Clase pre-parcial

Organización del Computador 2
Segundo cuatrimestre 2023

Ejercicio 1 - 2c 2023 recu

En un sistema como el desarrollado en los talleres se desea modificar la forma en que las tareas se muestran en la pantalla. En lugar de realizar la syscall **`draw`** (previamente implementada como **`int 88`**) se quiere que **las tareas tengan acceso directo a la memoria de video**:

- La memoria física de video (**`0xB8000-0xB9FFF`**) es la que se refleja en la pantalla real
- Sólo puede haber una única tarea en un determinado momento con la memoria física de video (**`0xB8000-0xB9FFF`**) mapeada
- El resto de las tareas tendrá asignadas una pantalla dummy (falsa) en la región física (**`0x1E000-0x1FFFF`**)
- La memoria de video de una tarea se mapeará siempre en el rango virtual **`0x08004000-0x08005FFF`**, independientemente si tiene acceso a la pantalla real o no

Ejercicio 1 - 2c 2023 recu

Con el diseño propuesto hay una única tarea *actualmente en pantalla* (con acceso a la memoria física de video). Las tareas que no se encuentren en pantalla podrán escribir a las direcciones virtuales de video pero verán sus escrituras reflejadas en la *pantalla virtual compartida* (dummy).

Soltar la tecla TAB cambiará la tarea *actualmente en pantalla*. Los cambios de tarea en pantalla se realizarán de manera cíclica (T1-T2-T3-T4-T1-T2-. . .).

Se solicita describir los cambios requeridos para implementar esta nueva característica.

Ejercicio 1 - 2c 2023 recu - Preguntas

a. Dibuje el esquema de memoria virtual de las tareas del nuevo sistema.
b. Describa los cambios al proceso de creación de tareas (**`init_task_dir`**, **`create_task`**, **`sched_add_task`**, etc. . .). Muestre código y/o pseudocódigo.
c. Explique qué mecanismo usará para que el sistema sepa *a qué tarea le toca* el acceso a la pantalla.
d. Describa los cambios necesarios para realizar el *cambio de pantalla* al **soltar** la tecla TAB. Proponga una implementación posible mostrando código y/o pseudocódigo.
e. En el mecanismo propuesto las tareas no tienen forma sencilla de saber si “es su turno” de usar la pantalla. Proponga una solución. No se pide código ni pseudocódigo, sólo la idea.
f. En el mecanismo propuesto la tarea debe redibujar toda su pantalla cuando logra conseguir acceso a la misma. ¿Cómo podría evitarse eso? No se pide código ni pseudocódigo, sólo la idea.

Ejercicio 1 - 2c 2023 recu - Resolución conceptual

Nos piden un montón de cosas, busquemos qué hay que resolver sí o sí:

- (a, b) Agregar un mapping al esquema de memoria de las tareas
- (c) Poder *intercambiar* la pantalla entre una tarea y *la que le sigue*
- (c) Al arrancar el sistema: determinar quién tiene *la primer pantalla*
  - Una tarea tiene la pantalla *posta*
  - El resto tienen la pantalla *dummy*
- (d) Detectar que se apretó TAB
- (e) Agregar un mecanismo para que una tarea pueda responder: *¿Quién tiene la pantalla?*
- (f) Agregar una forma de que las tareas no se *choquen* en la dummy

Ejercicio 1 - 2c 2023 recu - Esquema de memoria

Agregar un mapping al esquema de memoria de las tareas

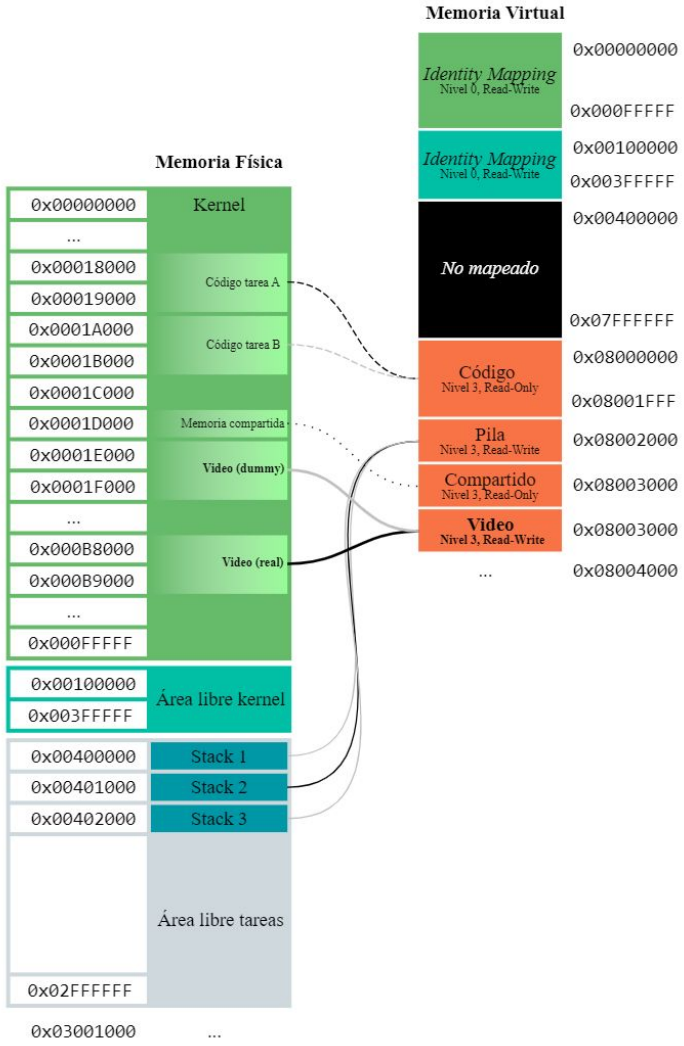
- ¿Dónde se define el esquema de memoria?
- ¿Cuál es el valor inicial para **phy**?
- ¿Cómo queda la memoria?

