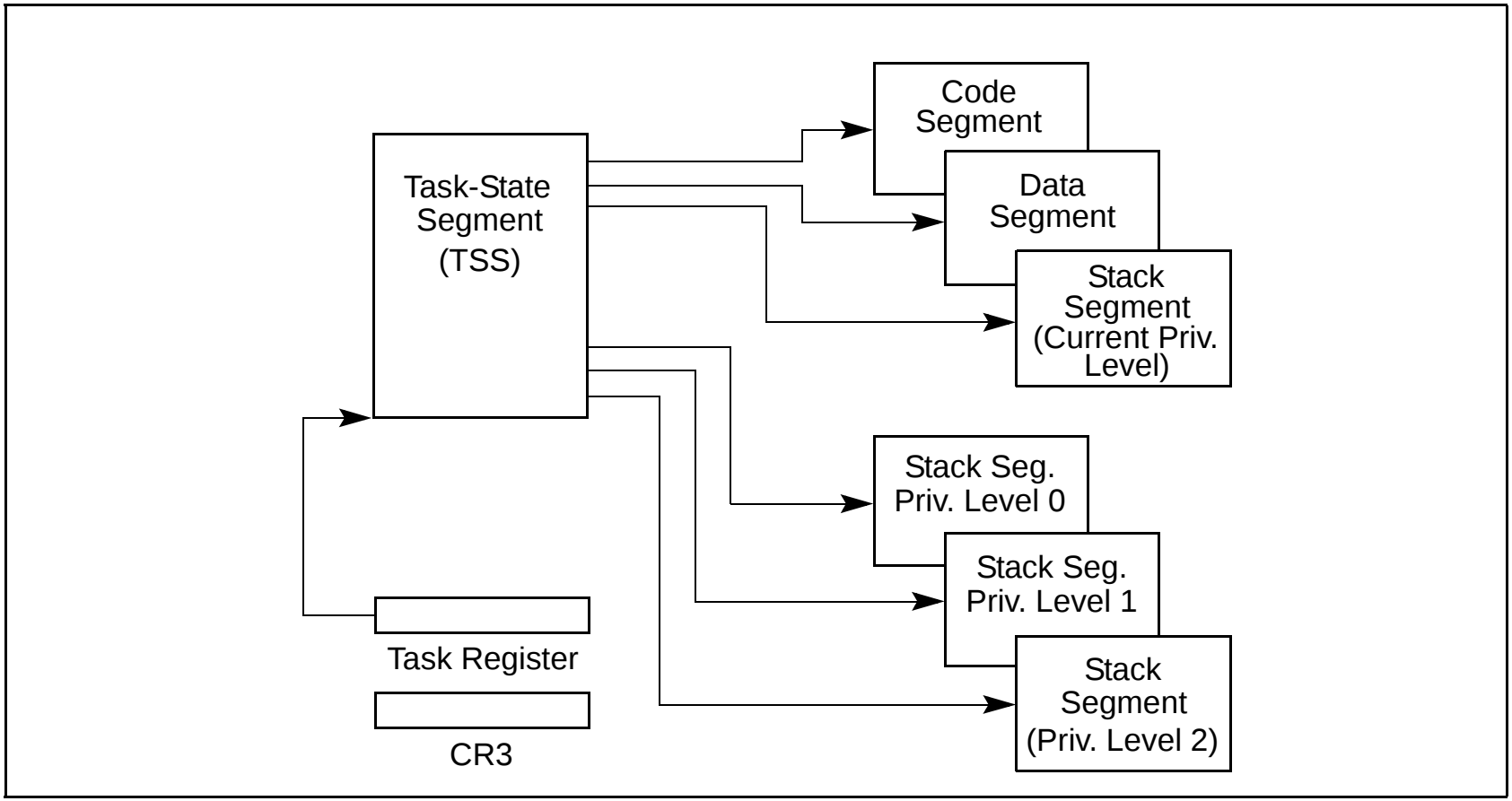


Figure 7-1. Structure of a Task

Introducción: Identificación de una tarea

- Una tarea está identificada por un selector de segmento de TSS.
- La TSS debe estar descripta en la GDT del mismo modo que se describen los segmentos de código y datos.
- El registro Task Register (TR) contiene selector de segmento de TSS de la tarea que se está ejecutando actualmente.

TSS: Task-State Segment

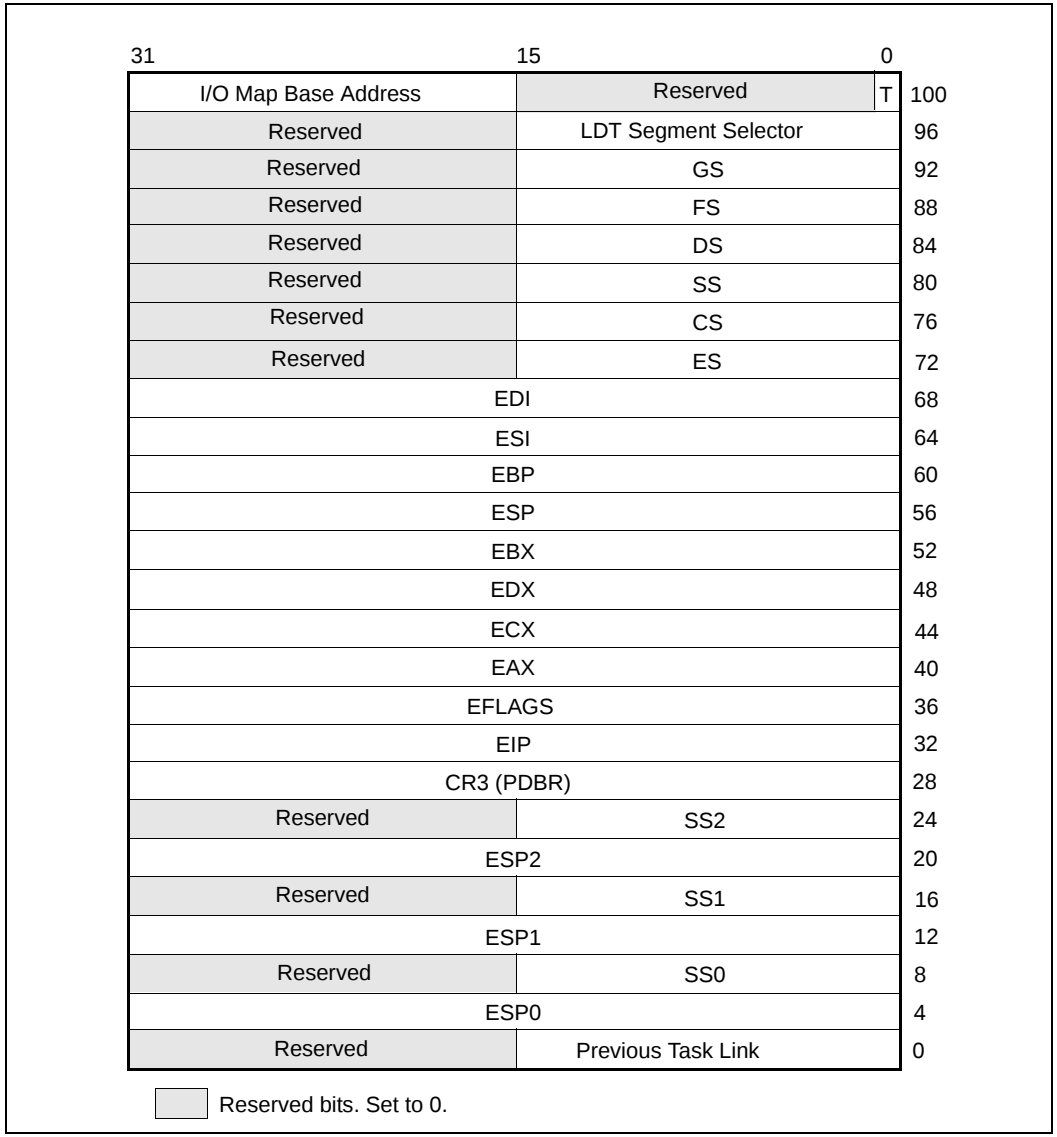


Figure 7-2. 32-Bit Task-State Segment (TSS)

