Clase pre-parcial

Organización del Computador 2
Segundo cuatrimestre 2023

Ejercicio 1 (70 puntos)

En un sistema similar al que implementamos en los talleres del curso (modo protegido con paginación activada) se pide:

A. Implementar la ***Syscall exit*** que al ser llamada por una tarea, **inactiva dicha tarea y pone a correr la siguiente** (segun indique el sistema de prioridad utilizado). Mostrar el código.

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

C. ¿Y si ahora no es la *Syscall exit* la que modifica el EAX de nivel 3 de la tarea que va a ser ejecutada luego de la llamada a la *Syscall* sino la **interrupción de reloj**? Como deberías modificar el código de la interrupción de reloj?. Mostrar el código y explicar todo lo que agregues al sistema.

D. ¿Considerás que es una buena práctica que las tareas se comuniquen sobre-escribiendo los registros de propósito general de otra tarea? ¿Qué problemas pueden surgir? Teniendo en cuenta lo visto en la materia, ¿de que **otra forma se podrían pasar mensajes las tareas** entre sí? Hacer un esquema que muestre los mapeos de memoria virtual a física de dos tareas que usen la forma de comunicarse propuesta por ustedes.
Realizá un diagrama que muestre la interacción del mecanismo con el scheduler y/o los mapeos de memoria.

Se recomienda organizar la resolución del ejercicio realizando paso a paso los items mencionados anteriormente y explicar las decisiones que toman.

Ejercicio 1 (70 puntos)

En un sistema similar al que implementamos en los talleres del curso (modo protegido con paginación activada) se pide:

A. Implementar la ***Syscall exit*** que al ser llamada por una tarea, **inactiva dicha tarea y pone a correr la siguiente** (segun indique el sistema de prioridad utilizado). Mostrar el código.

Ejercicio 1 (70 puntos)

A. Implementar la ***Syscall exit*** que al ser llamada por una tarea, **inactiva dicha tarea y pone a correr la siguiente** (segun indique el sistema de prioridad utilizado). Mostrar el código.

isr.asm

```
global _isr99

; Syscall exit
_isr99:
```

Ejercicio 1 (70 puntos)

A. Implementar la ***Syscall exit*** que al ser llamada por una tarea, **inactiva dicha tarea y pone a correr la siguiente** (segun indique el sistema de prioridad utilizado). Mostrar el código.

isr.asm

```
global _isr99

; Syscall exit
_isr99:
  pushad
  call sched_exit_task
```

Ejercicio 1 (70 puntos)

A. Implementar la ***Syscall exit*** que al ser llamada por una tarea, **inactiva dicha tarea y pone a correr la siguiente** (segun indique el sistema de prioridad utilizado). Mostrar el código.

isr.asm

```
global _isr99

; Syscall exit
_isr99:
  pushad
  call sched_exit_task
  ; Igual que en RAI de reloj
```

Ejercicio 1 (70 puntos)

A. Implementar la ***Syscall exit*** que al ser llamada por una tarea, **inactiva dicha tarea y pone a correr la siguiente** (segun indique el sistema de prioridad utilizado). Mostrar el código.

isr.asm

```
global _isr99

; Syscall exit
_isr99:
  pushad
  call sched_exit_task
  ; Igual que en RAI de reloj
  mov word [sched_task_selector], ax
  jmp far [sched_task_offset]
  ...
  ...
.fin:
  ; Actualizamos las estructuras
compartidas ante el tick del reloj
  call tasks_tick
  ; Actualizamos la "interfaz" del
sistema en pantalla
  call tasks_screen_update
  popad
  iret
```

Ejercicio 1 (70 puntos)

A. Implementar la ***Syscall exit*** que al ser llamada por una tarea, **inactiva dicha tarea y pone a correr la siguiente** (segun indique el sistema de prioridad utilizado). Mostrar el código.

isr.asm

```
global _isr99
_isr99:
  pushad
  call sched_exit_task
  mov word [sched_task_selector], ax
  jmp far [sched_task_offset]
  .fin:
  call tasks_tick
  call tasks_screen_update
  popad
  iret
```

sched.c

```
uint16_t sched_exit_task(void) {}
```

Ejercicio 1 (70 puntos)

A. Implementar la ***Syscall exit*** que al ser llamada por una tarea, **inactiva dicha tarea y pone a correr la siguiente** (segun indique el sistema de prioridad utilizado). Mostrar el código.