Ejercicio 1 - 2c 2023 recu - Esquema de memoria

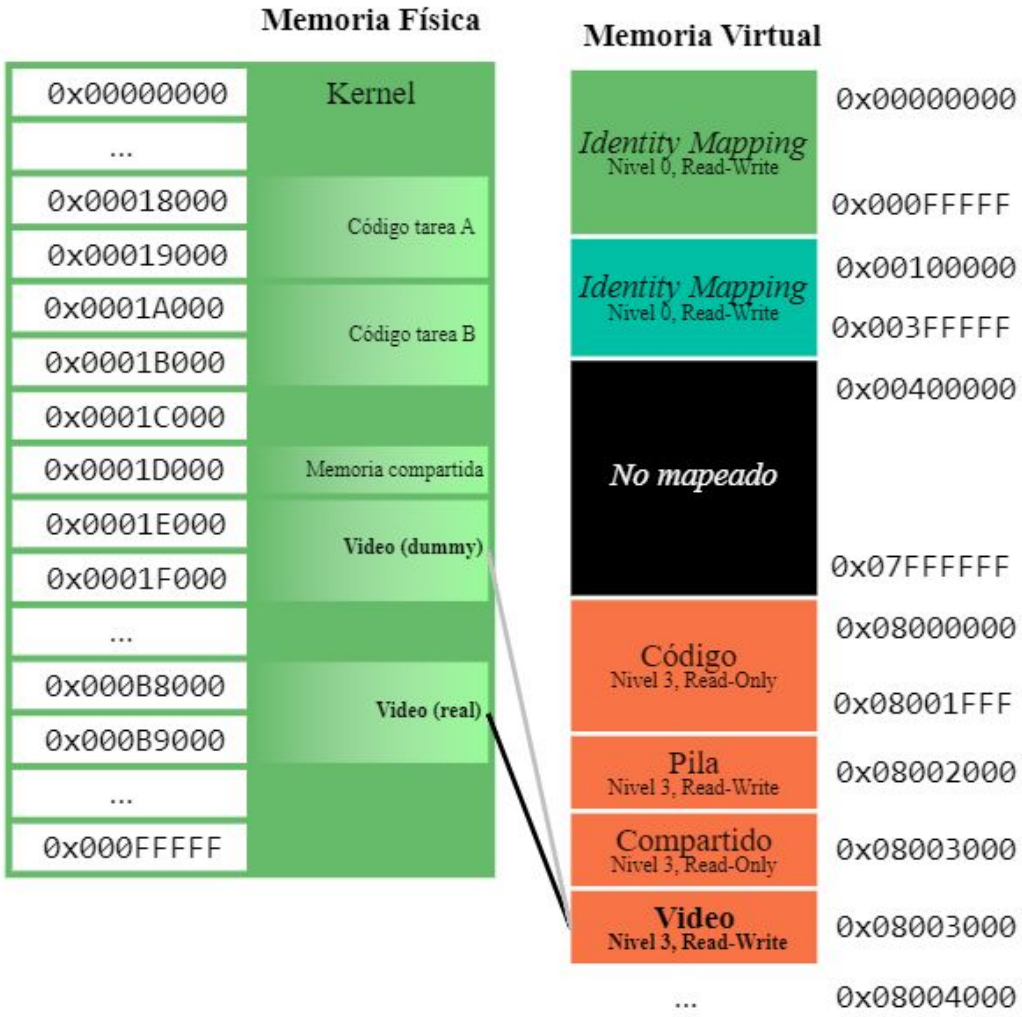
Agregar un mapping al esquema de memoria de las tareas

- ¿Dónde se define el esquema de memoria?
  En **`init_task_dir`**. Alcanza con agregar un **`mmu_map_page`** ahí.
- ¿Cuál es el valor inicial para **`phy`**?
  Probemos con *dummy* :)
- ¿Cómo queda la memoria?
  Tenemos dibujito

1a


Memoria Virtual
Identity Mapping
Nivel 0, Read-Write
0x00000000
0x000FFFFF
Identity Mapping
Nivel 0, Read-Write
0x00100000
0x003FFFFF
No mapeado
0x00400000
0x07FFFFFF
Código
Nivel 3, Read-Only
0x08000000
0x08001FFF
Pila
Nivel 3, Read-Write
0x08002000
Compartido
Nivel 3, Read-Only
0x08003000
Video
Nivel 3, Read-Write
0x08003000
0x08004000
Memoria Física
0x00000000
Kernel
0x00018000
0x00019000
Código tarea A
0x0001A000
0x0001B000
Código tarea B
0x0001C000
0x0001D000
Memoria compartida
0x0001E000
0x0001F000
Video (dummy)
0x000B8000
0x000B9000
Video (real)
0x000FFFFF
0x00100000
0x003FFFFF
Área libre kernel
0x00400000
Stack 1
0x00401000
Stack 2
0x00402000
Stack 3
Área libre tareas
0x02FFFFFF
0x03001000

1a


Memoria Física
0x00000000
Kernel
...
0x00018000
Código tarea A
0x00019000
0x0001A000
Código tarea B
0x0001B000
0x0001C000
0x0001D000
Memoria compartida
0x0001E000
Video (dummy)
0x0001F000
...
0x000B8000
Video (real)
0x000B9000
...
0x000FFFFF
Memoria Virtual
Identity Mapping
Nivel 0, Read-Write
0x00000000
0x000FFFFF
Identity Mapping
Nivel 0, Read-Write
0x00100000
0x003FFFFF
No mapeado
0x00400000
0x07FFFFFF
Código
Nivel 3, Read-Only
0x08000000
0x08001FFF
Pila
Nivel 3, Read-Write
0x08002000
Compartido
Nivel 3, Read-Only
0x08003000
Video
Nivel 3, Read-Write
0x08003000
...
0x08004000