TSS: Descriptor de TSS

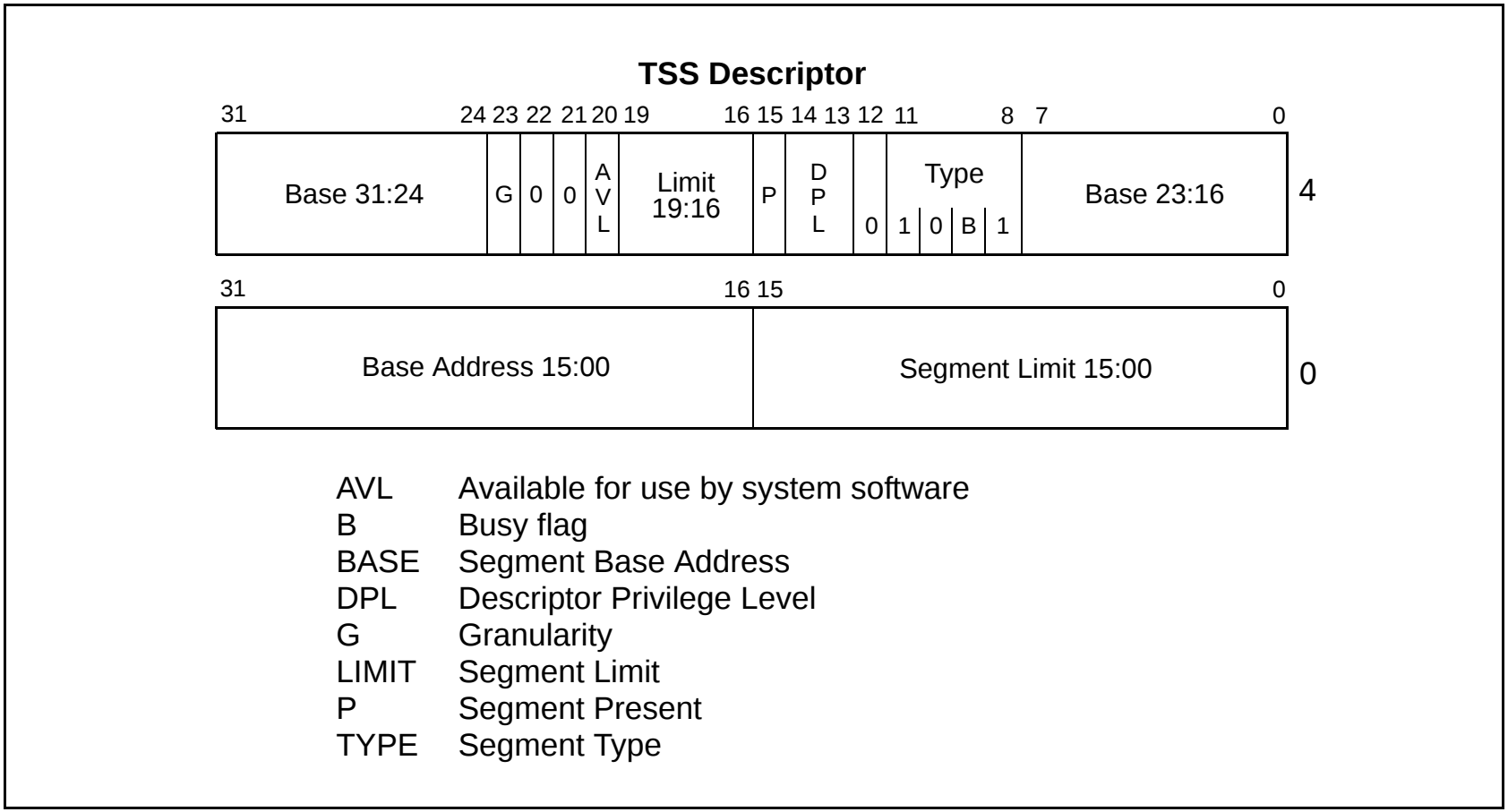


Figure 7-3. TSS Descriptor

TSS: Encontrando la TSS de la tarea acutal

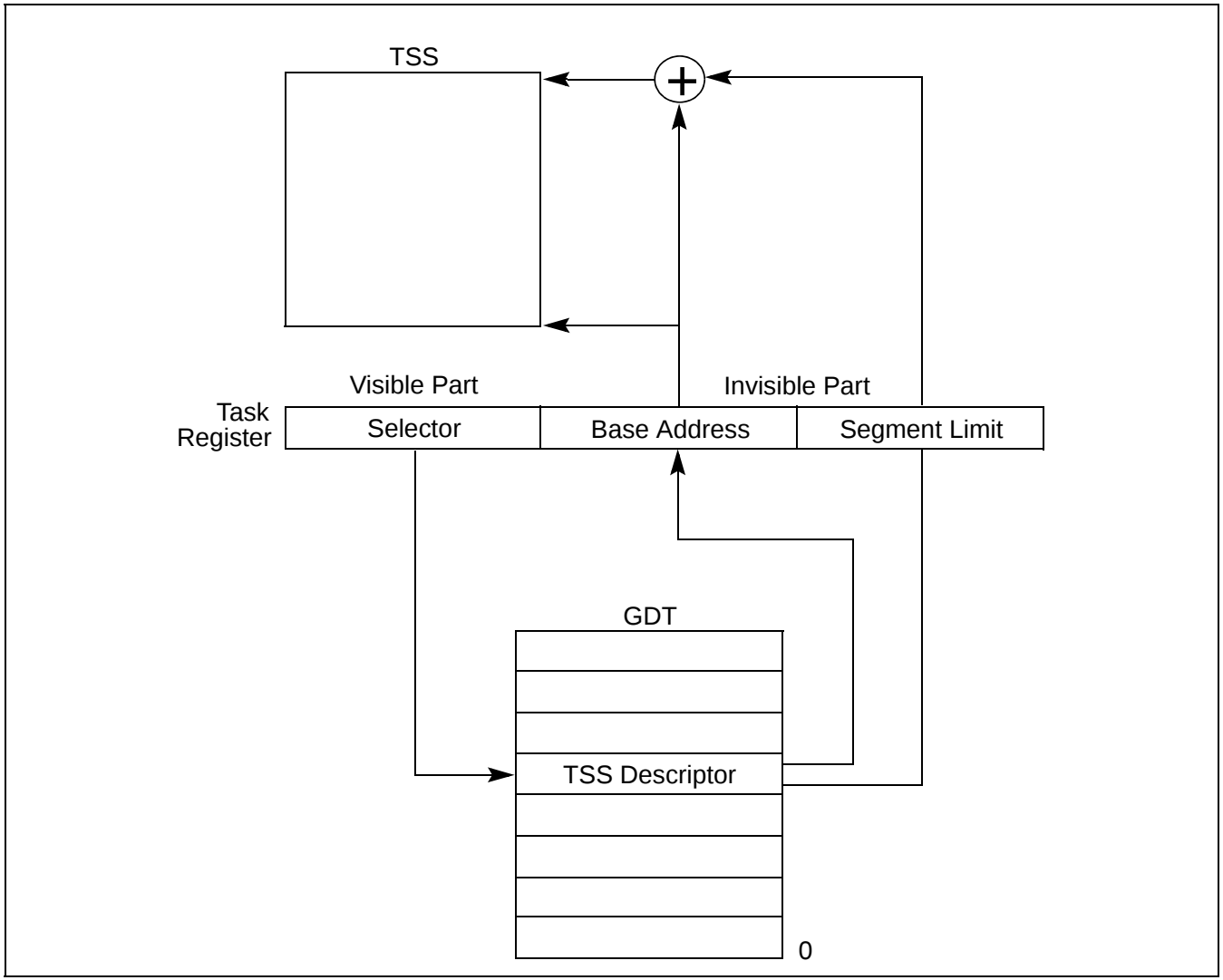
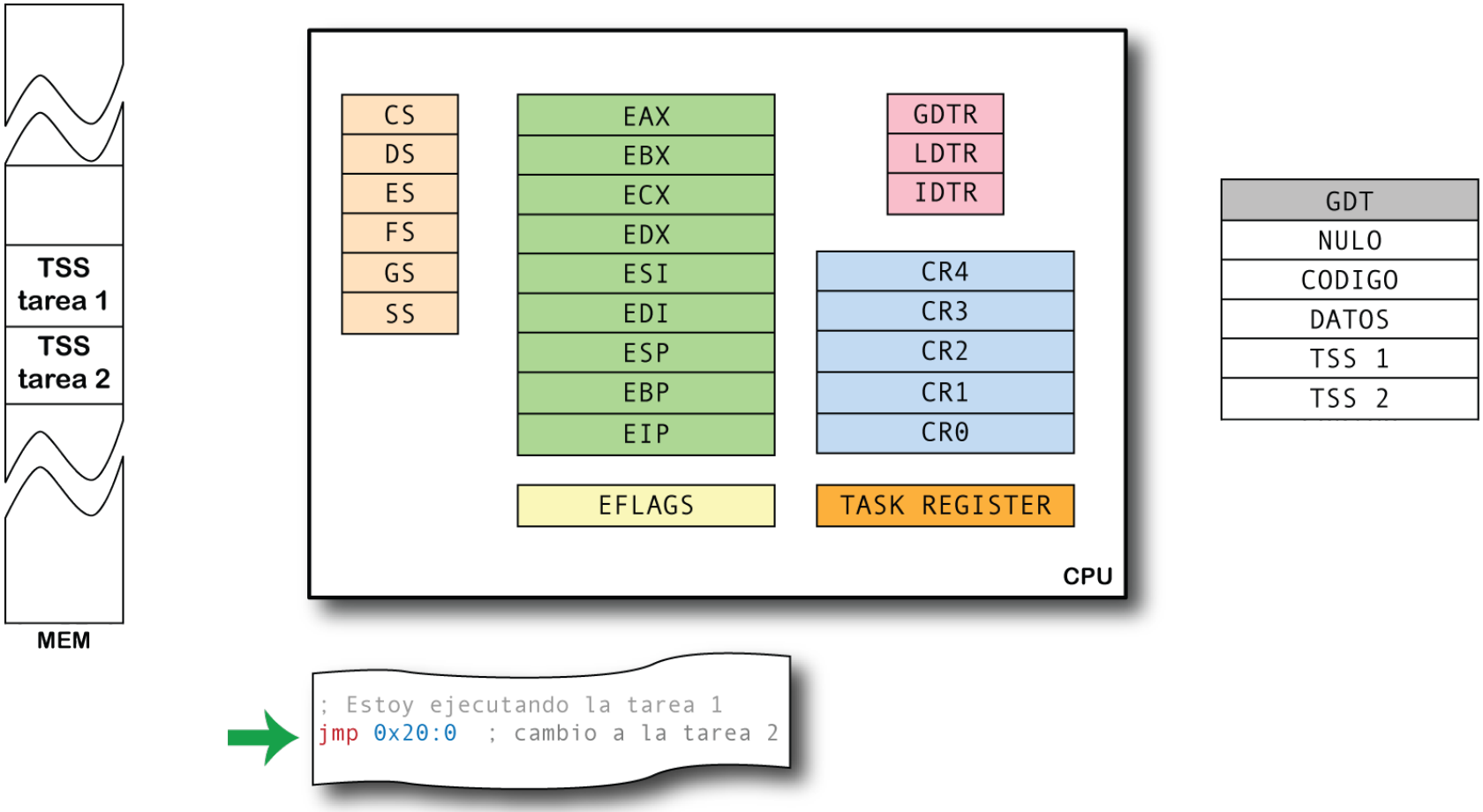