isr.asm

```
global _isr99
_isr99:
  pushad
  call sched_exit_task
  mov word [sched_task_selector], ax
  jmp far [sched_task_offset]
  .fin:
  call tasks_tick
  call tasks_screen_update
  popad
  iret
```

sched.c

```
uint16_t sched_exit_task(void) {
  // apagamos a la tarea
  sched_disable_task(current_task);
  // buscamos la siguiente tarea activa
  // de acuerdo a la politica de
scheduling
  // y devolvemos la nueva tarea
  return sched_next_task();
}
```

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

Pasos de la resolución:

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

Pasos de la resolución:

1. Obtener el id de la tarea que llama a la syscall

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

Pasos de la resolución:

1. Obtener el id de la tarea que llama a la syscall
2. Desactivar la tarea actual

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

Pasos de la resolución:

1. Obtener el id de la tarea que llama a la syscall
2. Desactivar la tarea actual
3. Buscar la siguiente tarea a ejecutar

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

Pasos de la resolución:

1. Obtener el id de la tarea que llama a la syscall
2. Desactivar la tarea actual
3. Buscar la siguiente tarea a ejecutar
    a. Obtener el id

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

Pasos de la resolución:

1. Obtener el id de la tarea que llama a la syscall
2. Desactivar la tarea actual
3. Buscar la siguiente tarea a ejecutar
    a. Obtener el id
    b. Obtener el selector

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

Pasos de la resolución:

1. Obtener el id de la tarea que llama a la syscall
2. Desactivar la tarea actual
3. Buscar la siguiente tarea a ejecutar
    a. Obtener el id
    b. Obtener el selector
4. Antes de realizar el cambio de contexto, escribir el id obtenido en el paso 1 en el EAX destino. Para eso:

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

Pasos de la resolución:

1. Obtener el id de la tarea que llama a la syscall
2. Desactivar la tarea actual
3. Buscar la siguiente tarea a ejecutar
    a. Obtener el id
    b. Obtener el selector
4. Antes de realizar el cambio de contexto, escribir el id obtenido en el paso 1 en el EAX destino. Para eso:
    a. Obtenemos la TSS de la nueva tarea

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

Pasos de la resolución:

1. Obtener el id de la tarea que llama a la syscall
2. Desactivar la tarea actual
3. Buscar la siguiente tarea a ejecutar
    a. Obtener el id
    b. Obtener el selector
4. Antes de realizar el cambio de contexto, escribir el id obtenido en el paso 1 en el EAX destino. Para eso:
    a. Obtenemos la TSS de la nueva tarea
    b. Buscamos el ESP de la nueva tarea

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

Pasos de la resolución:

1. Obtener el id de la tarea que llama a la syscall
2. Desactivar la tarea actual
3. Buscar la siguiente tarea a ejecutar
    a. Obtener el id
    b. Obtener el selector
4. Antes de realizar el cambio de contexto, escribir el id obtenido en el paso 1 en el EAX destino. Para eso:
    a. Obtenemos la TSS de la nueva tarea
    b. Buscamos el ESP de la nueva tarea
    c. Nos movemos en la pila hasta donde se ubica el EAX (pusheado por `pushad` antes de cambiar de contexto)

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

Pasos de la resolución:

1. Obtener el id de la tarea que llama a la syscall
2. Desactivar la tarea actual
3. Buscar la siguiente tarea a ejecutar
    a. Obtener el id
    b. Obtener el selector
4. Antes de realizar el cambio de contexto, escribir el id obtenido en el paso 1 en el EAX destino. Para eso:
    a. Obtenemos la TSS de la nueva tarea
    b. Buscamos el ESP de la nueva tarea
    c. Nos movemos en la pila hasta donde se ubica el EAX (pusheado por `pushad` antes de cambiar de contexto)
    d. Lo pisamos con el id

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

Pasos de la resolución:

1. Obtener el id de la tarea que llama a la syscall
2. Desactivar la tarea actual
3. Buscar la siguiente tarea a ejecutar
    a. Obtener el id
    b. Obtener el selector
4. Antes de realizar el cambio de contexto, escribir el id obtenido en el paso 1 en el EAX destino. Para eso:
    a. Obtenemos la TSS de la nueva tarea
    b. Buscamos el ESP de la nueva tarea
    c. Nos movemos en la pila hasta donde se ubica el EAX (pusheado por `pushad` antes de cambiar de contexto)
    d. Lo pisamos con el id
5. Seguimos con el cambio de contexto normal

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

Consideraciones?