Ejercicio 1 - 2c 2023 recu - Esquema de memoria

mmu.c

```
paddr_t mmu_init_task_dir(paddr_t phy_start) {
  /* … */
  // Mappeo código, stack y shared
  mmu_map_page(cr3, TASK_CODE_VIRTUAL + PAGE_SIZE * 0, phy_start,
               MMU_P | MMU_U);
  mmu_map_page(cr3, TASK_CODE_VIRTUAL + PAGE_SIZE * 1, phy_start + PAGE_SIZE,
               MMU_P | MMU_U);
  mmu_map_page(cr3, TASK_CODE_VIRTUAL + PAGE_SIZE * 2, stack,
               MMU_P | MMU_U | MMU_W);
  mmu_map_page(cr3, TASK_CODE_VIRTUAL + PAGE_SIZE * 3, SHARED,
               MMU_P | MMU_U);
  // Mappeo video dummy (2 páginas)
  mmu_map_page(cr3, 0x08004000, 0x1E000, MMU_P | MMU_U | MMU_W);
  mmu_map_page(cr3, 0x08005000, 0x1F000, MMU_P | MMU_U | MMU_W);
}
```

Ejercicio 1 - 2c 2023 recu - Intercambio de pantallas

Poder *intercambiar* la pantalla entre una tarea y *la que le sigue*:

- Esto se parece peligrosamente al scheduler. ¿Podemos usar ideas de ahí?

Idea	Scheduling de tareas	Scheduling de pantallas
Tarea actual	`int8_t current_task;`	`int8_t current_video_task;`
Inicializar tareas	`void sched_init();`	`void video_init();`
Cambiar de tarea	`int de clock (_isr32)` `+ sched_next_task();`	`int de teclado (_isr33)` `+ swap_video_page();`

- No nos importa si la tarea está detenida o no.
- No nos piden considerar el problema de agregar tareas.

Ejercicio 1 - 2c 2023 recu - Intercambio de pantallas

Poder *intercambiar* la pantalla entre una tarea y *la que le sigue*:

Cambiar de tarea son tres cosas:

1. La tarea vieja pasa a ver la *dummy* en su esquema de paginación
2. La tarea nueva pasa a ver la *posta* en su esquema de paginación
3. Se actualiza **`current_video_task`**

Ejercicio 1 - 2c 2023 recu - Intercambio de pantallas

video.c

```c
int8_t current_video_task; // Variable global

void swap_video_page() {
  int next_video_task = (current_video_task + 1) % MAX_TASKS;
  paddr_t cr3_current = tss_tasks[current_video_task].cr3;
  paddr_t cr3_next = tss_tasks[next_video_task].cr3;
  // La tarea actual pasa a la dummy
  mmu_map_page(cr3_current, 0x08004000, 0x1E000, MMU_P | MMU_U | MMU_W);
  mmu_map_page(cr3_current, 0x08005000, 0x1F000, MMU_P | MMU_U | MMU_W);
  // La siguiente tarea pasa a la posta
  mmu_map_page(cr3_next, 0x08004000, 0xB8000, MMU_P | MMU_U | MMU_W);
  mmu_map_page(cr3_next, 0x08005000, 0xB9000, MMU_P | MMU_U | MMU_W);
  // Y ahora la tarea nueva es la actual :)
  current_video_task = next_video_task;
}
```

Ejercicio 1 - 2c 2023 recu - Primer pantalla

Al arrancar el sistema: determinar quién tiene *la primer pantalla*

Alguien tiene que ser la primera en tener la pantalla. Dos opciones:

- **Opción 1:** **`init_task_dir`** toma un parámetro **`es_la_primera_tarea`**. Tenemos que modificar todo lo que use **`init_task_dir`**
- **Opción 2:** Todas las tareas arrancan con la *dummy* y en **`kernel.asm`** le damos la pantalla posta a alguna tarea después de crearla

Ejercicio 1 - 2c 2023 recu - Primer pantalla

video.c

```c
int8_t current_video_task; // Variable global

// NOTA: En algún lugar de kernel.asm vamos a tener que llamar a video_init!
paddr_t video_init() {
  int first_task = 0;
  paddr_t cr3 = tss_tasks[first_task].cr3;
  current_video_task = first_task;
  // Mappeo video posta (2 páginas)
  mmu_map_page(cr3, 0x08004000, 0xB8000, MMU_P | MMU_U | MMU_W);
  mmu_map_page(cr3, 0x08005000, 0xB9000, MMU_P | MMU_U | MMU_W);
}
```

Ejercicio 1 - 2c 2023 recu - Responder al teclado

- Detectar que se apretó TAB

  Es agregar un **`if`** en el código que procesa los eventos de teclado.

  El enunciado dice que soltar TAB es el scancode **`0x8F`**.

Ejercicio 1 - 2c 2023 recu - Responder al teclado

- Detectar que se apretó TAB

  Es agregar un **`if`** en el código que procesa los eventos de teclado.

  El enunciado dice que soltar TAB es el scancode **`0x8F`**.

isr.asm

```
global _isr33
_isr33: ;rutina de atención del teclado
  pushad
  ; 1. Le decimos al PIC que vamos a atender la interrupción
  call pic_finish1
  ; 2. Leemos la tecla desde el teclado y la procesamos
  in al, 0x60
  push eax
  cmp al, 0x8F
  jne .no_es_soltar_tab
  call swap_video_page

  .no_es_soltar_tab:
  call tasks_input_process
  add esp, 4
  popad
  iret
```

Ejercicio 1 - 2c 2023 recu - `current_video_task`

Agregar un mecanismo para que una tarea pueda responder: *¿Quién tiene la pantalla?*

- Uno diría *“Ya tenemos* ***`current_video_task`*** *¿Qué más necesitamos?”*
- Pero **`current_video_task`** es una variable global en el kernel
- Hay varias soluciones:
  - Agregar una syscall
  - Exponer **`current_video_task`** en la página compartida
  - ¿Alguna otra?

