Manejo Básico de Tareas

Organización del Computador II

29 de octubre de 2015

Introducción: Tareas

- Una tarea/task es una unidad de trabajo que el procesador puede despachar, ejecutar y suspender.
- La tarea se suele usar para ejecutar una instancia de un programa.
- La arquitectura provee mecanismos para salvar el estado de una tarea, para despachar una tarea para su ejecución y para conmutar tareas.

Introducción: Tareas

Una tarea está compuesta por:

1. Espacio de ejecución:
   - Segmento de código.
   - Segmento de pila (uno o varios).
   - Segmento de datos (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

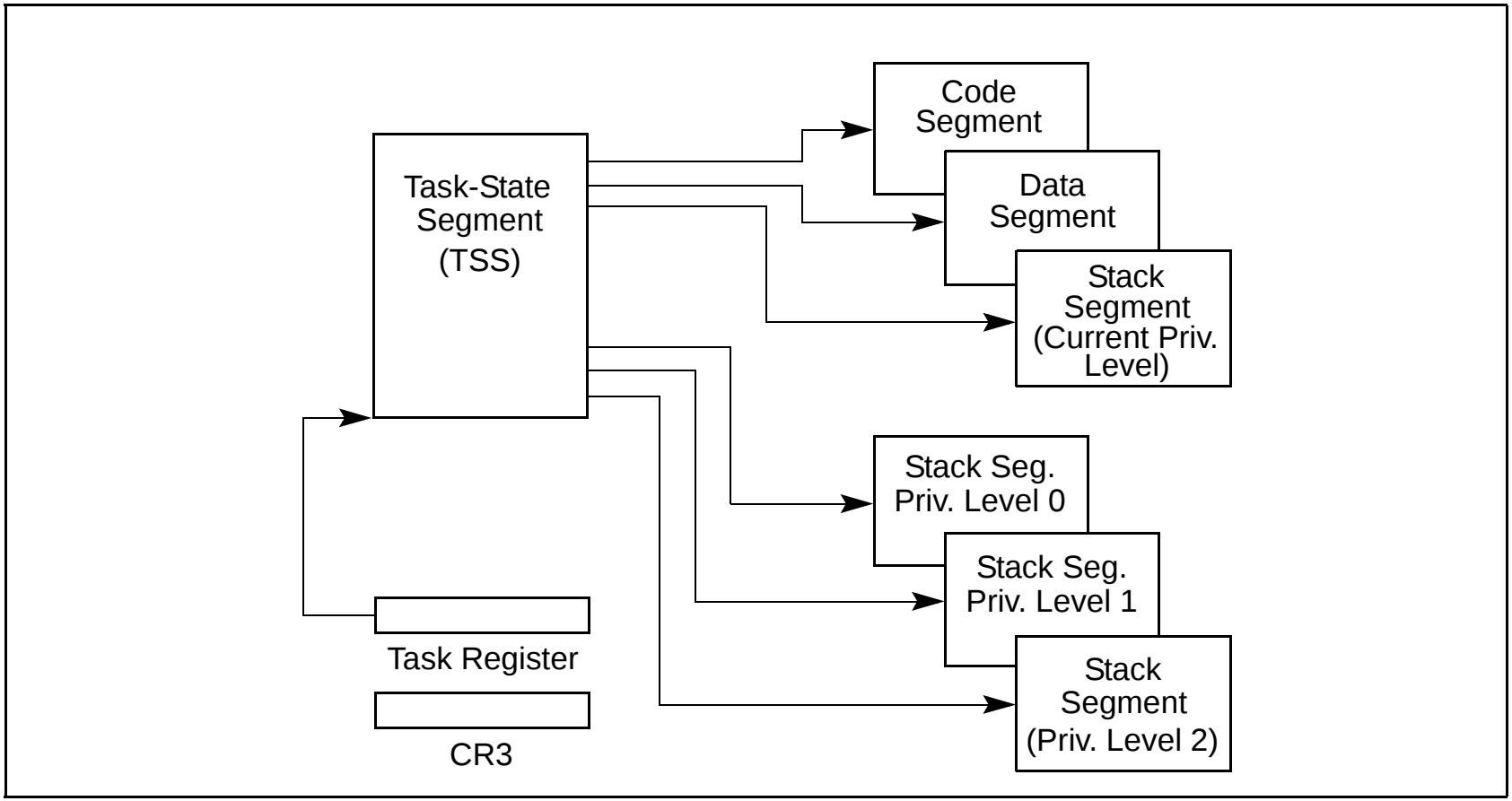


Figure 7-1. Structure of a Task

Introducción: Identificación de una tarea

- Una tarea está identificada por 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: Task-State Segment

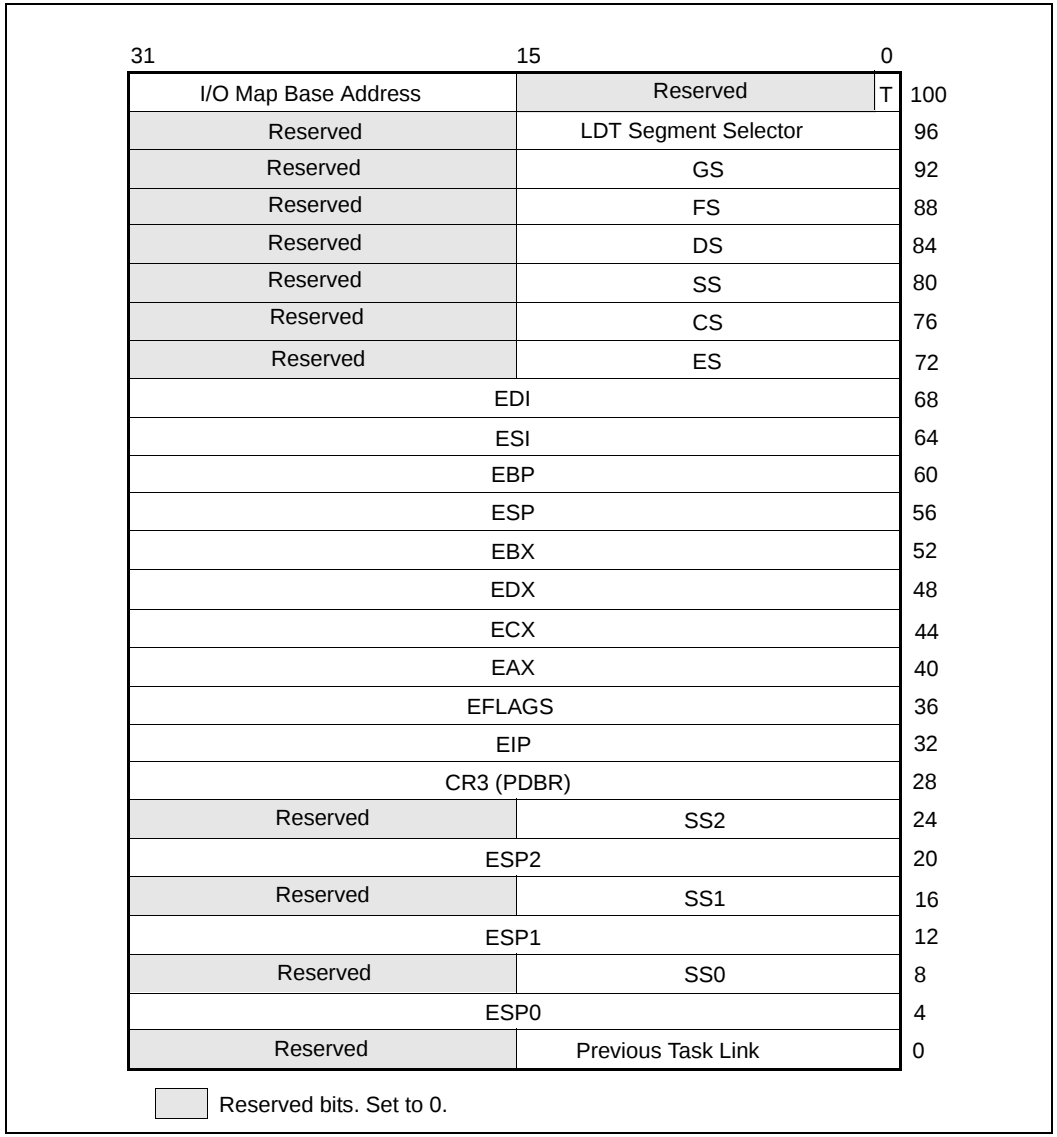


Figure 7-2. 32-Bit Task-State Segment (TSS)