Figure 7-5. Task Register

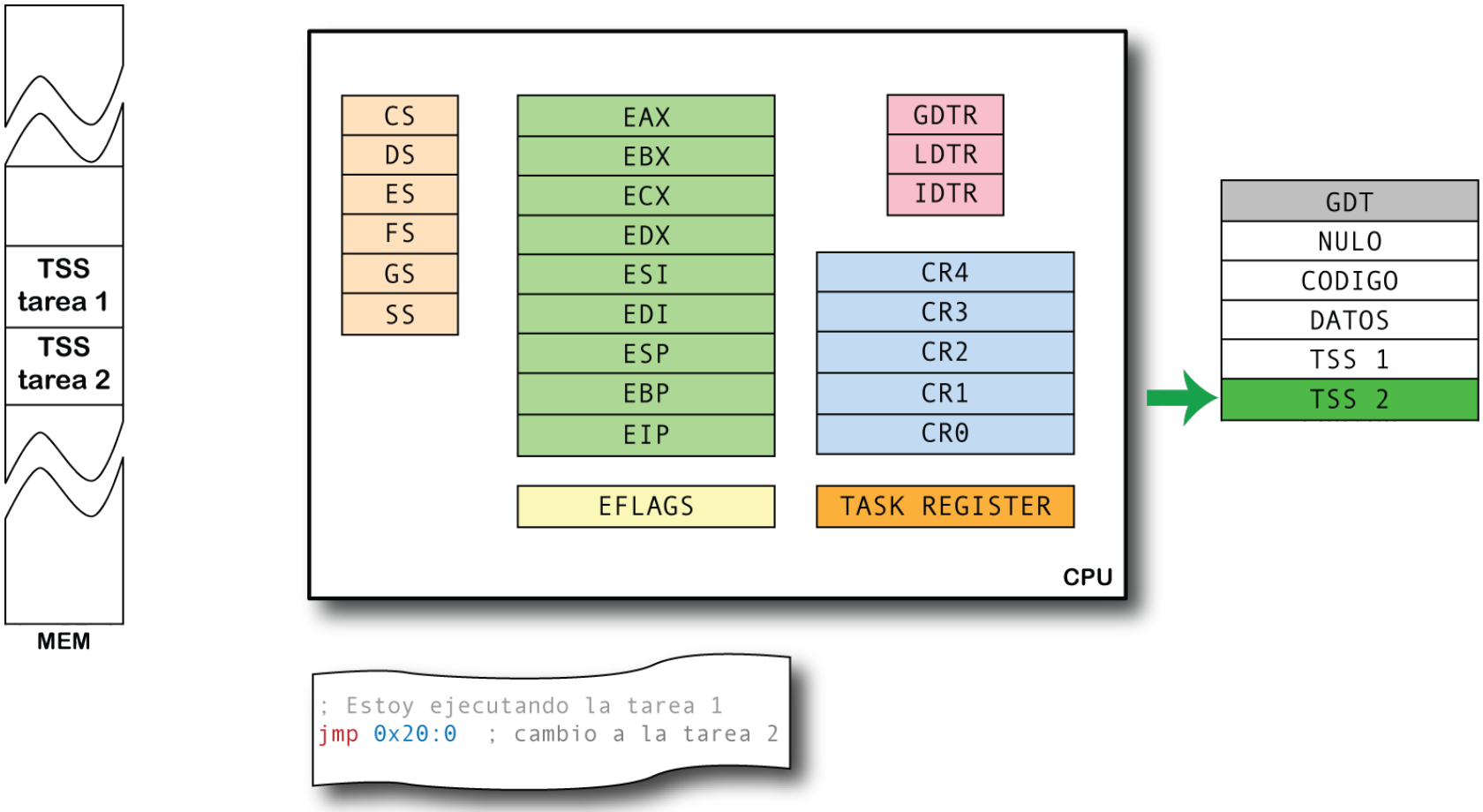
Conmutación de Tareas

1. Ejecutamos la instrucción `jmp 0x20:0`



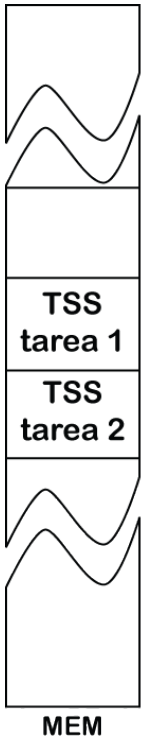
Conmutación de Tareas

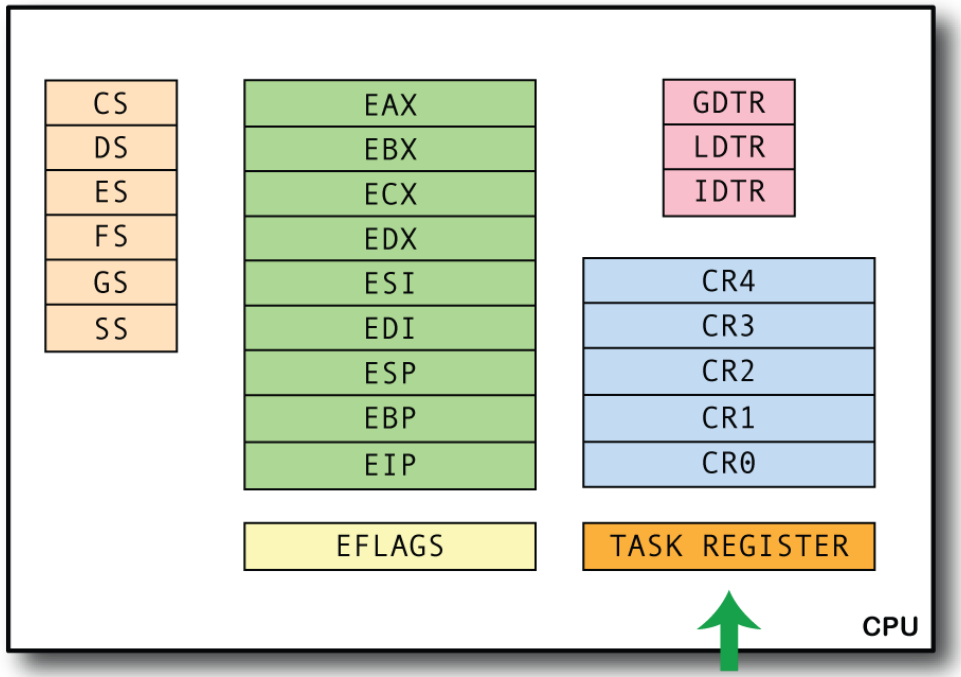
2. Se busca el descriptor correspondiente en la GDT.

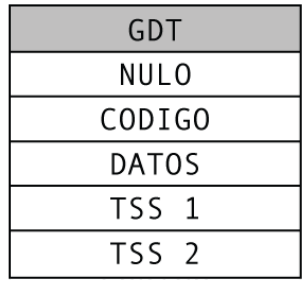


Conmutación de Tareas

3. Como es un cambio de tarea, se lee el TR.



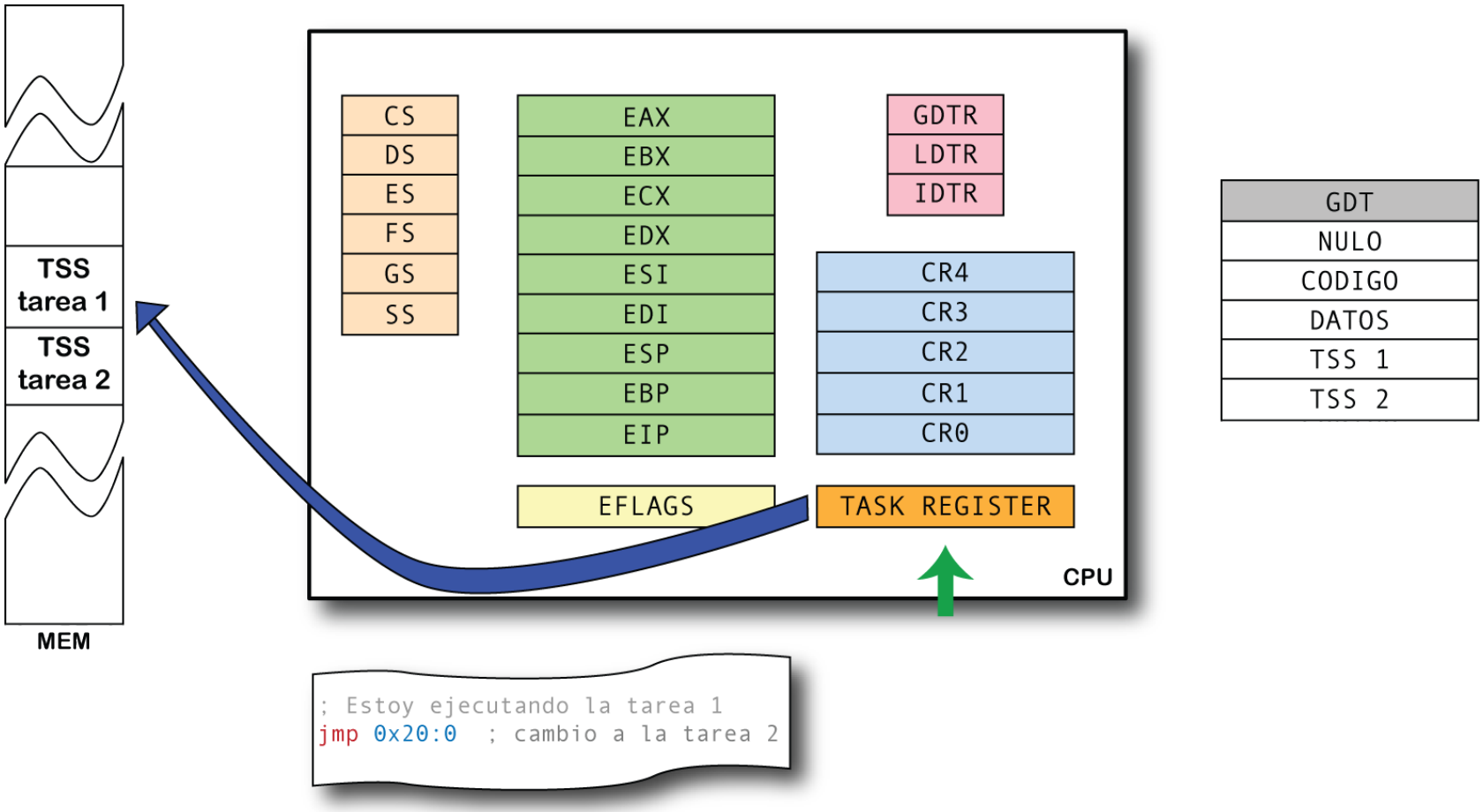





```
; Estoy ejecutando la tarea 1
jmp 0x20:0  ; cambio a la tarea 2
```