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

Consideraciones:

- Se asume que la nueva tarea fue desalojada por el reloj al menos una vez.
  - Si no fuera así, el eax de la tss va a ser el de nivel de usuario y deberíamos pisar ese directamente
- Va a ser necesario modificar la RAI de reloj para que reciba por parámetro el id de la nueva tarea. queda para el punto C.

Ejercicio 1 (70 puntos)

Lo que teníamos en A

isr.asm

```
global _isr99
_isr99:
  pushad
  call sched_exit_task
  mov word [sched_task_selector], ax
  jmp far [sched_task_offset]
  .fin:
  call tasks_tick
  call tasks_screen_update
  popad
  iret
```

sched.c

```
uint16_t sched_exit_task(void) {
  // apagamos a la tarea
  sched_disable_task(current_task);
  // buscamos la siguiente tarea activa
  // de acuerdo a la politica de
scheduling
  // y devolvemos la nueva tarea
  return sched_next_task();
}
```

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

Pasos de la resolución:

1. Obtener el id de la tarea que llama a la syscall
2. Desactivar la tarea actual
3. Buscar la siguiente tarea a ejecutar
    a. Obtener el id
    b. Obtener el selector
4. Antes de realizar el cambio de contexto, escribir el id obtenido en el paso 1 en el EAX destino. Para eso:
    a. Obtenemos la TSS de la nueva tarea
    b. Buscamos el ESP de la nueva tarea
    c. Nos movemos en la pila hasta donde se ubica el EAX (pusheado por `pushad` antes de cambiar de contexto)
    d. Lo pisamos con el id
5. Seguimos con el cambio de contexto normal

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

isr.asm

```
extern current_task
…
_isr99:
  pushad
  ; paso 1
  push DWORD [current_task]
  ; paso 2
  call sched_disable_task
  ; paso 3.a
  call sched_next_task_id
; ahora tenemos en eax el id de la nueva tarea en eax
  push eax
  ...
```

1. Obtener el id de la tarea que llama a la syscall
2. Desactivar la tarea actual
3. Buscar la siguiente tarea a ejecutar
    a. Obtener el id
    b. Obtener el selector

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

isr.asm

```
  ; tenemos en eax el id de la nueva tarea
en eax
  ; paso 4
  call pass_exit_id_to_next_task
  ; paso 3.b
  call sched_next_task
  ; paso 5, sigue igual que antes
  ; Igual que en RAI de reloj
  mov word [sched_task_selector], ax
  jmp far [sched_task_offset]

  ;terminamos la interrupción
  ...
```

3. Buscar la siguiente tarea a ejecutar
   a. Obtener el id
   b. Obtener el selector
4. Escribir el id obtenido en el paso 1 en el EAX destino
   a. Obtenemos TSS de la nueva tarea
   b. Buscamos el ESP de la nueva tarea
   c. Buscamos EAX en pila (por pushad)
   d. Lo pisamos con el id
5. Seguimos con el cambio de contexto normal

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

isr.asm

```
  .fin:
  mov esp, 8
  ; Actualizamos las estructuras compartidas ante
el tick del reloj
  call tasks_tick
  ; Actualizamos la "interfaz" del sistema en
pantalla
  call tasks_screen_update
  popad
  iret
```

5. Seguimos con el cambio de contexto normal

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

sched.c

```
// es la primera parte de sched_next_task como funcion aux
uint8_t sched_next_task_id(void) {
  // Buscamos la próxima tarea viva (comenzando en la
actual)
  int8_t i;
  for (i = (current_task + 1); (i % MAX_TASKS) !=
current_task; i++) {
    // Si esta tarea está disponible la ejecutamos
    if (sched_tasks[i % MAX_TASKS].state == TASK_RUNNABLE)
      break;
  }
  // Ajustamos i para que esté entre 0 y MAX_TASKS-1
  i = i % MAX_TASKS;
  return i
}
```