Ejercicio 1 - 2c 2023 recu - Múltiples *dummy*

Agregar una forma de que las tareas no se *choquen* en la *dummy*

- El problema es que todas las tareas comparten la misma *dummy*
- Podría no ser así: Cada tarea tiene su propia dummy. Luego el cambio de pantallas:
  1. Guarda la pantalla actual
  2. Restaura la pantalla de la próxima tarea

Ejercicio 1 - 2c 2023 recu - Múltiples *dummy*

video.c

```
void swap_video_page() {
  /* ... */
  // La tarea actual pasa a la dummy
  copy_page(dummy_video_page_de(current_video_task), 0xB8000);
  copy_page(dummy_video_page_de(current_video_task) + PAGE_SIZE, 0xB9000);
  mmu_map_page(cr3_current, 0x08004000, 0x1E000, MMU_P | MMU_U | MMU_W);
  mmu_map_page(cr3_current, 0x08005000, 0x1F000, MMU_P | MMU_U | MMU_W);
  // La siguiente tarea pasa a la posta
  copy_page(0xB8000, dummy_video_page_de(next_video_task));
  copy_page(0xB9000, dummy_video_page_de(next_video_task) + PAGE_SIZE);
  mmu_map_page(cr3_next, 0x08004000, 0xB8000, MMU_P | MMU_U | MMU_W);
  mmu_map_page(cr3_next, 0x08005000, 0xB9000, MMU_P | MMU_U | MMU_W);
  /* ... */
}
```

Preguntas?

Ahora pasamos al Ej. 2

Ejercicio 2 - 1c 2023 recu

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes `0x0F 0x0B`**.**

Tenemos un montón de software escrito para el ENTEL575 pero lamentablemente no poseemos hardware que lo pueda correr. ¿Podrías desarrollar un sistema que nos permita hacerlo?

Ejercicio 2 - 1c 2023 recu

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes *0x0F 0x0B*.

Tenemos un montón de software escrito para el ENTEL575 pero lamentablemente no poseemos hardware que lo pueda correr. ¿Podrías desarrollar un sistema que nos permita hacerlo? Para ello, respondé los siguientes puntos:

A. ¿**Qué excepción** ocurre cuándo un procesador x86 intenta ejecutar una instrucción no soportada?

Ejercicio 2 - 1c 2023 recu

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes *`0x0F 0x0B`*.

Tenemos un montón de software escrito para el ENTEL575 pero lamentablemente no poseemos hardware que lo pueda correr. ¿Podrías desarrollar un sistema que nos permita hacerlo? Para ello, respondé los siguientes puntos:

A. ¿**Qué excepción** ocurre cuándo un procesador x86 intenta ejecutar una instrucción no soportada?
B. Realice un **diagrama de pila** que muestre el estado de la pila del kernel luego de que una aplicación de usuario intentó ejecutar RSTL00P.

Ejercicio 2 - 1c 2023 recu

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes *0x0F 0x0B*.

Tenemos un montón de software escrito para el ENTEL575 pero lamentablemente no poseemos hardware que lo pueda correr. ¿Podrías desarrollar un sistema que nos permita hacerlo? Para ello, respondé los siguientes puntos:

A. ¿**Qué excepción** ocurre cuándo un procesador x86 intenta ejecutar una instrucción no soportada?
B. Realice un **diagrama de pila** que muestre el estado de la pila del kernel luego de que una aplicación de usuario intentó ejecutar RSTL00P.
C. ¿Qué **dirección de retorno** se encuentra en la pila al atender la excepción?
D. Describa una posible implementación de RSTL00P utilizando el mecanismo descrito en (a) y (b).
    - El mecanismo propuesto **sólo debe actuar** cuándo la instrucción no soportada es RSTL00P.
    - Si la instrucción que generó la excepción **no es RSTL00P la tarea debe ser deshabilitada** y la ejecución debe saltar a la tarea idle.
    - Si la instrucción que generó la excepción es RSTL00P adecúe la dirección de retorno de manera que permita a la tarea **continuar la ejecución** sin problemas.
E. ¿Qué ocurriría si no se adecuara la dirección de retorno luego de simular RSTL00P?
F. Detalle los **cambios a las estructuras** del sistema visto en el taller que haría para realizar la implementación descrita en (d).
G. Muestre código para la **rutina de atención de interrupciones** descrita en (d) y todo otro cambio de comportamiento que haya visto necesario.

Ejercicio 2

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes `0x0F 0x0B`**.**

Tenemos un montón de software escrito para el ENTEL575 pero lamentablemente no poseemos hardware que lo pueda correr. ¿Podrías desarrollar un sistema que nos permita hacerlo? Para ello, respondé los siguientes puntos:

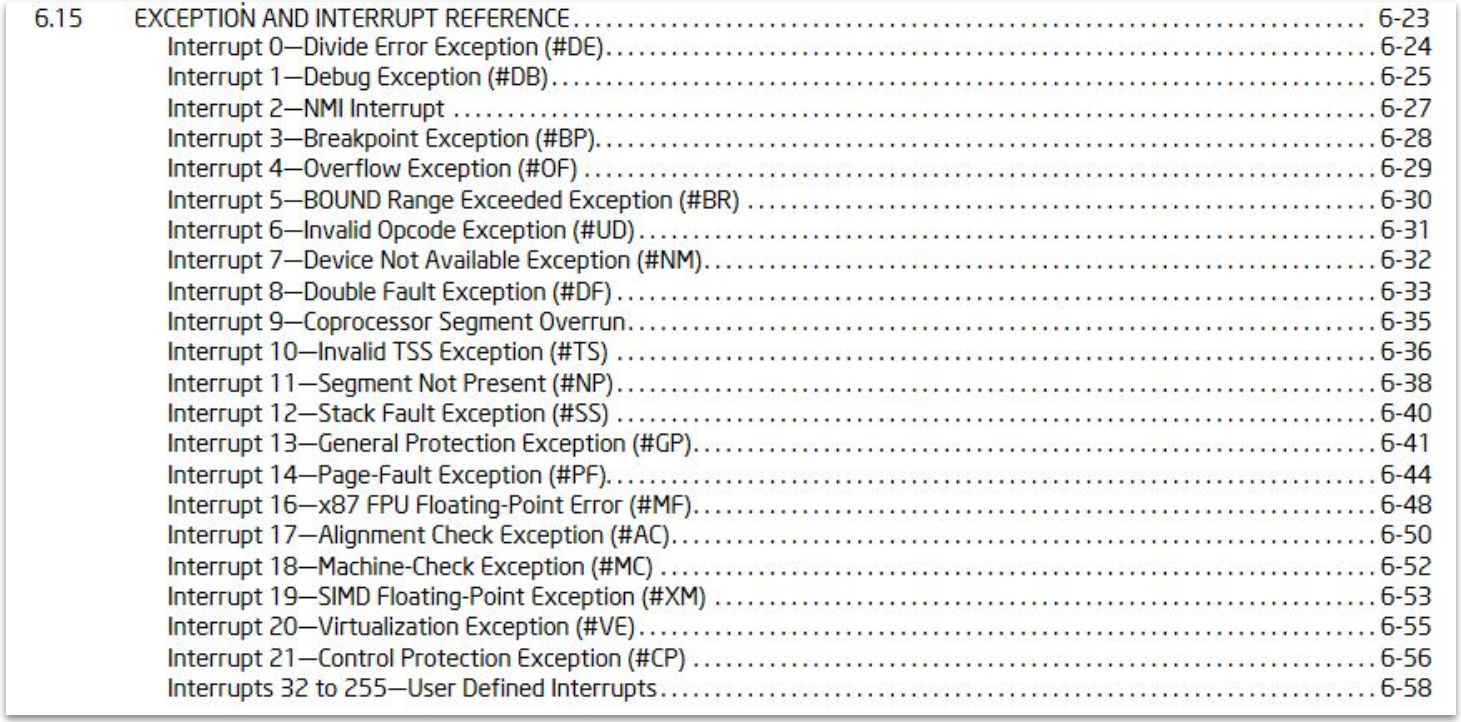
Recomendaciones:

- Lea el capítulo del manual sobre interrupciones y excepciones
- Revise con sumo cuidado el *"Exception and interrupt reference"*
- Repase el mecanismo de cambio de pila
- Recuerde los mecanismos que el procesador le ofrece para realizar cambios de tareas

Ejercicio 2

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes `0x0F 0x0B`**.**

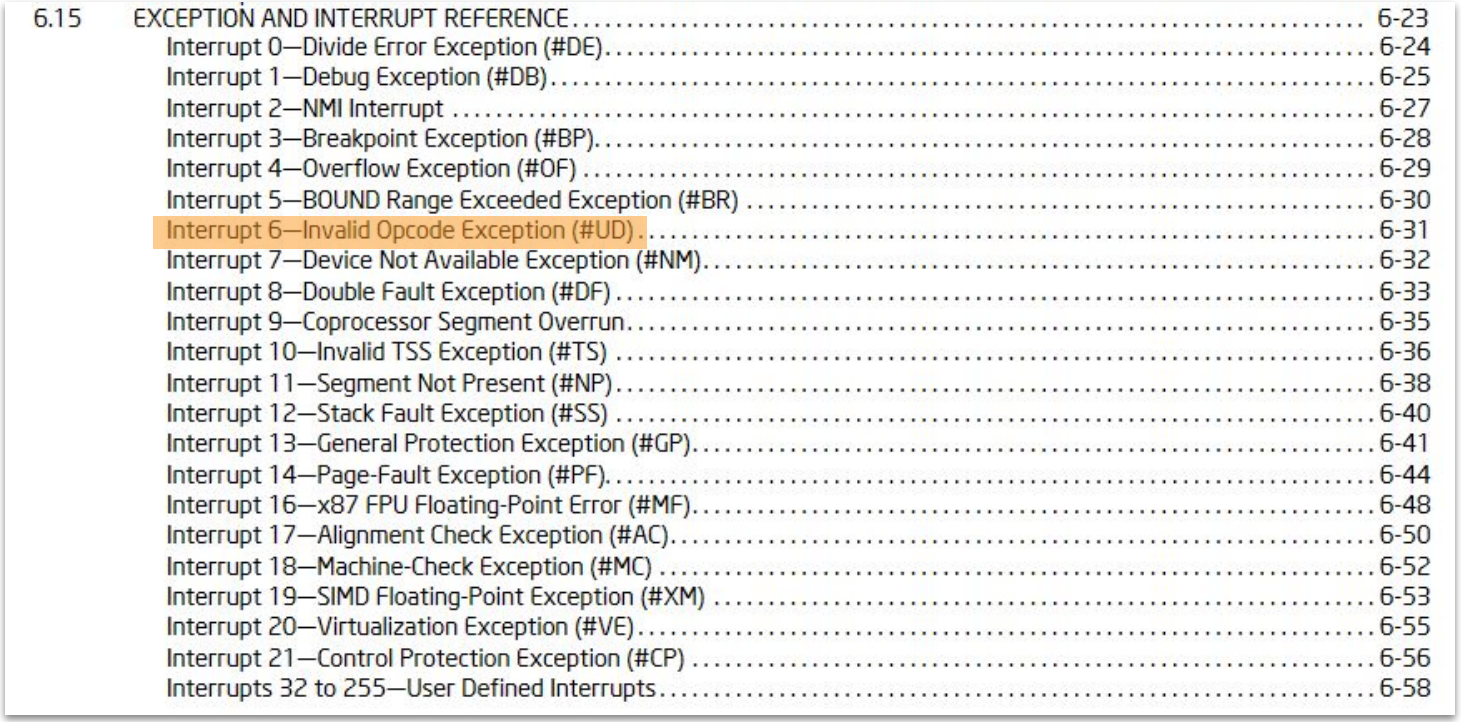
A. ¿**Qué excepción** ocurre cuándo un procesador x86 intenta ejecutar una instrucción no soportada?



Ejercicio 2

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes *0x0F 0x0B*.

A. ¿**Qué excepción** ocurre cuándo un procesador x86 intenta ejecutar una instrucción no soportada?



Ejercicio 2

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes `0x0F 0x0B`.

A. ¿**Qué excepción** ocurre cuándo un procesador x86 intenta ejecutar una instrucción no soportada?

Interrupt 6—Invalid Opcode Exception (#UD)

Exception Class Fault.