TSS: Descriptor de TSS

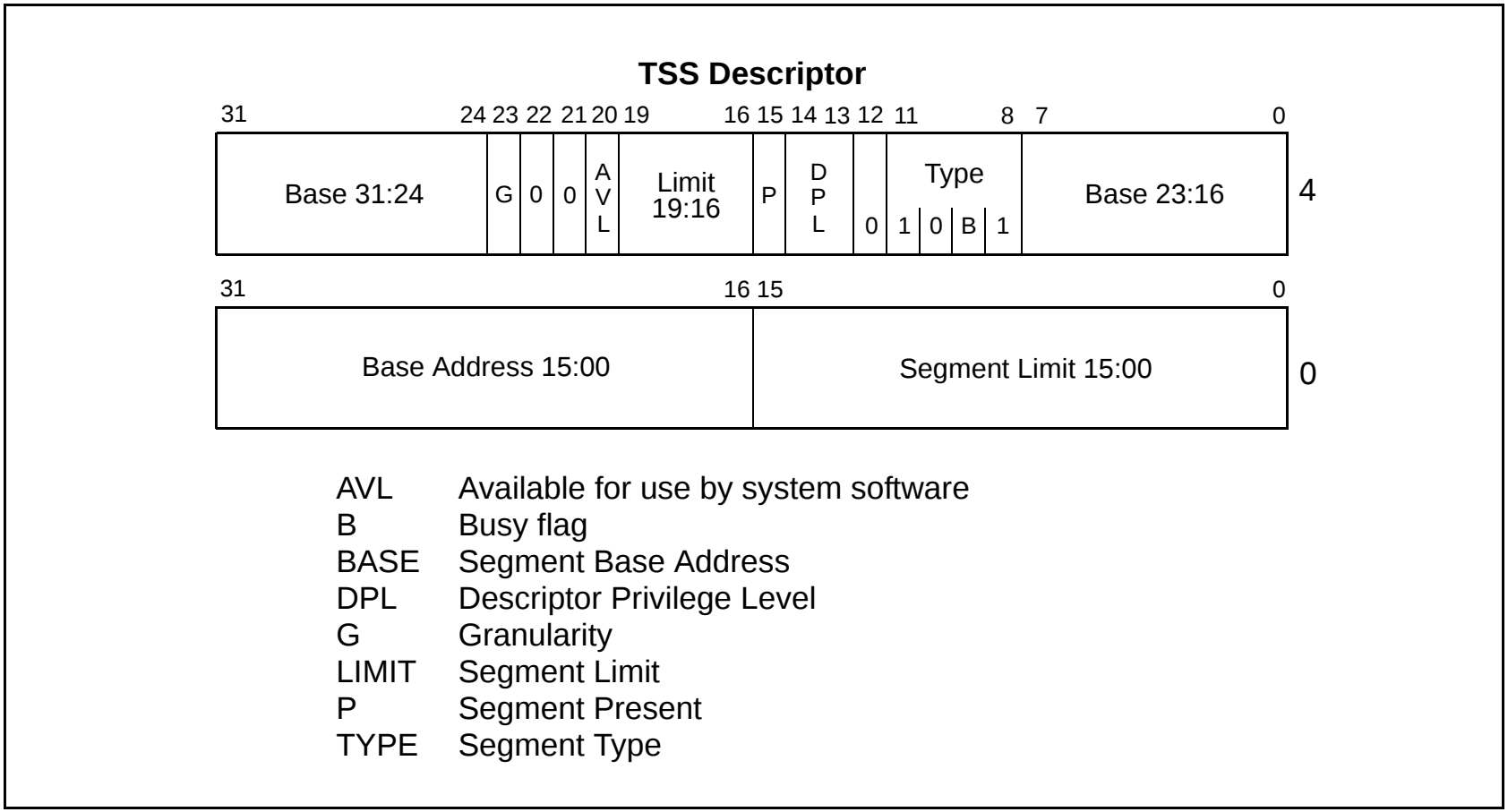


Figure 7-3. TSS Descriptor