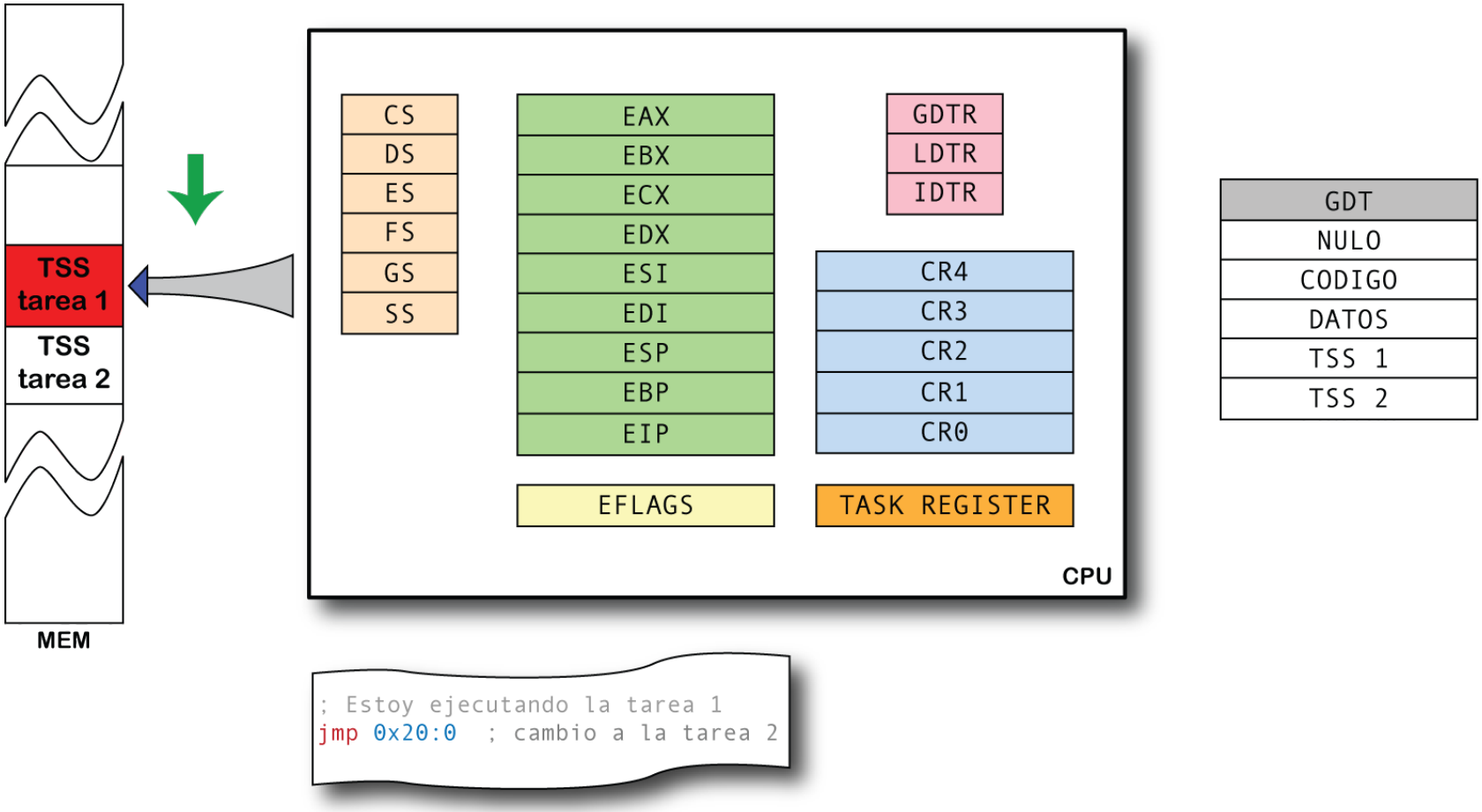
Conmutación de Tareas

4. Se busca el TSS apuntado por el TR.



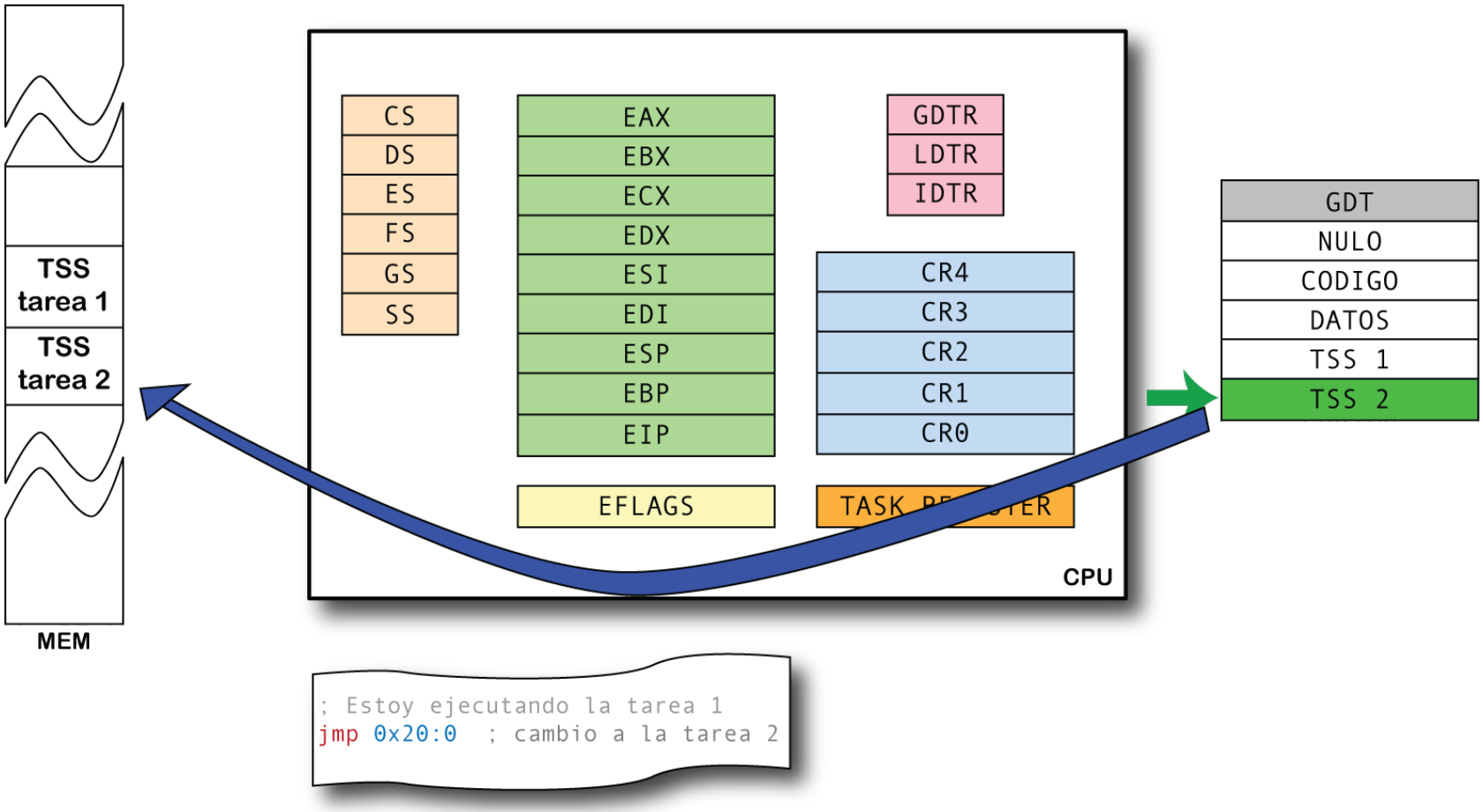
Conmutación de Tareas

5. Se guarda el contexto actual.



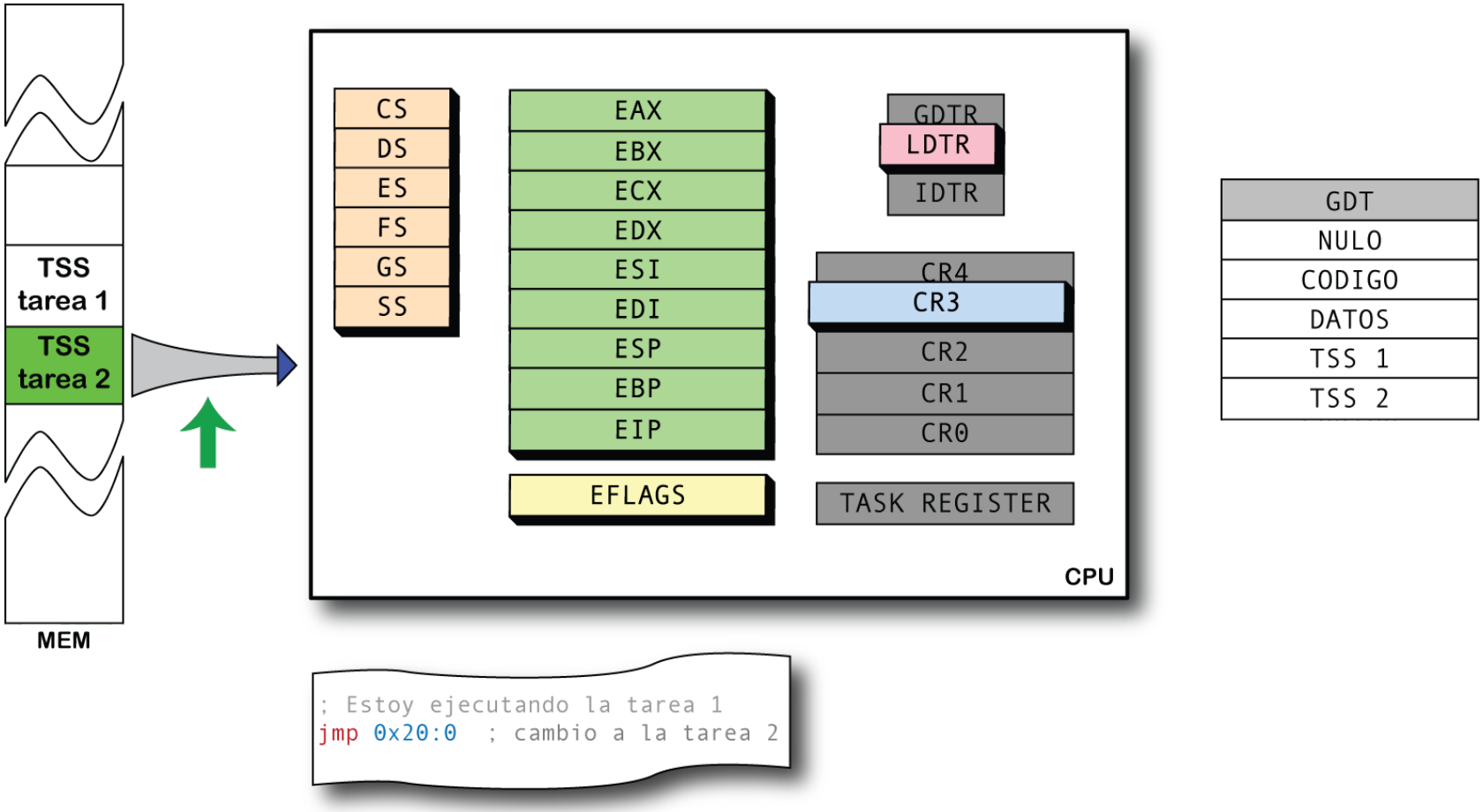
Conmutación de Tareas

6. Se busca TSS apuntado por descriptor de tarea a ejecutar.



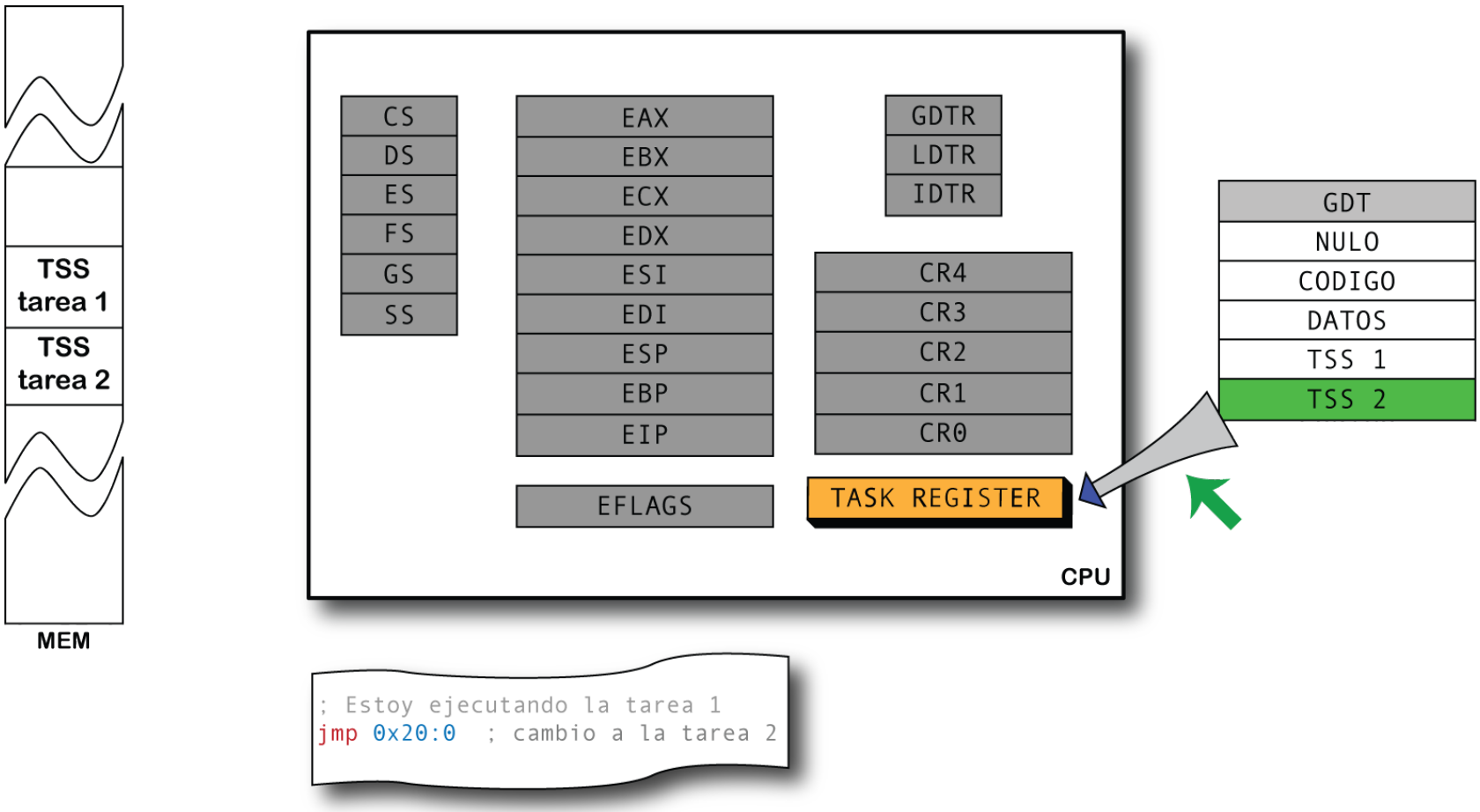
Conmutación de Tareas

7. Se obtiene el nuevo contexto.



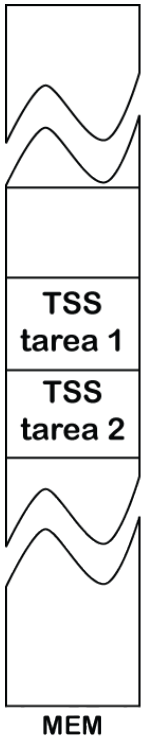
Conmutación de Tareas

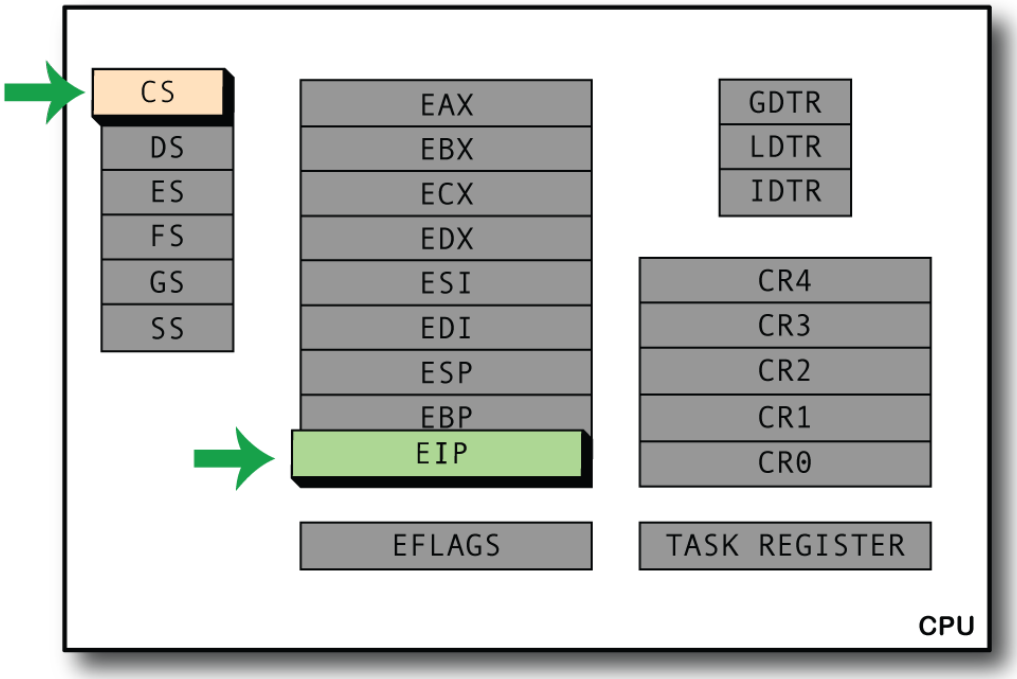
8. Se actualiza el TR.

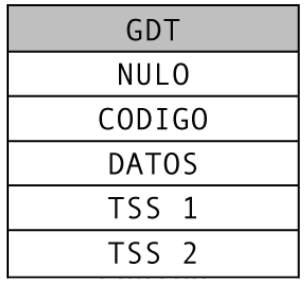


Conmutación de Tareas

9. Se continúa le ejecución con el nuevo contexto.








```
; Estoy ejecutando la tarea 1
jmp 0x20:0  ; cambio a la tarea 2
```

Conmutación de Tareas: Tarea Inicial

1. Siempre que se salta a una tarea, hay un cambio de contexto, ¡Siempre!

2. El procesador guarda el contexto actual de la tarea (identificada en TR) y carga el contexto de la tarea a la cual se está saltando.

Conmutación de Tareas: Tarea Inicial

1. Siempre que se salta a una tarea, hay un cambio de contexto, ¡Siempre!

2. El procesador guarda el contexto actual de la tarea (identificada en TR) y carga el contexto de la tarea a la cual se está saltando.

3. Entonces, ¿qué pasa la primera vez? ¿Qué pasa cuando se salta a la primera tarea? ¿Qué valor contiene TR? ¿Dónde se guarda el contexto?

Conmutación de Tareas: Tarea Inicial

1. Siempre que se salta a una tarea, hay un cambio de contexto, ¡Siempre!