Description

Indicates that the processor did one of the following things:

- Attempted to execute an invalid or reserved opcode.
- Attempted to execute an instruction with an operand type that is invalid for its accompanying opcode; for example, the source operand for a LES instruction is not a memory location.
- Attempted to execute an MMX or SSE/SSE2/SSE3 instruction on an Intel 64 or IA-32 processor that does not support the MMX technology or SSE/SSE2/SSE3/SSSE3 extensions, respectively. CPUID feature flags MMX (bit 23), SSE (bit 25), SSE2 (bit 26), SSE3 (ECX, bit 0), SSSE3 (ECX, bit 9) indicate support for these extensions.
- Attempted to execute an MMX instruction or SSE/SSE2/SSE3/SSSE3 SIMD instruction (with the exception of the MOVNTI, PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, CLFLUSH, MONITOR, and MWAIT instructions) when the EM flag in control register CR0 is set (1).
- Attempted to execute an SSE/SE2/SSE3/SSSE3 instruction when the OSFXSR bit in control register CR4 is clear (0). Note this does not include the following SSE/SSE2/SSE3 instructions: MASKMOVQ, MOVNTQ, MOVNTI, PREFETCH*h*, SFENCE, LFENCE, MFENCE, and CLFLUSH; or the 64-bit versions of the PAVGB, PAVGW

Exception Error Code

None.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the instruction that generated the exception.

Program State Change

A program-state change does not accompany an invalid-opcode fault, because the invalid instruction is not executed.

Ejercicio 2

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes *`0x0F 0x0B`*.

B. Realice un **diagrama de pila** que muestre el estado de la pila del kernel luego de que una aplicación de usuario intentó ejecutar RSTL00P.

¿Qué ocurre cuando una tarea intenta ejecutar RSTLOOP?

Ejercicio 2

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes *0x0F 0x0B*.

B. Realice un **diagrama de pila** que muestre el estado de la pila del kernel luego de que una aplicación de usuario intentó ejecutar RSTL00P.

¿Qué ocurre cuando una tarea intenta ejecutar RSTLOOP?

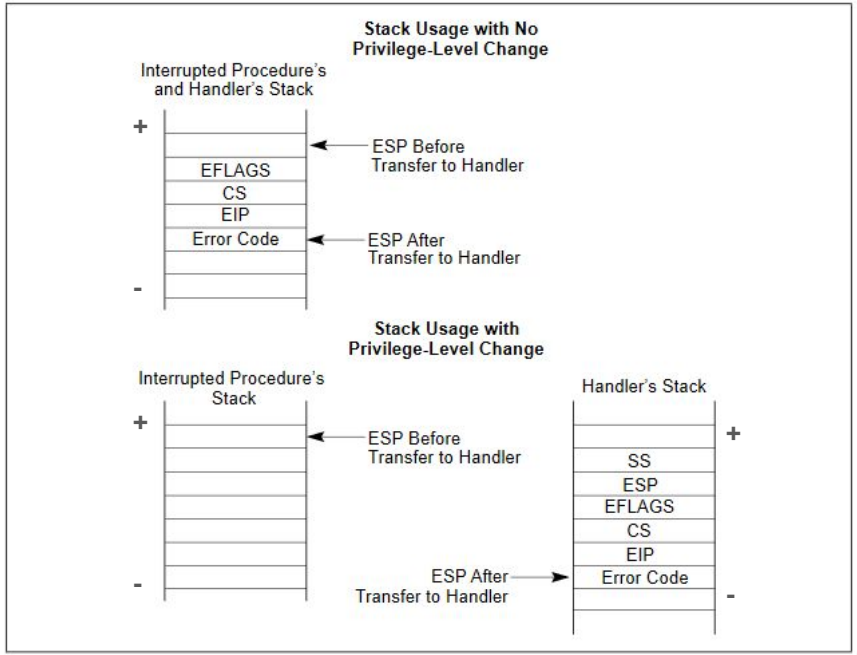


Figure 6-4. Stack Usage on Transfers to Interrupt and Exception-Handling Routines

Ejercicio 2

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes *0x0F 0x0B*.

B. Realice un **diagrama de pila** que muestre el estado de la pila del kernel luego de que una aplicación de usuario intentó ejecutar RSTL00P.

¿Qué ocurre cuando una tarea intenta ejecutar RSTLOOP?

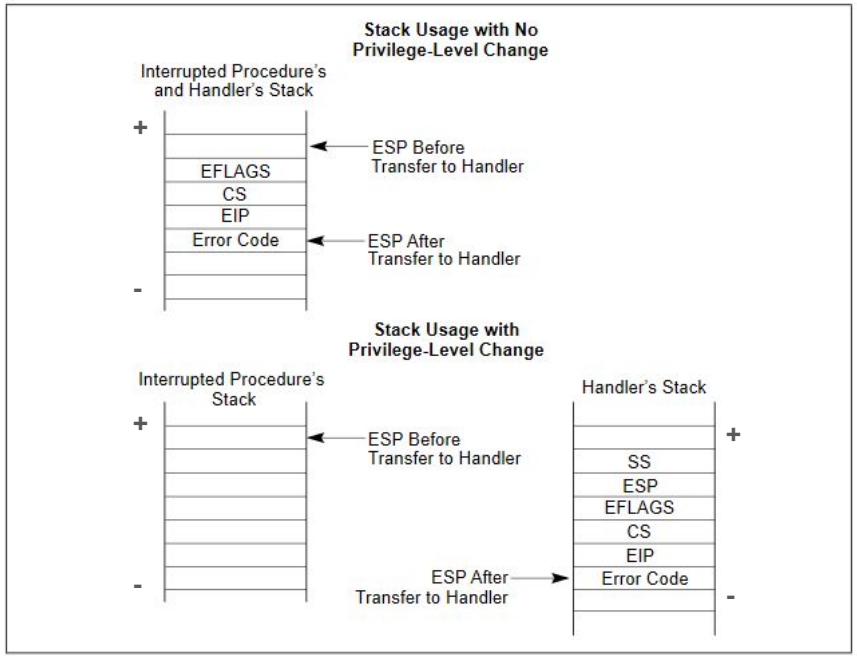


Figure 6-4. Stack Usage on Transfers to Interrupt and Exception-Handling Routines