TSS: Encontrando la TSS de la tarea acutal

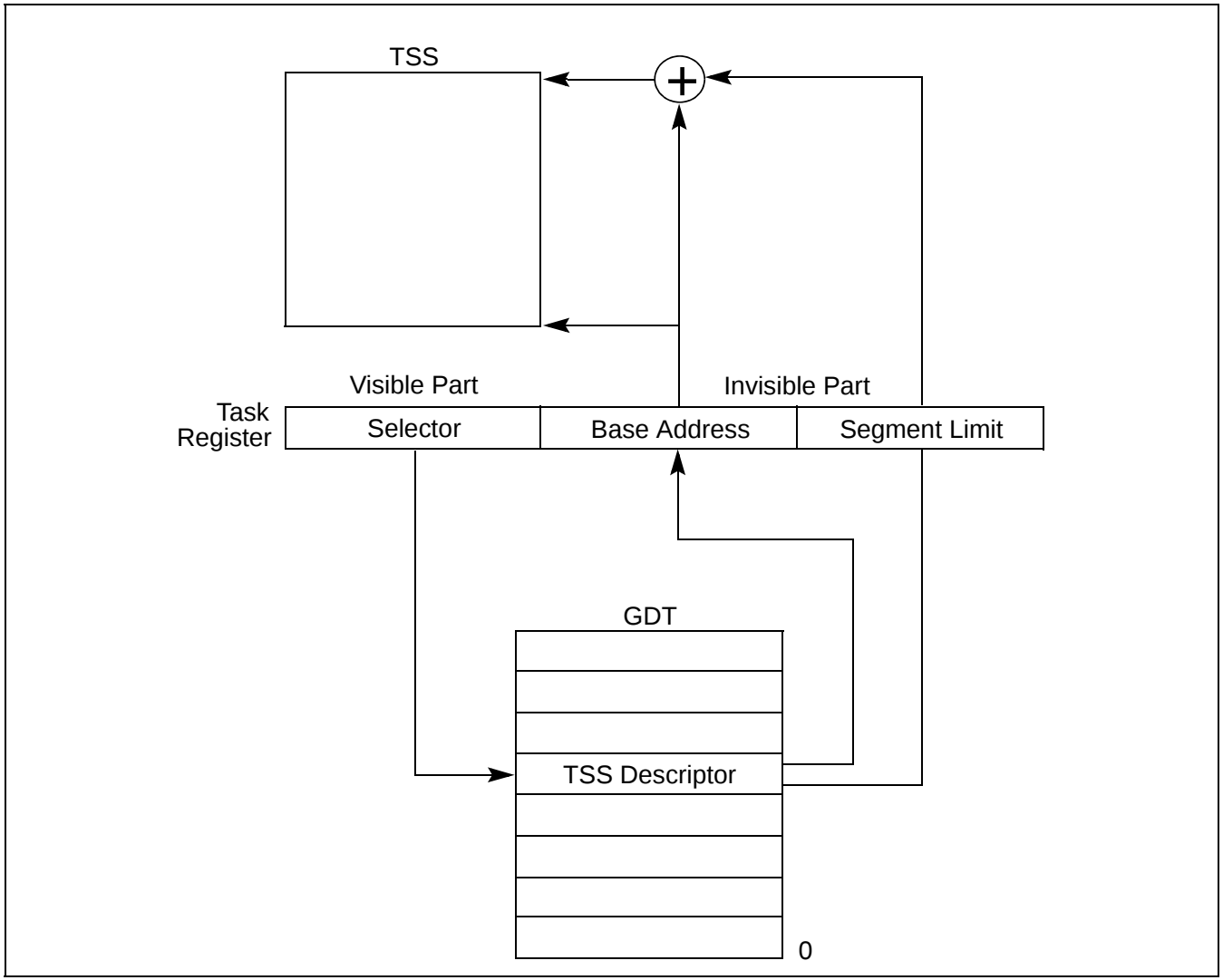
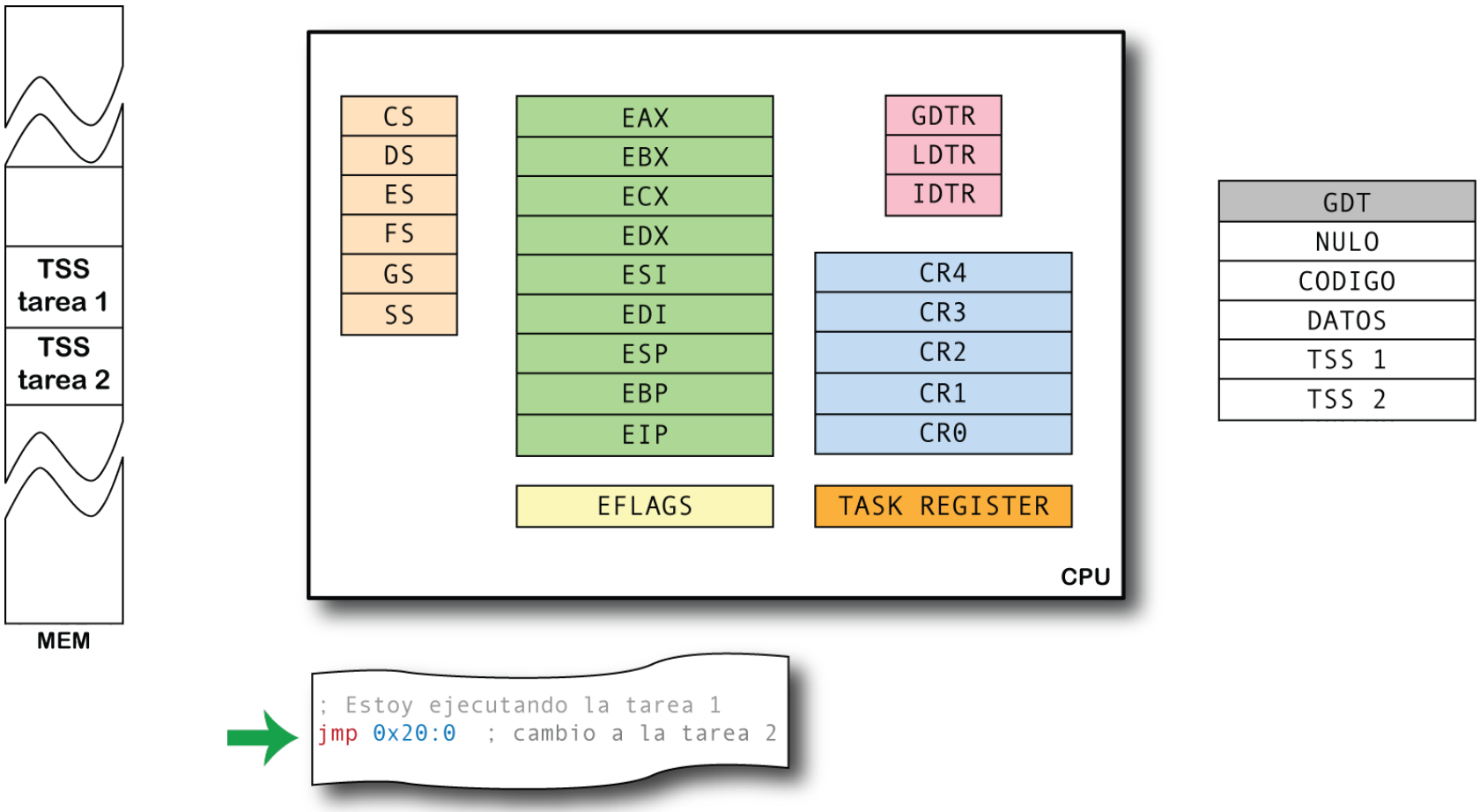