2. El procesador guarda el contexto actual de la tarea (identificada en TR) y carga el contexto de la tarea a la cual se está saltando.

3. Entonces, ¿qué pasa la primera vez? ¿Qué pasa cuando se salta a la primera tarea? ¿Qué valor contiene TR? ¿Dónde se guarda el contexto?

4. 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.

El TP3

Ejercicio 6

a) Definir las entradas en la `GDT` que considere necesarias. Mínimo, una para la *tarea inicial* y otra para la tarea Idle.

b) Completar la entrada de la `TSS` de la tarea Idle (`0x0001C000`). La pila será la misma del kernel mapeada con *identity mapping*, utilizando además el mismo `CR3` que el *kernel*. La tarea ocupa 4KB y debe ser mapeada con *identity mapping*.

c) Construir una función que complete una `TSS` libre con los datos correspondientes a una tarea. El código de las tareas está a partir de `0x00010000` ocupando 8kb cada una. El tope de la pila crecerá desde la dirección relativa 7kb o 8kb según sea una tarea o el *handler*. Construir un mapa de memoria utilizando `mmu_initTaskDir`. Para la pila de nivel 0, pedir una nueva pagina del área libre de kernel.

El TP3

Ejercicio 6

d) Completar la entrada de la `GDT` correspondiente a la *tarea inicial*.

e) Completar la entrada de la `GDT` correspondiente a la tarea `Idle`.

f) Escribir el código necesario para ejecutar la tarea `Idle`, es decir, saltar intercambiando las `TSS`, entre la *tarea inicial* y la tarea `Idle`.