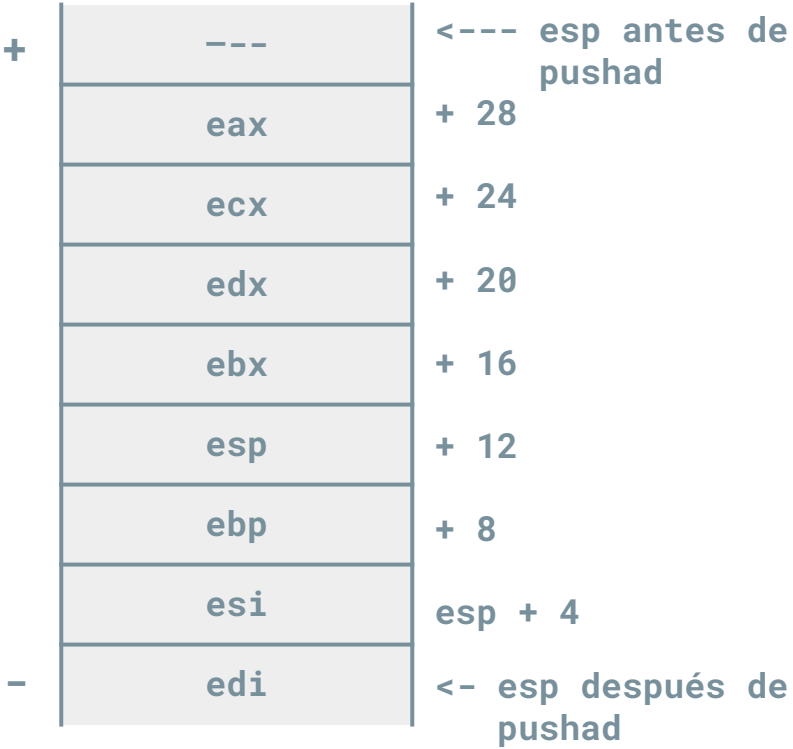
```
uint16_t sched_next_task(uint8_t task_id){
  // Si la tarea que encontramos es
ejecutable entonces vamos a correrla.
  if (sched_tasks[i].state == TASK_RUNNABLE){
    current_task = i;
    return sched_tasks[i].selector;
  }
  // En el peor de los casos no hay ninguna
tarea viva. Usemos la idle como selector.
  return GDT_IDX_TASK_IDLE << 3;
}
```

Ejercicio 1 (70 puntos)

B. ¿Cómo modificarías el punto anterior para que exit (además de lo que hace normalmente) guarde el ID de quién la llamó en el **EAX de próxima tarea a ejecutar**? Mostrar código.

tasks.c

```
void pass_exit_id_to_next_task(uint8_t exit_task_id,
uint8_t new_task_id) {
  tss_t new_task_tss = tss_tasks[new_task_id];
  uint32_t* new_task_esp = (uint32_t*) new_task_tss.esp;
  // como es nivel 0, esta mapeado con identity mapping
  // EAX es el primer registro en pushearse con pushad,
  // por lo tanto le sumo 28 al esp
  *(new_task_esp + 28) = exit_task_id;
  return;
}
```



Ejercicio 1 (70 puntos)

C. ¿Y si ahora no es la *Syscall exit* la que modifica el EAX de nivel 3 de la tarea que va a ser ejecutada luego de la llamada a la *Syscall* sino la **interrupción de reloj**? Cómo deberías modificar el código de la interrupción de reloj? Mostrar el código y explicar todo lo que agregues al sistema.

isr.asm

```
_isr32: ;rutina de atención del reloj
  pushad
  call pic_finish1
  call next_clock

  push DWORD [current_task]
  call sched_next_task
  cmp ax, 0
  je .fin
  str bx
  cmp ax, bx
  je .fin

  mov word [sched_task_selector], ax
  jmp far [sched_task_offset]

  .fin:
  call tasks_tick
  call tasks_screen_update
  popad
  iret
```

isr.asm

```
_isr99: ;syscall exit
  pushad
  push DWORD [current_task]
  call sched_disable_task
  call sched_next_task_id
  push eax
  call pass_exit_id_to_next_task
  call sched_next_task

  mov word [sched_task_selector], ax
  jmp far [sched_task_offset]

  .fin:
  mov esp, 8
  call tasks_tick
  call tasks_screen_update
  popad
  iret
```

Ejercicio 1 (70 puntos)

C. ¿Y si ahora no es la *Syscall exit* la que modifica el EAX de nivel 3 de la tarea que va a ser ejecutada luego de la llamada a la *Syscall* sino la **interrupción de reloj**? Cómo deberías modificar el código de la interrupción de reloj? Mostrar el código y explicar todo lo que agregues al sistema.