Figure 7-5. Task Register

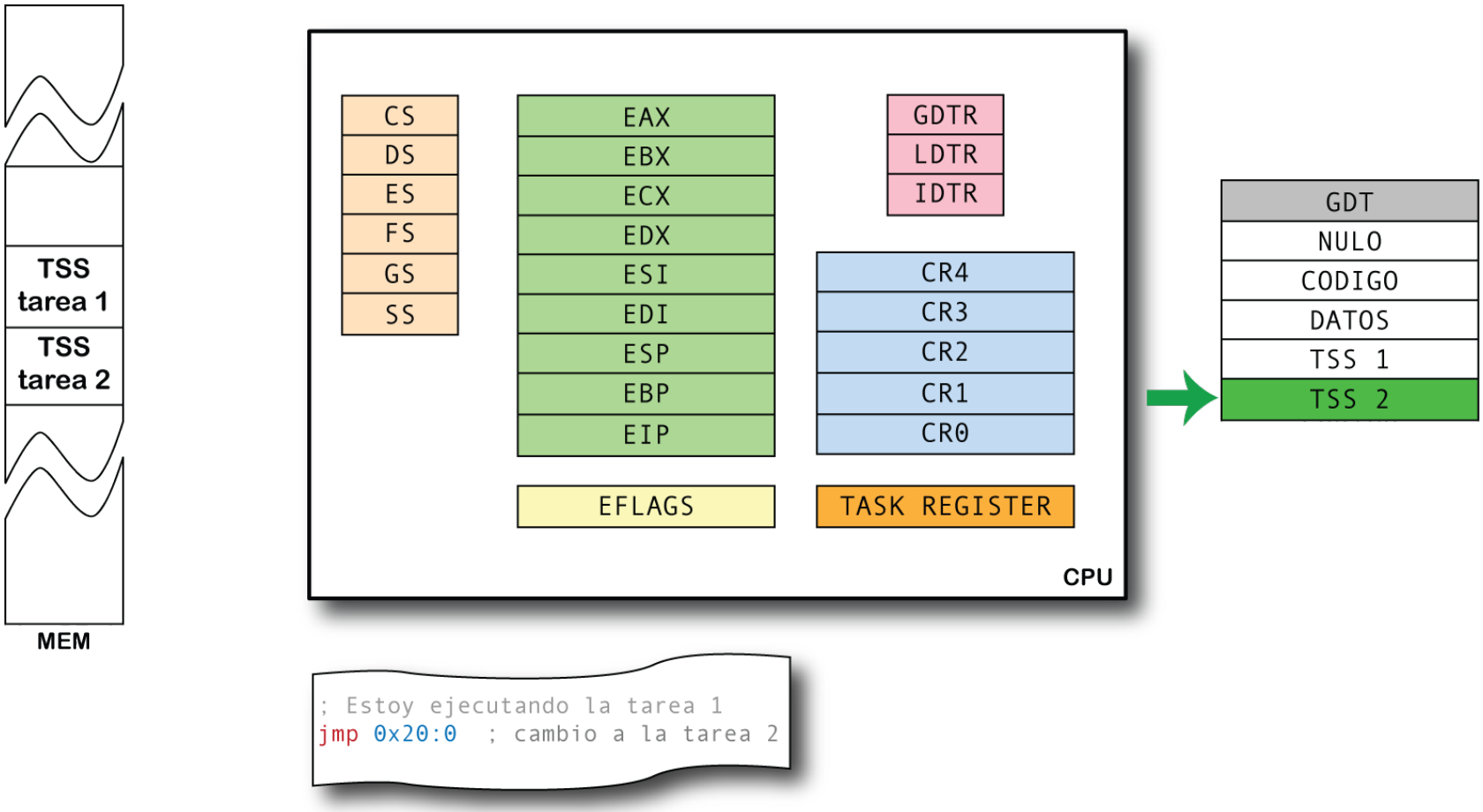
Conmutación de Tareas

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



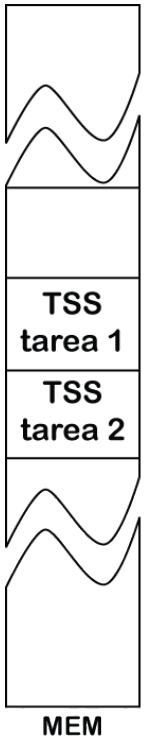
Conmutación de Tareas

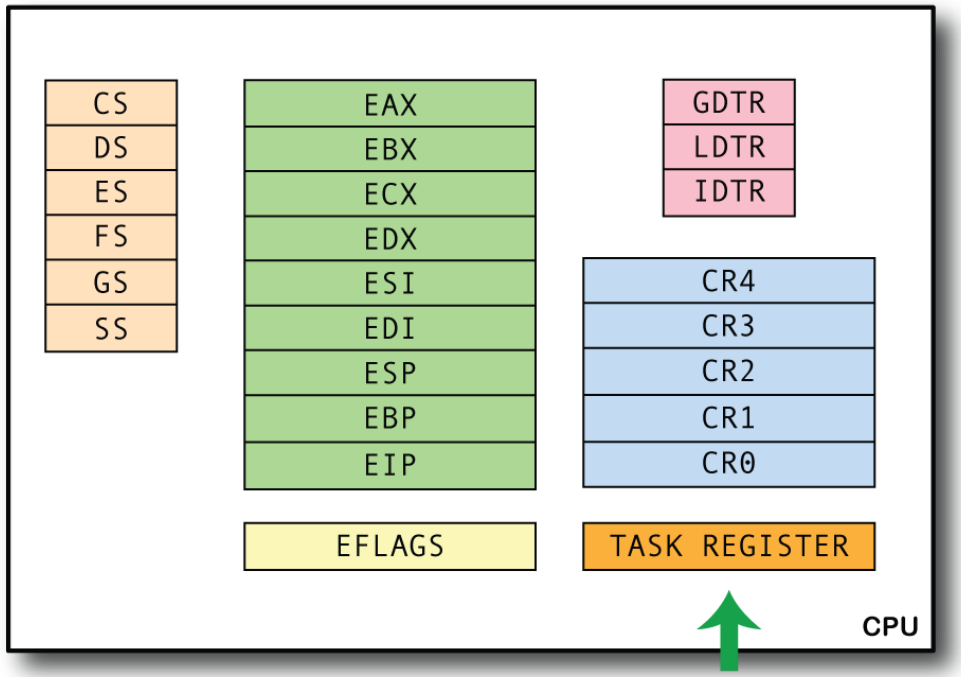
2. Se busca el descriptor correspondiente en la GDT.

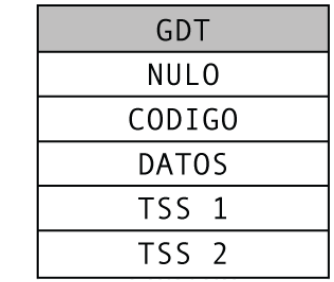


Conmutación de Tareas

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



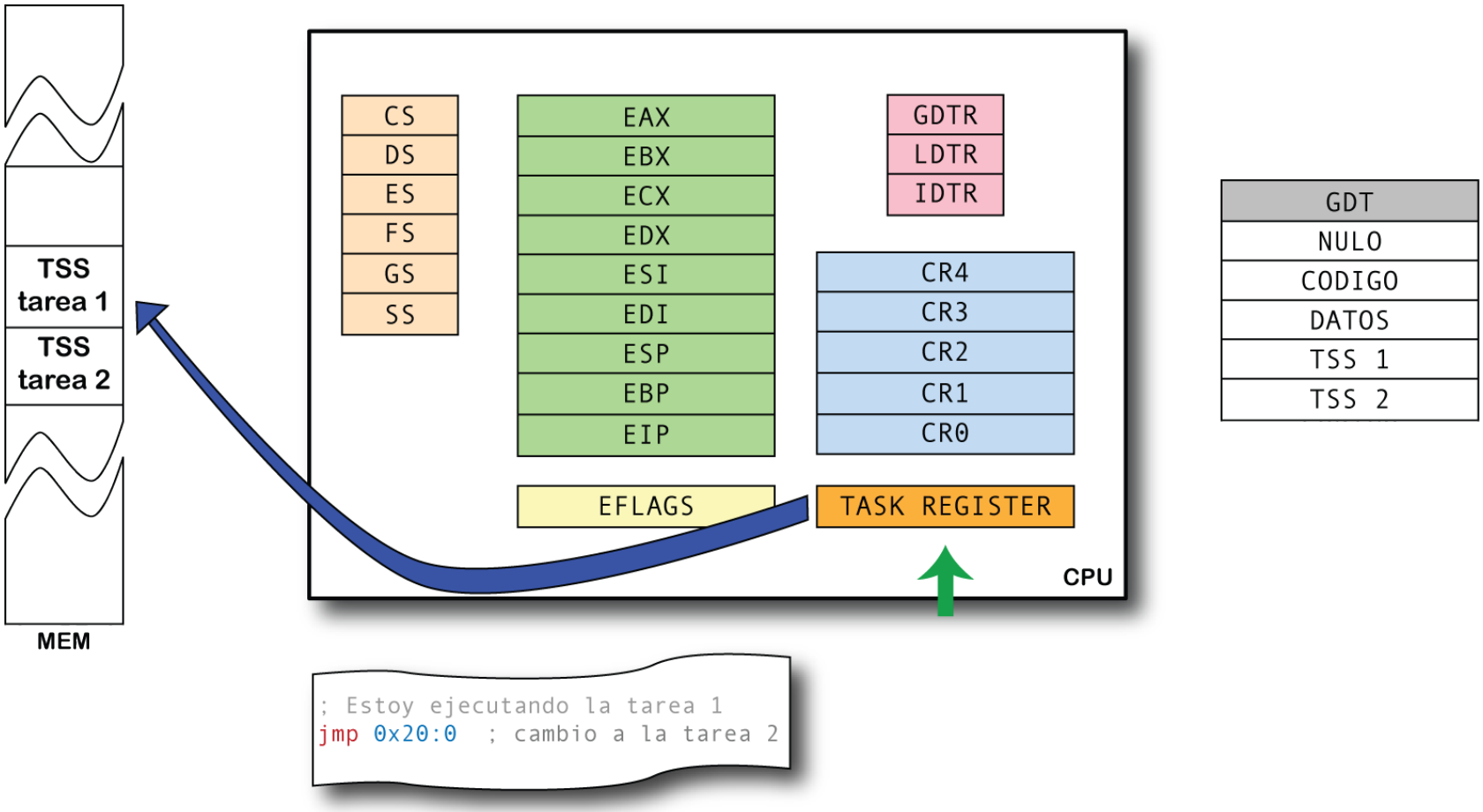





```
; Estoy ejecutando la tarea 1
jmp 0x20:0  ; cambio a la tarea 2
```