Checklist

1. Al iniciar las tareas:
   - completar **EIP**
   - completar **ESP** y **EBP**
   - completar selectores de segmento
   - completar **CR3**
   - completar **EFLAGS**

2. 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** algún valor válido (tarea inicial)

Checklist

1. Al iniciar las tareas:
   - completar **EIP**
   - completar **ESP** y **EBP**
   - completar selectores de segmento (recordar RPL)
   - completar **CR3**
   - completar **EFLAGS** (*)

2. 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** algún valor válido (tarea inicial)

EFLAGS

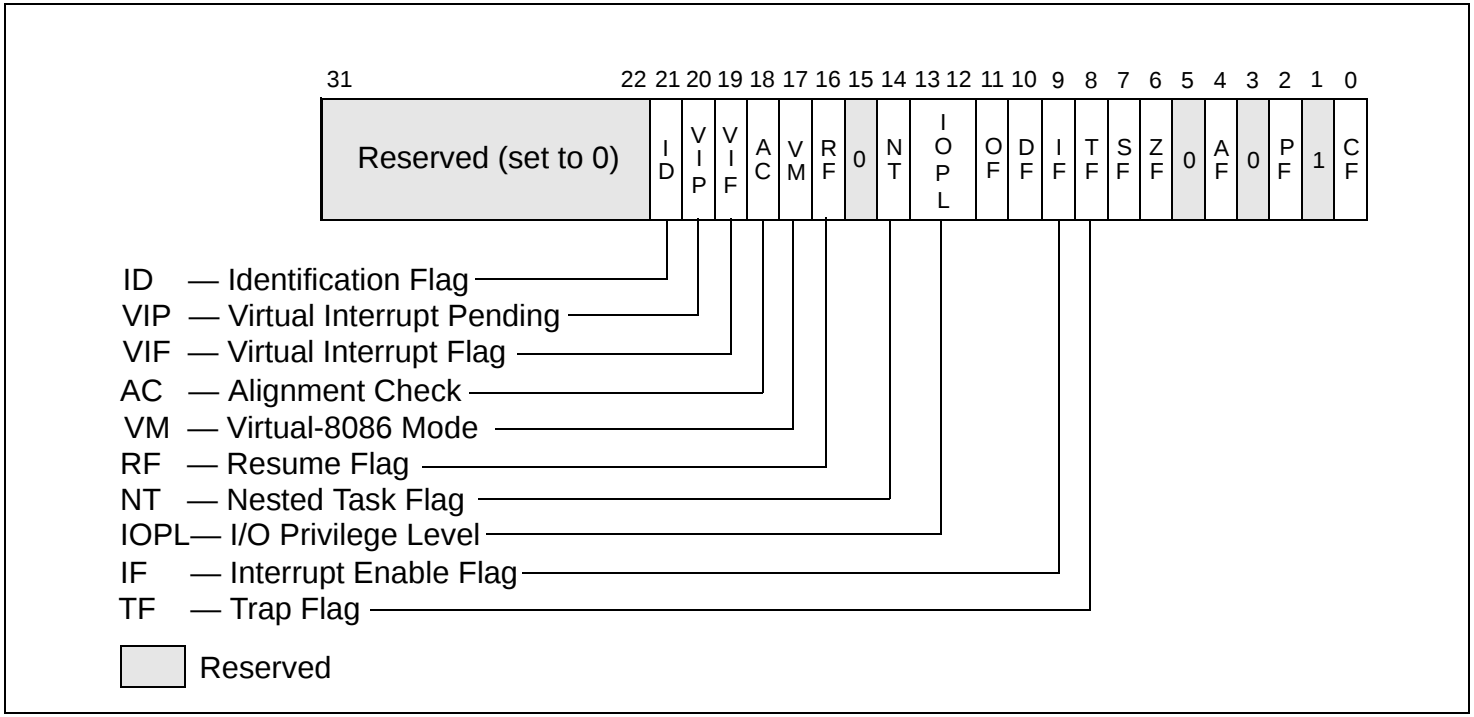


Figure 2-4. System Flags in the EFLAGS Register

- EFLAGS por defecto 0x00000002
- EFLAGS con Interrupciones habilitadas 0x00000202

Intercambiando tareas

- Cargar la tarea inicial en el `TR`

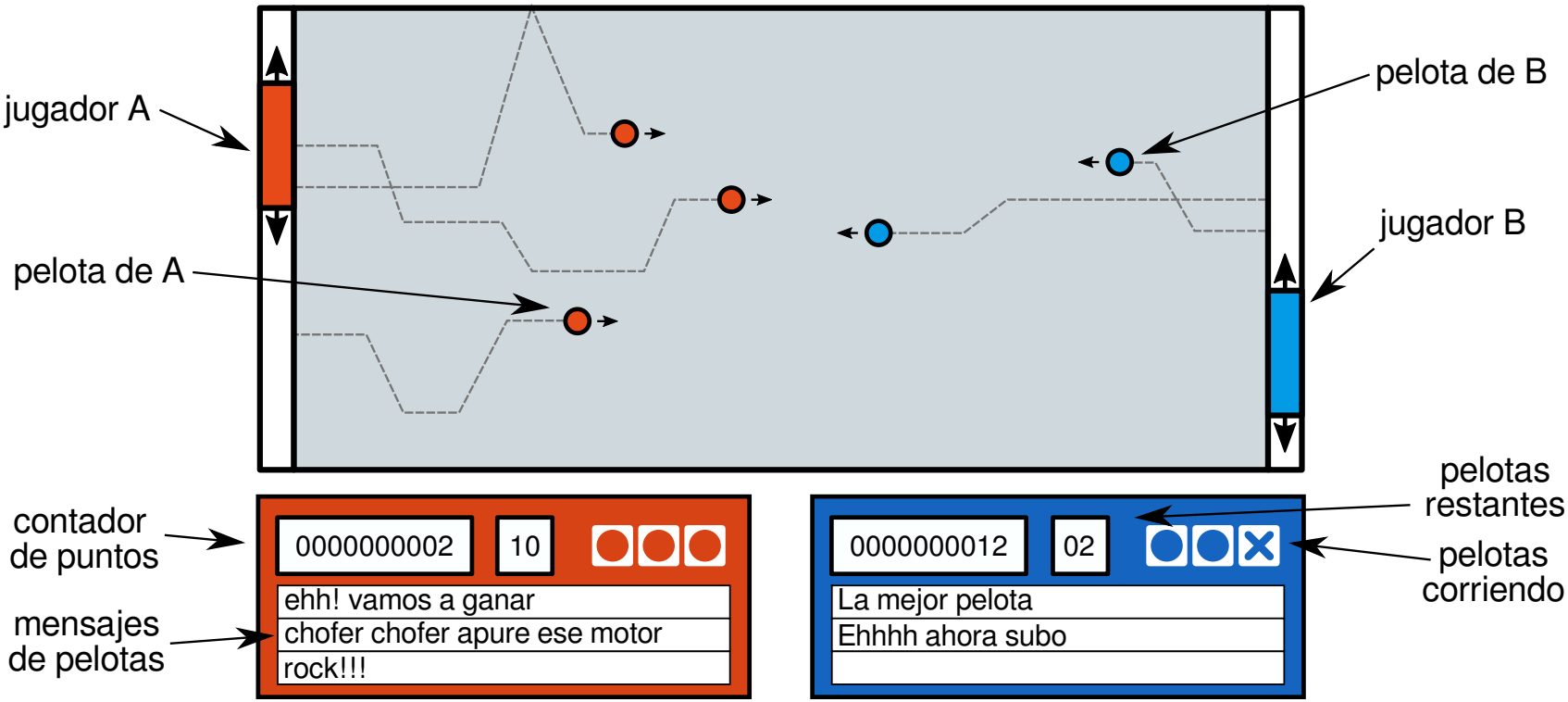
```
mov ax, <indice>
ltr ax
```

- Saltar a la nueva tarea (intercambio)

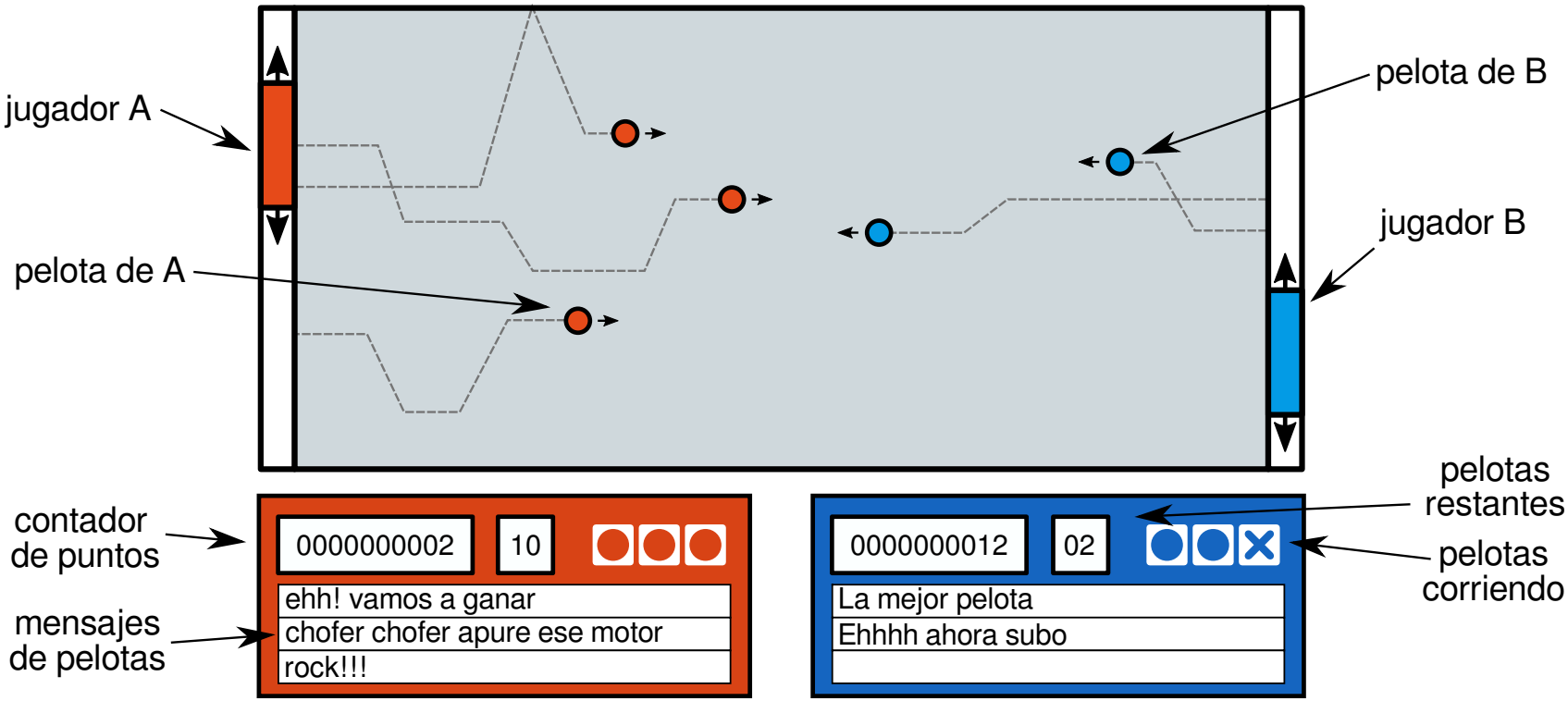
```
jmp <sel.tarea_idle>:0
```

Game

Game


jugador A
pelota de A
pelota de B
jugador B
contador
de puntos
mensajes
de pelotas
0000000002
10
ehh! vamos a ganar
chofer chofer apure ese motor
rock!!!
0000000012
02
La mejor pelota
Ehhhh ahora subo
pelotas
restantes
pelotas
corriendo