Ejercicio 2

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes `0x0F 0x0B`**.**

B. Realice un **diagrama de pila** que muestre el estado de la pila del kernel luego de que una aplicación de usuario intentó ejecutar RSTL00P**.**

C. ¿Qué **dirección de retorno** se encuentra en la pila al atender la excepción?

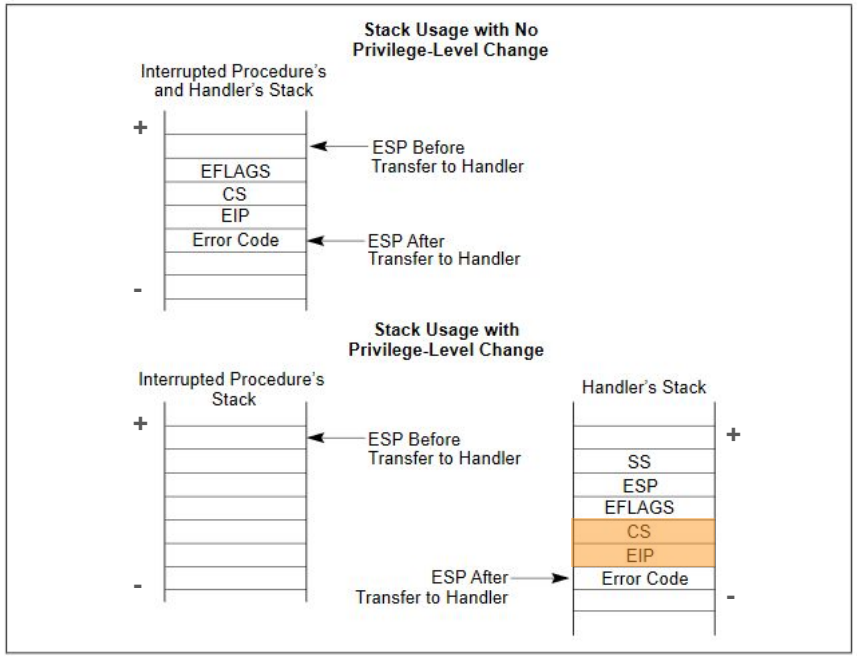


Figure 6-4. Stack Usage on Transfers to Interrupt and Exception-Handling Routines

Ejercicio 2

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes `0x0F 0x0B`.

B. Realice un **diagrama de pila** que muestre el estado de la pila del kernel luego de que una aplicación de usuario intentó ejecutar RSTL00P.

C. ¿Qué **dirección de retorno** se encuentra en la pila al atender la excepción?

Exception Error Code

None.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the instruction that generated the exception.

Program State Change

A program-state change does not accompany an invalid-opcode fault, because the invalid instructi executed.

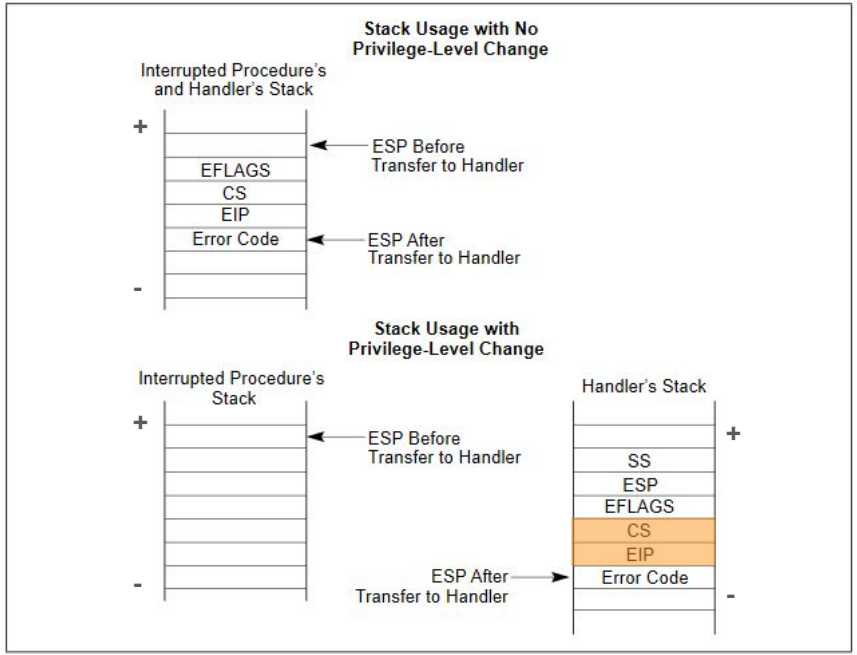


Figure 6-4. Stack Usage on Transfers to Interrupt and Exception-Handling Routines

Ejercicio 2

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes `0x0F 0x0B`**.**

D. Describa una posible implementación de RSTL00P utilizando el mecanismo descrito en (a) y (b).

- El mecanismo propuesto **sólo debe actuar** cuándo la instrucción no soportada es RSTL00P.
- Si la instrucción que generó la excepción **no es RSTL00P la tarea debe ser deshabilitada** y la ejecución debe saltar a la tarea idle.
- Si la instrucción que generó la excepción es RSTL00P adecúe la dirección de retorno de manera que permita a la tarea **continuar la ejecución** sin problemas.

Ejercicio 2

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes `0x0F 0x0B`**.**

D. Describa una posible implementación de RSTL00P utilizando el mecanismo descrito en (a) y (b).

- El mecanismo propuesto **sólo debe actuar** cuándo la instrucción no soportada es RSTL00P.
- Si la instrucción que generó la excepción **no es RSTL00P la tarea debe ser deshabilitada** y la ejecución debe saltar a la tarea idle.
- Si la instrucción que generó la excepción es RSTL00P adecúe la dirección de retorno de manera que permita a la tarea **continuar la ejecución** sin problemas.