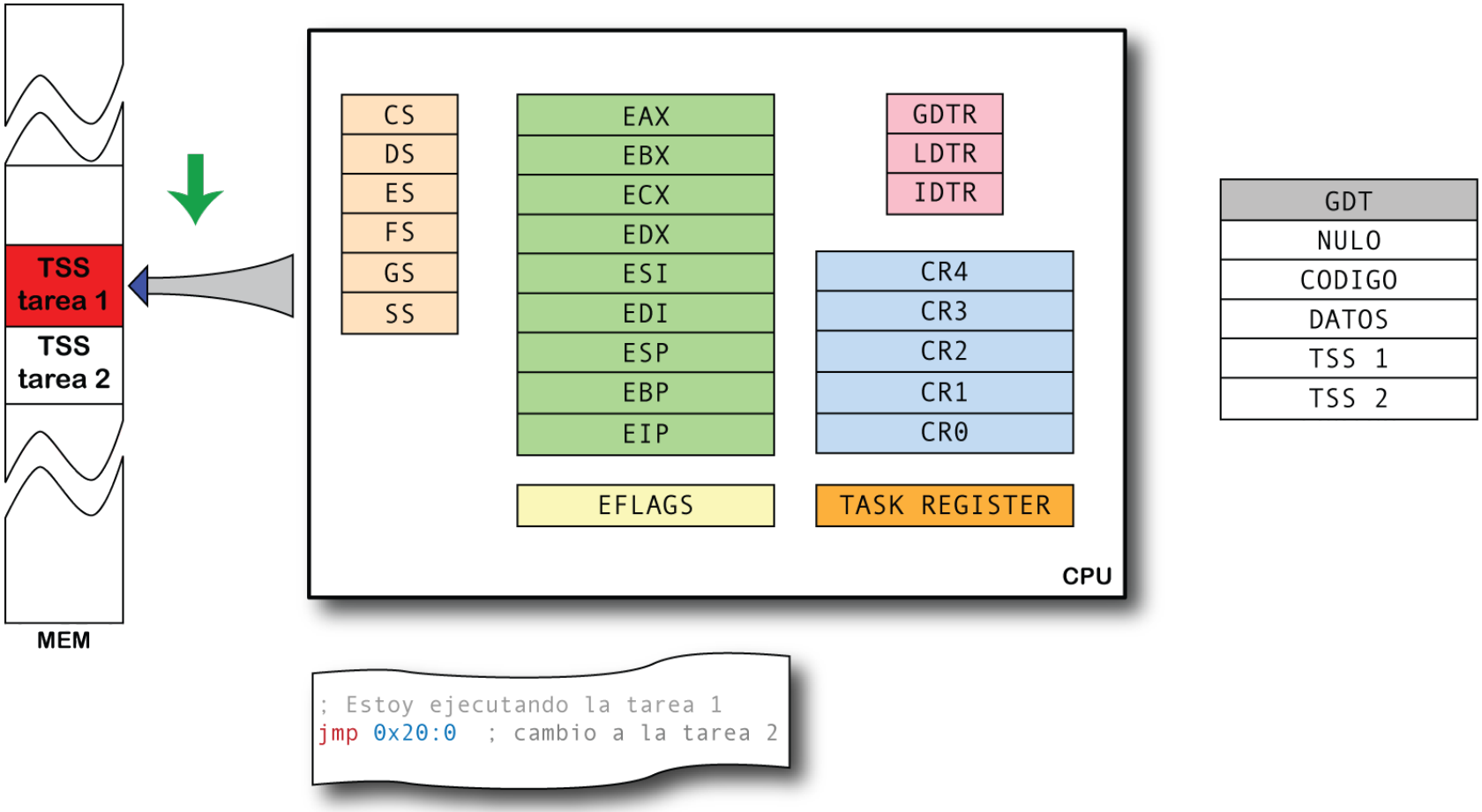
Conmutación de Tareas

4. Se busca el TSS apuntado por el TR.



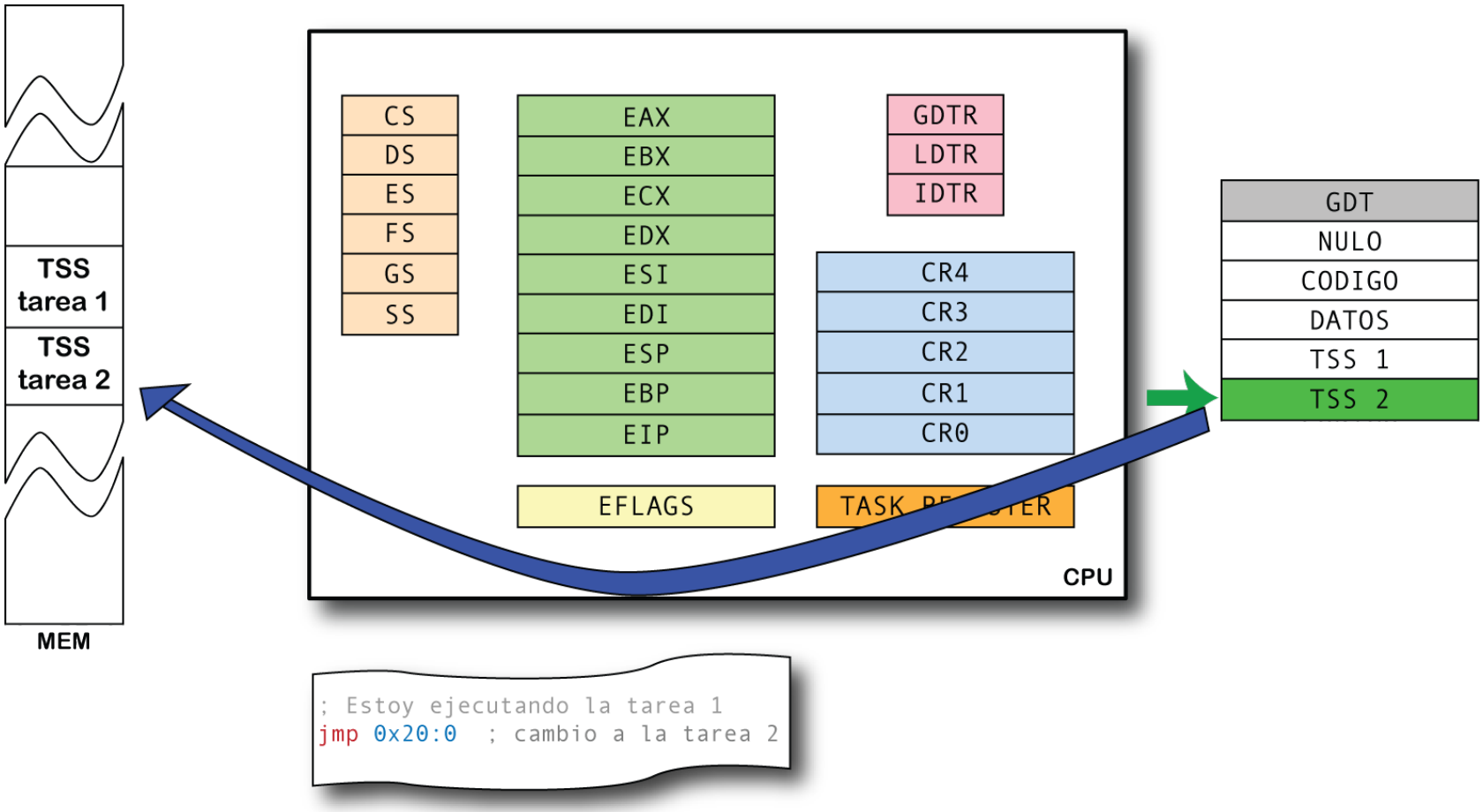
Conmutación de Tareas

5. Se guarda el contexto actual.



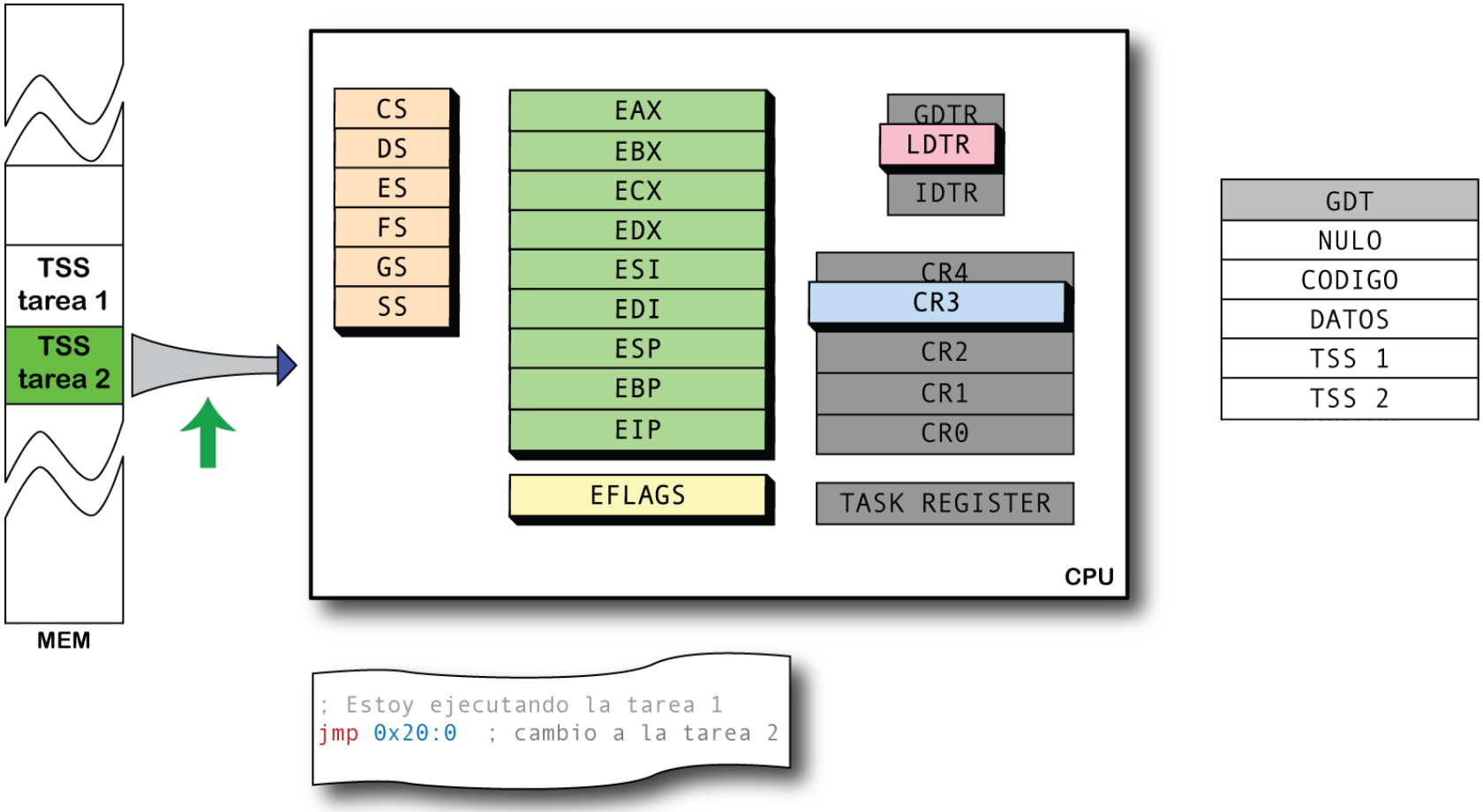
Conmutación de Tareas

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



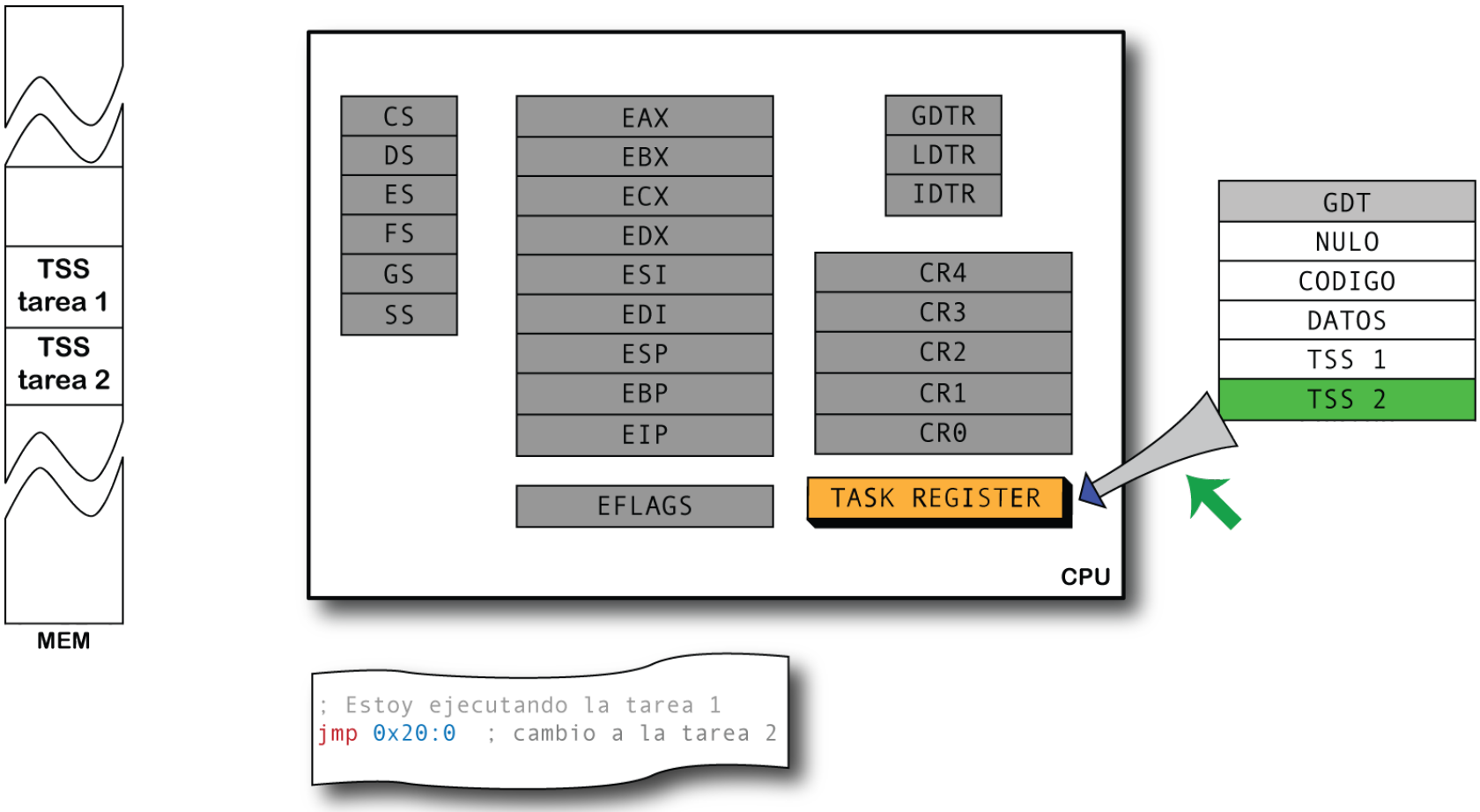
Conmutación de Tareas

7. Se obtiene el nuevo contexto.



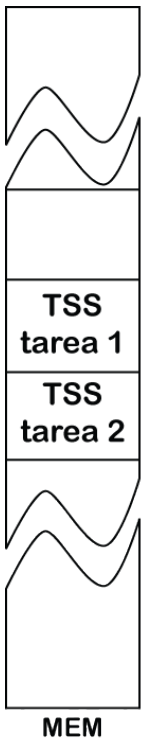
Conmutación de Tareas

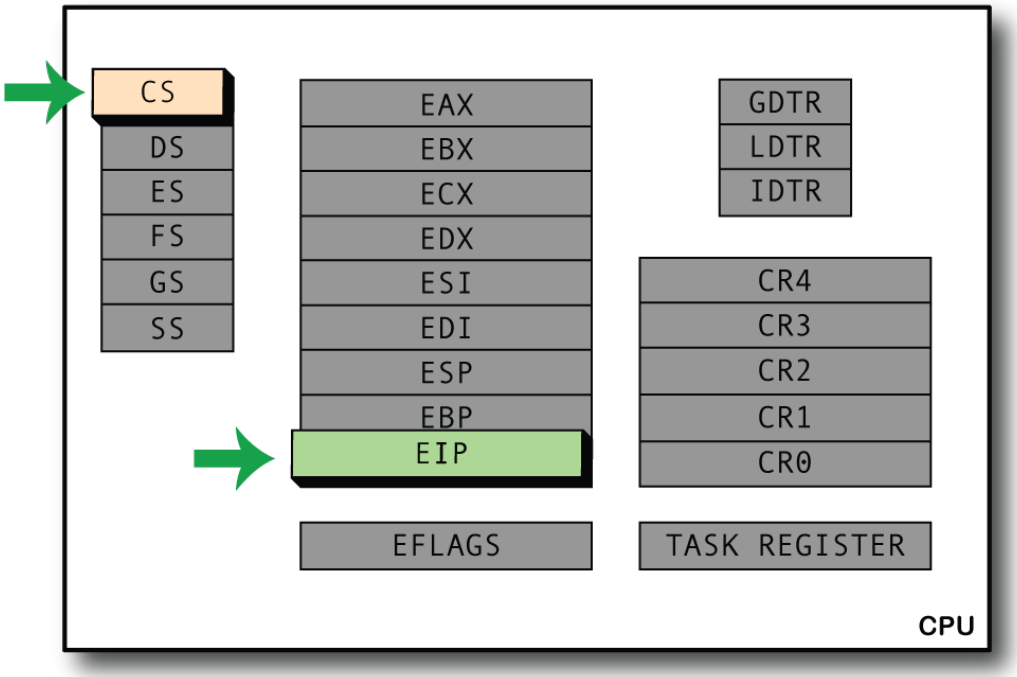
8. Se actualiza el TR.

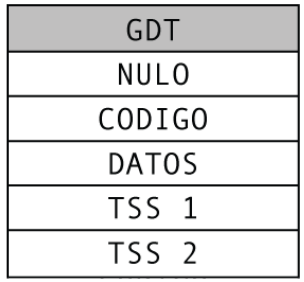


Conmutación de Tareas

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

1. Definir las entradas en la `GDT` que considere necesarias para ser usadas como descriptores de `TSS`. Minimamente, una para ser utilizada por la `tarea_inicial` y otra para la tarea Idle.