Game



- Posición de los Jugadores: `Pos[player]`
- Pelotas restantes por jugador: `Life[player]`
- Puntos por jugador: `Count[player]`
- Posición de las pelotas por jugador: `TaskPos[player][task][xy]`
- Interpretación de las coordenadas: `TaskRef[player][task][xy]`

Game

Jugador	Tecla	Acción
Jugador A	`w`	mover hacia arriba
	`s`	mover hacia abajo
	`z`, `x`, `c`	nueva pelota tipo 1, 2, 3
Jugador B	`i`	mover hacia arriba
	`k`	mover hacia abajo
	`b`, `n`, `m`	nueva pelota tipo 1, 2, 3

Game

Jugador	Tecla	Acción
Jugador A	`w`	mover hacia arriba
	`s`	mover hacia abajo
	`z`, `x`, `c`	nueva pelota tipo 1, 2, 3
Jugador B	`i`	mover hacia arriba
	`k`	mover hacia abajo
	`b`, `n`, `m`	nueva pelota tipo 1, 2, 3

Nueva tarea:

1. Buscar espacio libre donde crear la tarea
2. Crear un nuevo mapa de memoria
   - Identity Mapping primeros 4MB
   - Copiar y mapear código
3. Completar información de estado
   - Completar la posición dentro del mapa
   - Agregar a la lista de tareas

Game **Syscalls**

`talk`

```
in EAX=0x80000001
in EBX=char* m
```

Envia el mensaje en `m`. (`strlen(m)`<20)

`where`

```
in EAX=0x80000002
out EBX=uint32_t x
out ECX=uint32_t y
```

Retorna las coordenadas `x` e `y` de la pelota.

`setHandler`

```
in EAX=0x80000003
in EBX=f_handler_t* f
```

Registra la dirección de la función *handler*.

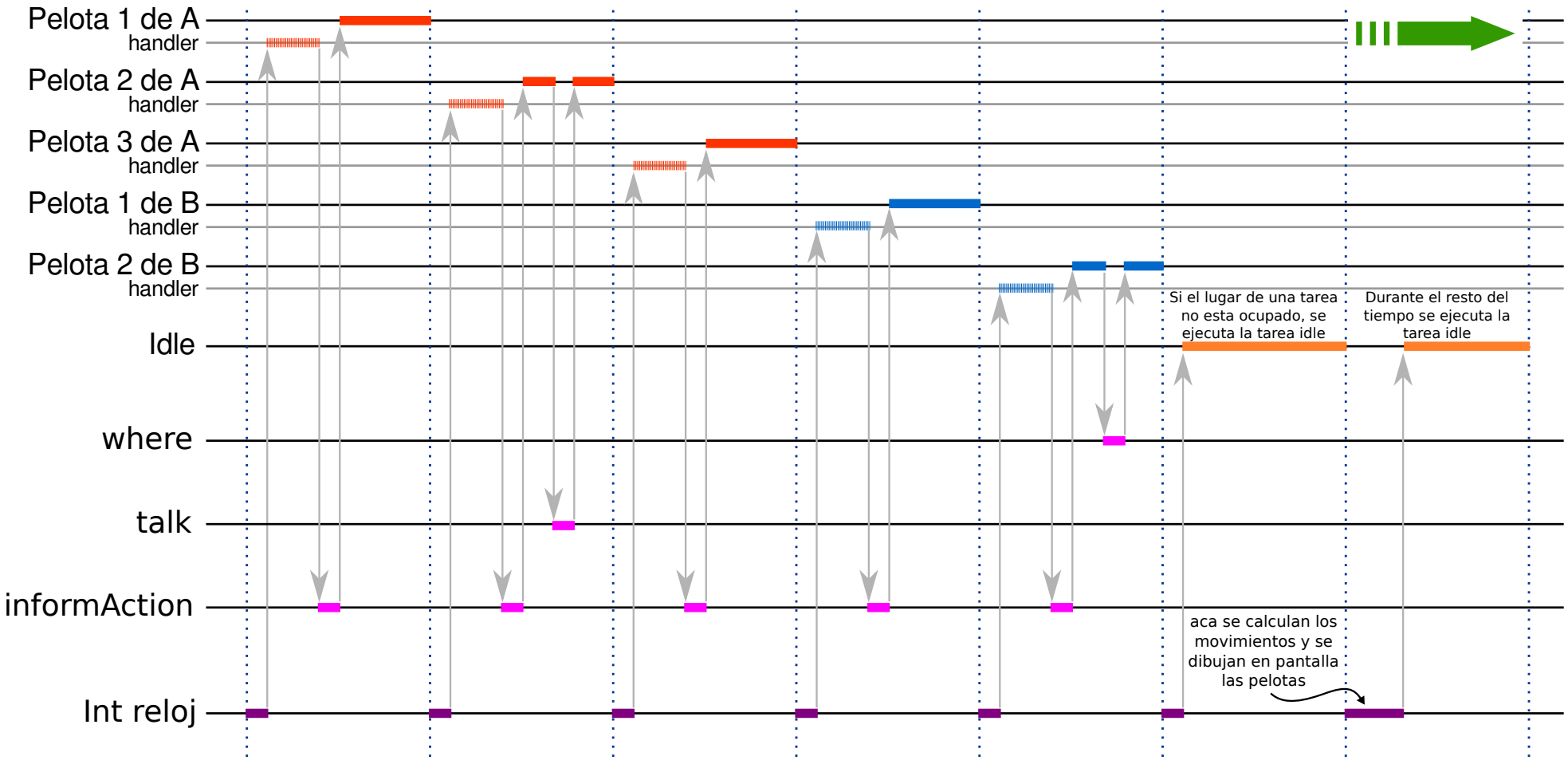
`informAction`

```
in EAX=0x8000FFFF
in EBX=e_action action
```

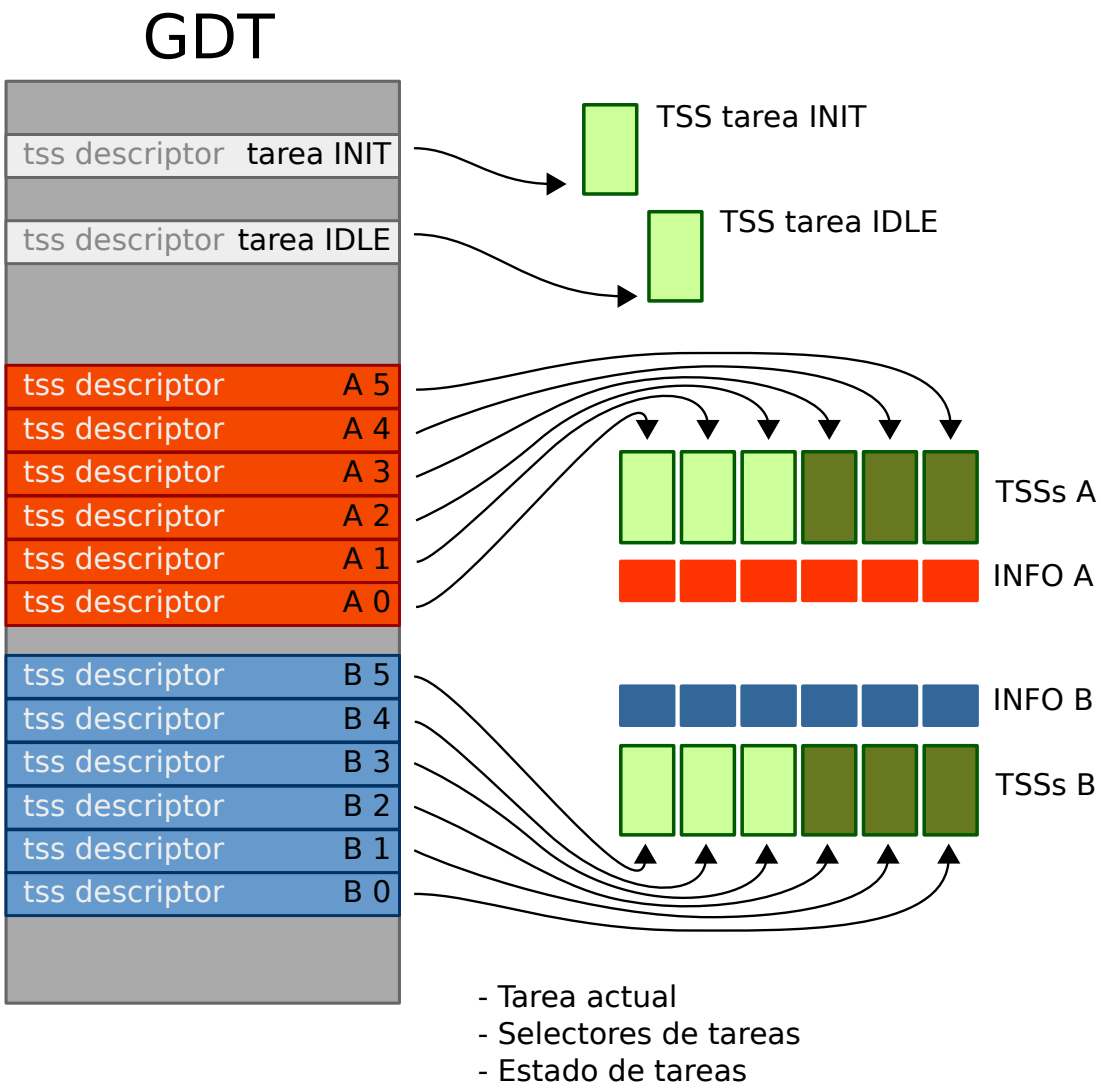
Retornar al sistema desde la función `handler`.