Ejercicio 2

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes `0x0F 0x0B`**.**

D. Describa una posible implementación de RSTL00P utilizando el mecanismo descrito en (a) y (b).

- El mecanismo propuesto **sólo debe actuar** cuándo la instrucción no soportada es RSTL00P.
- Si la instrucción que generó la excepción **no es RSTL00P la tarea debe ser deshabilitada** y la ejecución debe saltar a la tarea idle.
- Si la instrucción que generó la excepción es RSTL00P adecúe la dirección de retorno de manera que permita a la tarea **continuar la ejecución** sin problemas.

Modificamos la rutina de atención de la interrupción 6.
Dados:

- EIP: Puntero a la instrucción no reconocida
- CS, EFLAGS, ESP, SS: Estado de la tarea

Hacer:

- Si en EIP se encuentra la secuencia de bytes 0x0F, 0x0B: (Leemos los bytes en [EIP], [EIP+1])
  - Escribir 0 en el ECX de la tarea actual
  - Saltar a la siguiente instrucción de la tarea actual (EIP+2)
- Sino:
  - Deshabilitamos la tarea actual en el scheduler
  - Saltamos a IDLE

Ejercicio 2

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes `0x0F 0x0B`**.**

D. Describa una posible implementación de RSTL00P utilizando el mecanismo descrito en (a) y (b).

- El mecanismo propuesto **sólo debe actuar** cuándo la instrucción no soportada es RSTL00P.
- Si la instrucción que generó la excepción **no es RSTL00P la tarea debe ser deshabilitada** y la ejecución debe saltar a la tarea idle.
- Si la instrucción que generó la excepción es RSTL00P adecúe la dirección de retorno de manera que permita a la tarea **continuar la ejecución** sin problemas.

Modificamos la rutina de atención de la interrupción 6.
Dados:

- EIP: Puntero a la instrucción no reconocida
- CS, EFLAGS, ESP, SS: Estado de la tarea

Hacer:

- Si en EIP se encuentra la secuencia de bytes 0x0F, 0x0B: (Leemos los bytes en [EIP], [EIP+1])
  - Escribir 0 en el ECX de la tarea actual
  - Saltar a la siguiente instrucción de la tarea actual (EIP+2)
- Sino:
  - Deshabilitamos la tarea actual en el scheduler
  - Saltamos a IDLE

E. ¿Qué pasa si no hacemos este paso?

Ejercicio 2

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes `0x0F 0x0B`**.**

F. Detalle los cambios a las estructuras del sistema visto en el taller que haría para realizar la implementación descrita en (d).

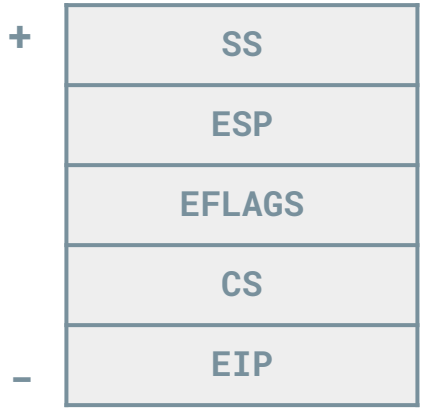
G. Muestre código para la rutina de atención de interrupciones descrita en (d) y todo otro cambio de comportamiento que haya visto necesario.

Ejercicio 2

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes *0x0F 0x0B*.

F. Detalle los cambios a las estructuras del sistema visto en el taller que haría para realizar la implementación descrita en (d). **-> modificamos sólo la RAI de opcode invalido (isr6)**

G. Muestre código para la rutina de atención de interrupciones descrita en (d) y todo otro cambio de comportamiento que haya visto necesario.



<- esp al comienzo de RAI opcode inválido

=> [ESP] = EIP

Si en EIP se encuentra la secuencia de bytes 0x0F, 0x0B: (Leemos los bytes en [EIP], [EIP+1])

- Escribir 0 en el ECX de la tarea actual
- Saltar a la siguiente instrucción de la tarea actual (EIP+2)

Sino:

- Deshabilitamos la tarea actual en el scheduler
- Saltamos a IDLE

Ejercicio 2

El **ENTEL575** fue un microprocesador compatible con los Intel i686. Este procesador incluía varias funcionalidades extra que nunca fueron replicadas por Intel. Una de ellas es la operación **RSTL00P**, la cual **escribe un cero en ECX *'reiniciando'* el contador de vueltas**. RSTL00P se codifica con la secuencia de bytes *0x0F 0x0B*.

G. Muestre código para la rutina de atención de interrupciones descrita en (d) y todo otro cambio de comportamiento que haya visto necesario.

isr.asm

```
extern current_task
extern sched_disable_task

_isr6:
  ; cargamos el EIP de la tarea
  mov ecx, [esp]
  ; cargamos la instrucción
  mov cx, [ecx]
  ; es rstloop?
  cmp cx, 0x0B0F
  je .emulate_rstloop

  ; no es, deshabilitamos
  push DWORD [current_task]
  call sched_disable_task
...
...
; salto a IDLE
  add esp, 4
  jmp (12 << 3):0

.emulate_rstloop:
      mov ecx, 0
      add DWORD [esp], 2
      iret
```

Si en EIP se encuentra la secuencia de bytes 0x0F, 0x0B: (Leemos los bytes en [EIP], [EIP+1])

- Escribir 0 en el ECX de la tarea actual
- Saltar a la siguiente instrucción de la tarea actual (EIP+2)

Sino:

- Deshabilitamos la tarea actual en el scheduler
- Saltamos a IDLE

Nota: [ESP] = EIP

sched.c

```c
typedef enum {
  TASK_SLOT_FREE,
  TASK_RUNNABLE,
  TASK_PAUSED
} task_state_t;

typedef struct {
  int16_t selector;
  task_state_t state;
} sched_entry_t;

static sched_entry_t sched_tasks[MAX_TASKS] = {0};
int8_t current_task = 0;
// ...

void sched_disable_task(int8_t task_id) {
  kassert(task_id >= 0 && task_id < MAX_TASKS, "Invalid task_id");
  sched_tasks[task_id].state = TASK_PAUSED;
}
// ...
```

Preguntas?