2. Completar la entrada de la `TSS` de la tarea Idle con la información de la tarea `Idle`. Esta información se encuentra en el archivo `TSS.C`. La tarea `Idle` se encuentra en la dirección `0x00016000`. La pila se alojará en la misma dirección que la pila del kernel y será mapeada con *identity mapping*. Esta tarea ocupa 1 pagina de 4KB y debe ser "mapeada" con *identity mapping*. Además la misma debe compartir el mismo `CR3` que el *kernel*.

3. Construir una función que complete una `TSS` libre con los datos correspondientes a una tarea. El código de las tareas se encuentra a partir de la dirección `0x00010000` ocupando una pagina de 4kb cada una. Para la dirección de la pila se debe utilizar el mismo espacio de la tarea, la misma crecerá desde la base de la tarea. Recordar que las tareas asumen que se habrán apilado sus 2 argumentos y luego una dirección de retorno. Para el mapa de memoria se debe construir uno nuevo utilizando la función `mmu_inicializar_dir_perro`. Además, tener en cuenta que cada tarea utilizará una pila distinta de nivel 0, para esto se debe pedir una nueva pagina libre a tal fin.

El TP3

Ejercicio 6

4. Completar la entrada de la `GDT` correspondiente a la tarea `Idle`.

5. Escribir el código necesario para ejecutar la tarea `Idle`, es decir, saltar intercambiando las `TSS`, entre la `tarea_inicial` y la tarea `Idle`.

6. Modificar la rutina de la interrupción `0x46`, para que implemente los servicios según se indica en la sección **??**, sin desalojar a la tarea que realiza el syscall.

7. Ejecutar una tarea `perro` manualmente. Es decir, crearla y saltar a la entrada en la gdt de su respectiva `TSS`

Nota: En `tss.c` están definidas las `tss` como estructuras `TSS`. Trabajar en `tss.c` y `kernel.asm`.

El TP3

Archivos para el ejercicio:

- `gdt.h` / `gdt.c`
  definición de la GDT
- `kernel.asm`
  código principal
- `tss.h` / `tss.c`
  definición estructura auxiliar de contextos y de TSS

Pequeños consejos

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** algun valor válido para guardar contexto.