Scheduler

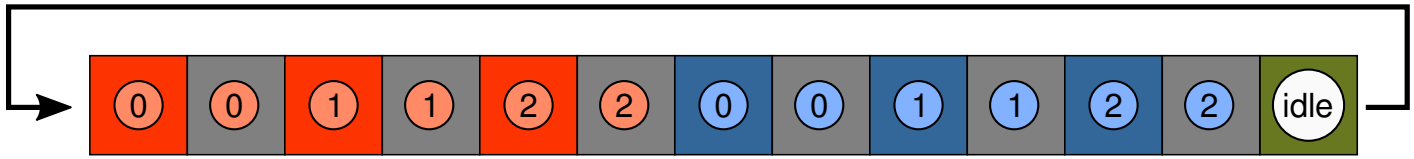
Scheduler


Pelota 1 de A
handler
Pelota 2 de A
handler
Pelota 3 de A
handler
Pelota 1 de B
handler
Pelota 2 de B
handler
Idle
where
talk
informAction
Int reloj
Si el lugar de una tarea no esta ocupado, se ejecuta la tarea idle
Durante el resto del tiempo se ejecuta la tarea idle
aca se calculan los movimientos y se dibujan en pantalla las pelotas

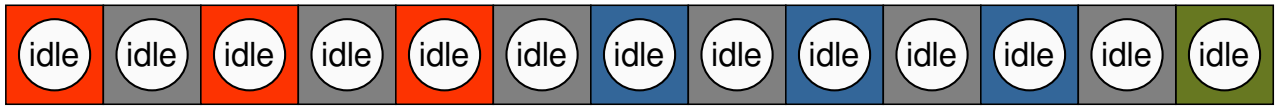
Scheduler


GDT
tss descriptor tarea INIT
tss descriptor tarea IDLE
tss descriptor A 5
tss descriptor A 4
tss descriptor A 3
tss descriptor A 2
tss descriptor A 1
tss descriptor A 0
tss descriptor B 5
tss descriptor B 4
tss descriptor B 3
tss descriptor B 2
tss descriptor B 1
tss descriptor B 0
TSS tarea INIT
TSS tarea IDLE
TSSs A
INFO A
INFO B
TSSs B
- Tarea actual
- Selectores de tareas
- Estado de tareas

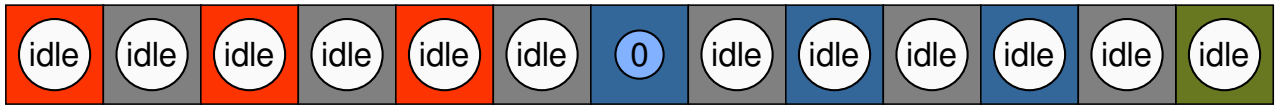
Scheduler



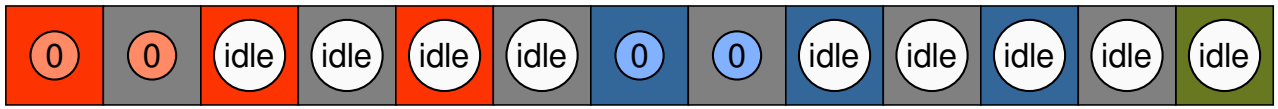
Incialmente



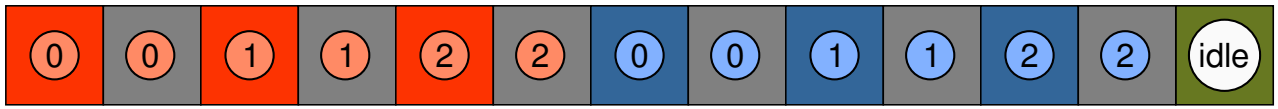
Se crea una pelota del jugador 2 (supongo que no registro su función durante el primer ciclo)



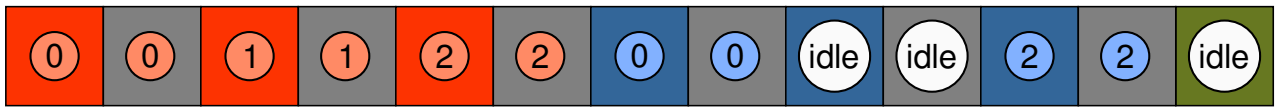
Se crea una pelota del jugador 1, se registra la función y la pelota de 2 registra su función también.



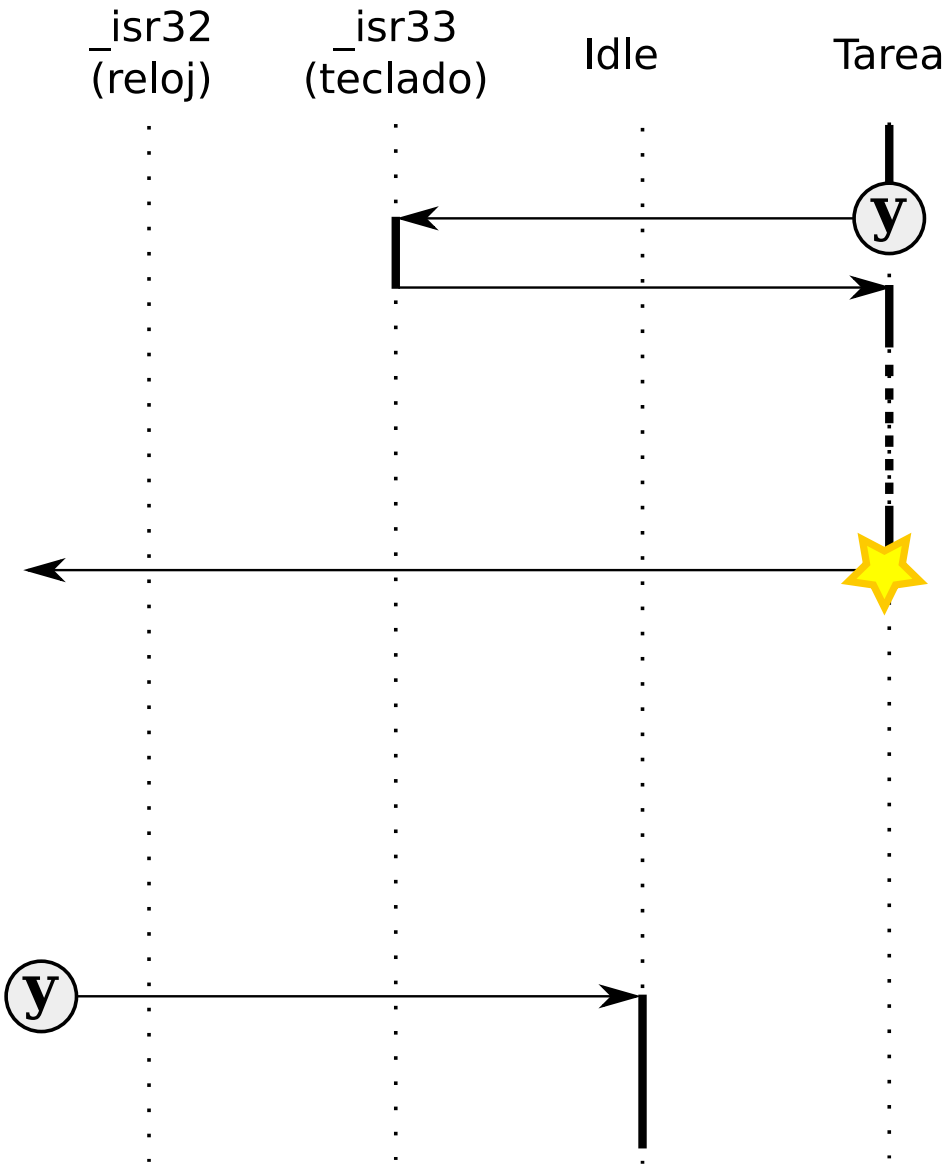
Continua el juego hasta que todos los slots están cubiertos



Si una pelota llega al otro extremo, o genera un error, el slot es liberado (tarea 1 de jugador 2)



Modo Debug



- Se presiona la tecla "y" y se activa/desactiva el modo debug
- Alguna tarea produce una excepción, de estar en modo debug se detiene la ejecución mostrando en patalla el estado
- Cuando se presiona la tecla "y" nuevamente, se reanuda la ejecución

El TP3

Ejercicio 7

a) Construir una función para inicializar las estructuras de datos del *scheduler*.

b) Crear la función `sched_nextTask()` que devuelve el índice en la `GDT` de la próxima tarea a ser ejecutada.

c) Modificar la rutina de interrupciones `0x47` para que implemente los distintos servicios del sistema.

d) Modificar el código necesario para que se realice el intercambio de tareas por cada ciclo de reloj. El intercambio se realizará según indique la función `sched_nextTask()`.

e) Modificar las rutinas de excepciones del procesador para que desalojen a la tarea que estaba corriendo y ejecuten la próxima.

f) Implementar el mecanismo de debugging indicando en pantalla la razón del desalojo de la tarea.

Ayudas

- Leer una tecla

```
xor eax, eax
in al, 60h
test al, 10000000b
jnz .fin
```

- Saltar a una tarea

```
offset: dd 0
selector: dw 0
...
mov [selector], ax
jmp far [offset]
...
```

Ayudas

Rutina de atención de interrupciones para el reloj

```
global _isr32
_isr32:
   pushad

   call pic_finish1

   call sched_nextTask

   str cx
   cmp ax, cx
   je .fin

      mov [selector], ax
      jmp far [offset]

   .fin:

   popad
 iret
```

Ayudas

ASM para usar desde C (i386.h)

```
void lcr0(unsigned int val)
unsigned int rcr0(void)

void lcr1(unsigned int val)
unsigned int rcr1(void)

void lcr2(unsigned int val)
unsigned int rcr2(void)

void lcr3(unsigned int val)
unsigned int rcr3(void)

void lcr4(unsigned int val)
unsigned int rcr4(void)

void tlbflush(void)

void ltr(unsigned short sel)
unsigned short rtr(void)

void breakpoint(void)
```

¿Preguntas?