Pequeños consejos: cómo completar una TSS

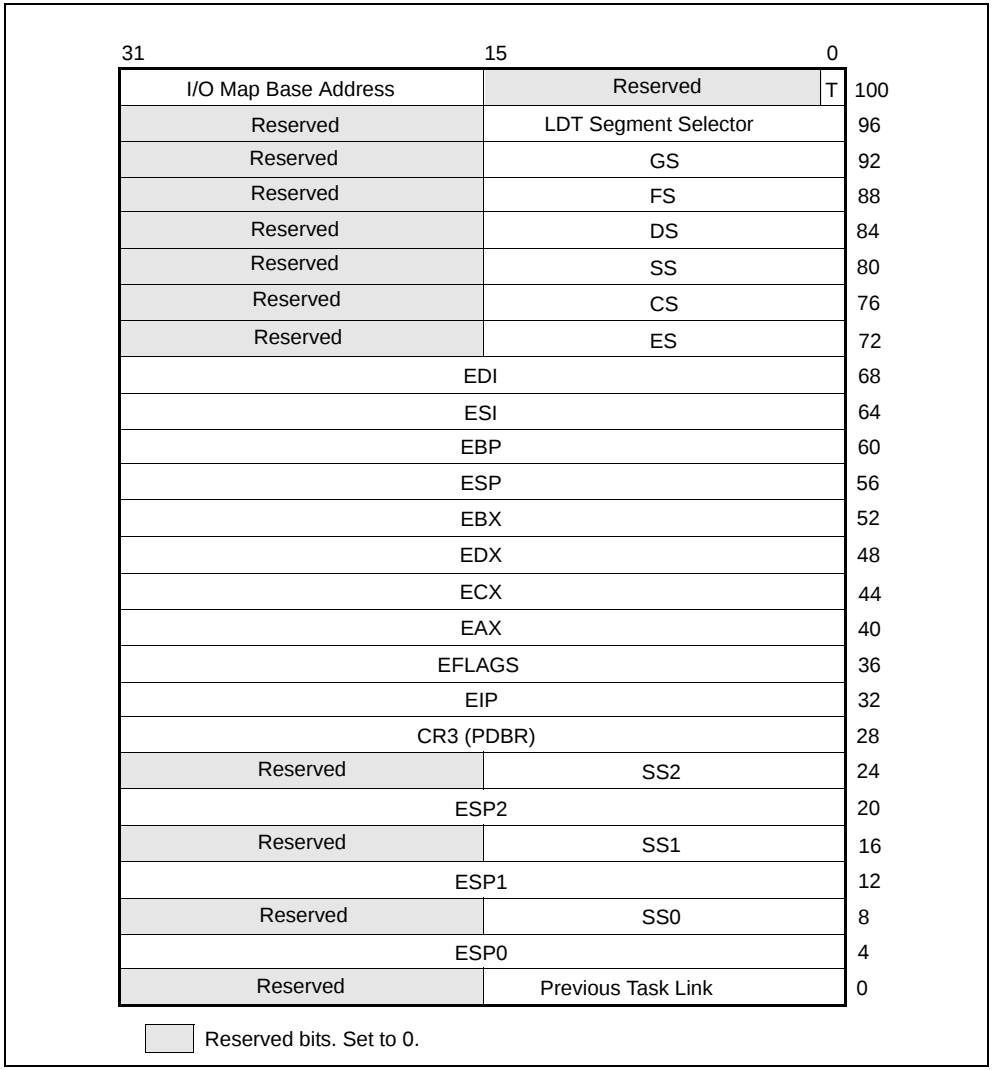


Figure 7-2. 32-Bit Task-State Segment (TSS)

```
mov edi,<inicioTSSs>
add edi,104*<indice>
mov eax,cr3
mov [edi+28],eax
mov dword [edi+32],<eip>
mov dword [edi+36],<flags>
mov dword [edi+56],<pila>
mov dword [edi+60],<pila>
mov word [edi+72],<seg.dat>
mov word [edi+76],<seg.cod>
mov word [edi+80],<seg.dat>
mov word [edi+84],<seg.dat>
mov word [edi+88],<seg.dat>
mov word [edi+92],<seg.dat>
mov word [edi+102],0xFFFF
```

Pequeños consejos: EFLAGS

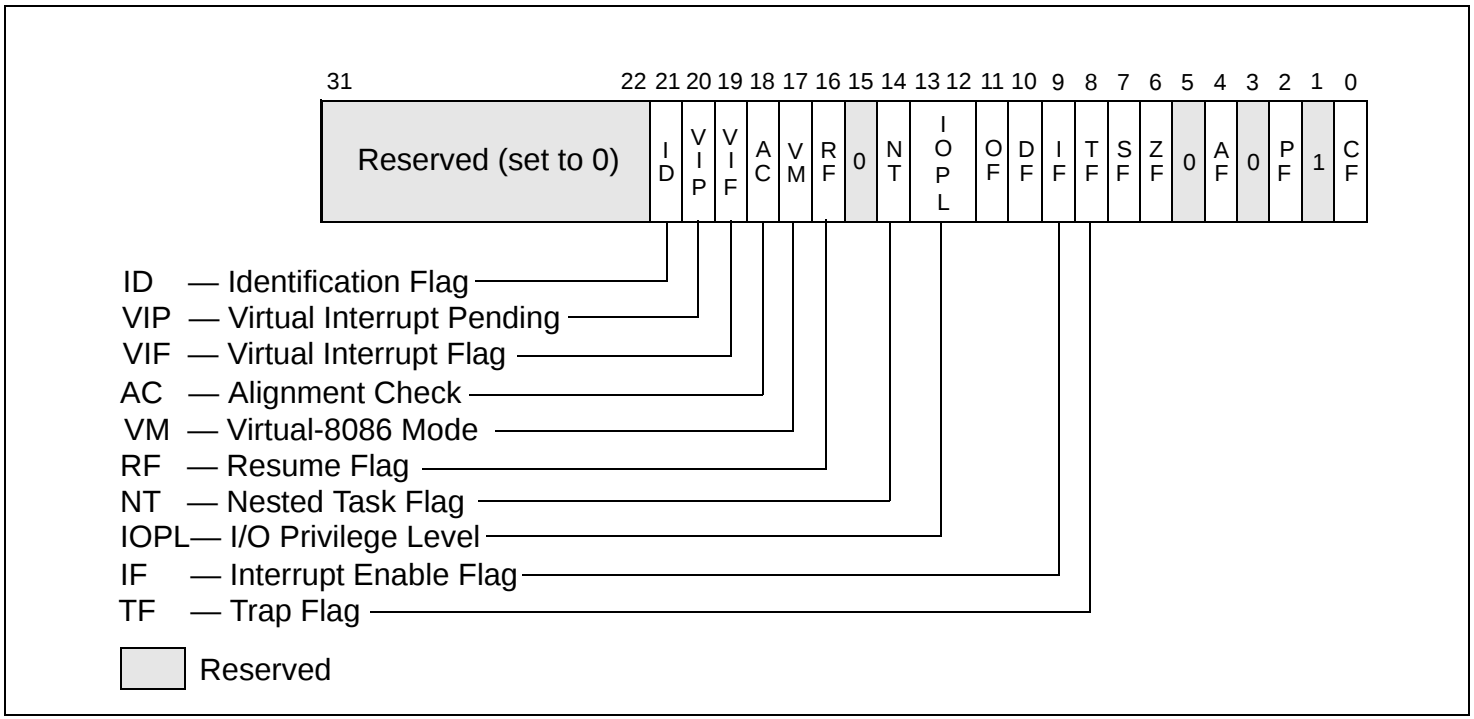
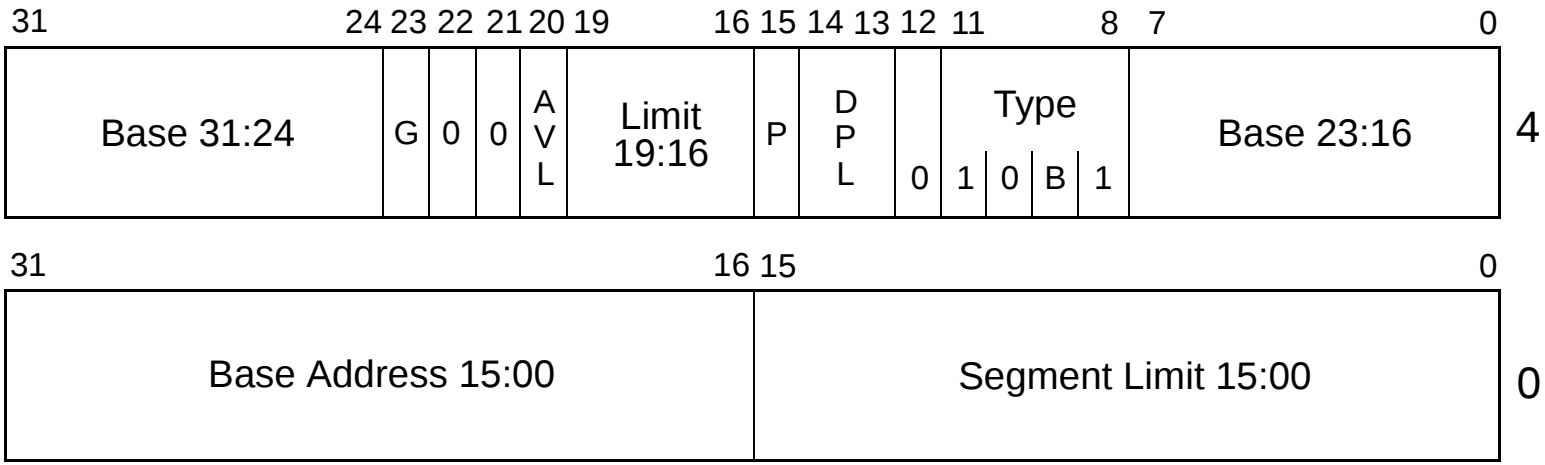


Figure 2-4. System Flags in the EFLAGS Register

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

Pequeños consejos: y el descriptor TSS?

- Supongo que `EDI` tiene la dirección de la `TSS`




```
mov esi, gdt
add esi, <indice>
mov word [esi], 0x67
mov word [esi+2], di
mov edx, edi
```

```
shr edx, 16
mov byte [esi+4], dl
mov byte [esi+5], 10001001b
mov byte [esi+6], 00010000b
mov byte [esi+7], dh
```

Pequeños consejos: intercambiando tareas

- Cargar la tarea inicial

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

- Cargar la nueva tarea

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