isr.asm

```
_isr32: ;rutina de atención del reloj
  pushad
  call pic_finish1
  call next_clock

  push DWORD [current_task]
  call sched_next_task
  cmp ax, 0
  je .fin
  str bx
  cmp ax, bx
  je .fin

  mov word [sched_task_selector], ax
  jmp far [sched_task_offset]

  .fin:
  call tasks_tick
  call tasks_screen_update
  popad
  iret
```

isr.asm

```
_isr99: ;syscall exit
  pushad
  push DWORD [current_task]
  call sched_disable_task
  call sched_next_task_id
  push eax
  call pass_exit_id_to_next_task
  call sched_next_task

  mov word [sched_task_selector], ax
  jmp far [sched_task_offset]

  .fin:
  mov esp, 8
  call tasks_tick
  call tasks_screen_update
  popad
  iret
```

isr32 e isr99 son parecidas, pero isr99:

Desactiva tarea actual (queda en isr99)

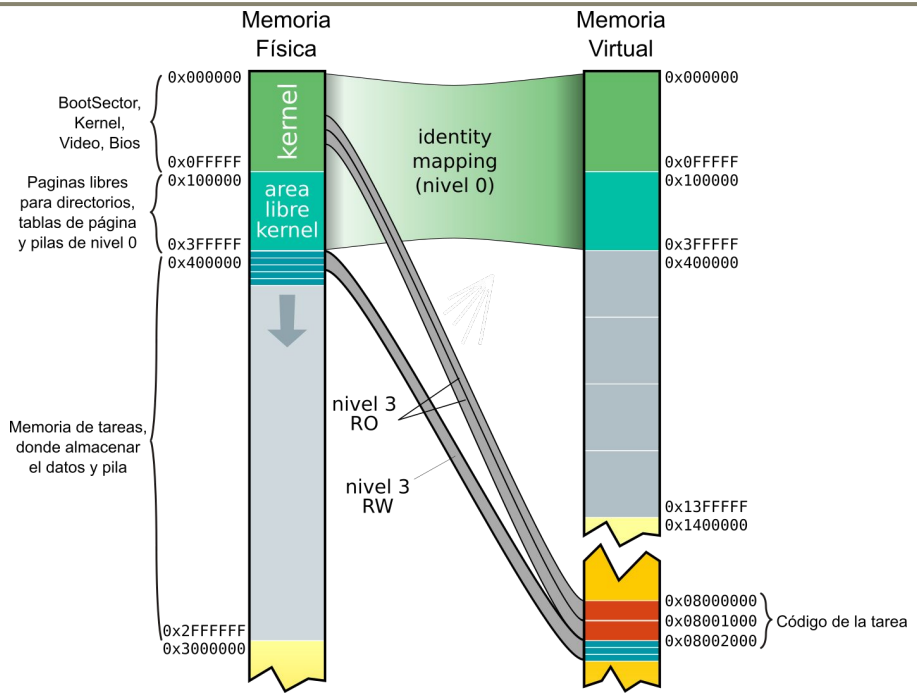
Escribe id en eax de la tarea siguiente (pasa a isr32)

Ejercicio 1 (70 puntos)

D. ¿Considerás que es una buena práctica que las tareas se comuniquen sobre-escribiendo los registros de propósito general de otra tarea? ¿Qué problemas pueden surgir? Teniendo en cuenta lo visto en la materia, ¿de que **otra forma se podrían pasar mensajes las tareas** entre sí? Hacer un esquema que muestre los mapeos de memoria virtual a física de dos tareas que usen la forma de comunicarse propuesta por ustedes.
Realizá un diagrama que muestre la interacción del mecanismo con el scheduler y/o los mapeos de memoria.

Ejercicio 1 (70 puntos)

D. ¿Considerás que es una buena práctica que las tareas se comuniquen sobre-escribiendo los registros de propósito general de otra tarea? ¿Qué problemas pueden surgir? Teniendo en cuenta lo visto en la materia, ¿de que **otra forma se podrían pasar mensajes las tareas** entre sí? Hacer un esquema que muestre los mapeos de memoria virtual a física de dos tareas que usen la forma de comunicarse propuesta por ustedes.
Realizá un diagrama que muestre la interacción del mecanismo con el scheduler y/o los mapeos de memoria.



Preguntas?

Ahora pasamos al Ej. 2

Ejercicio 2 (30 puntos)

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 (30 puntos)

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?

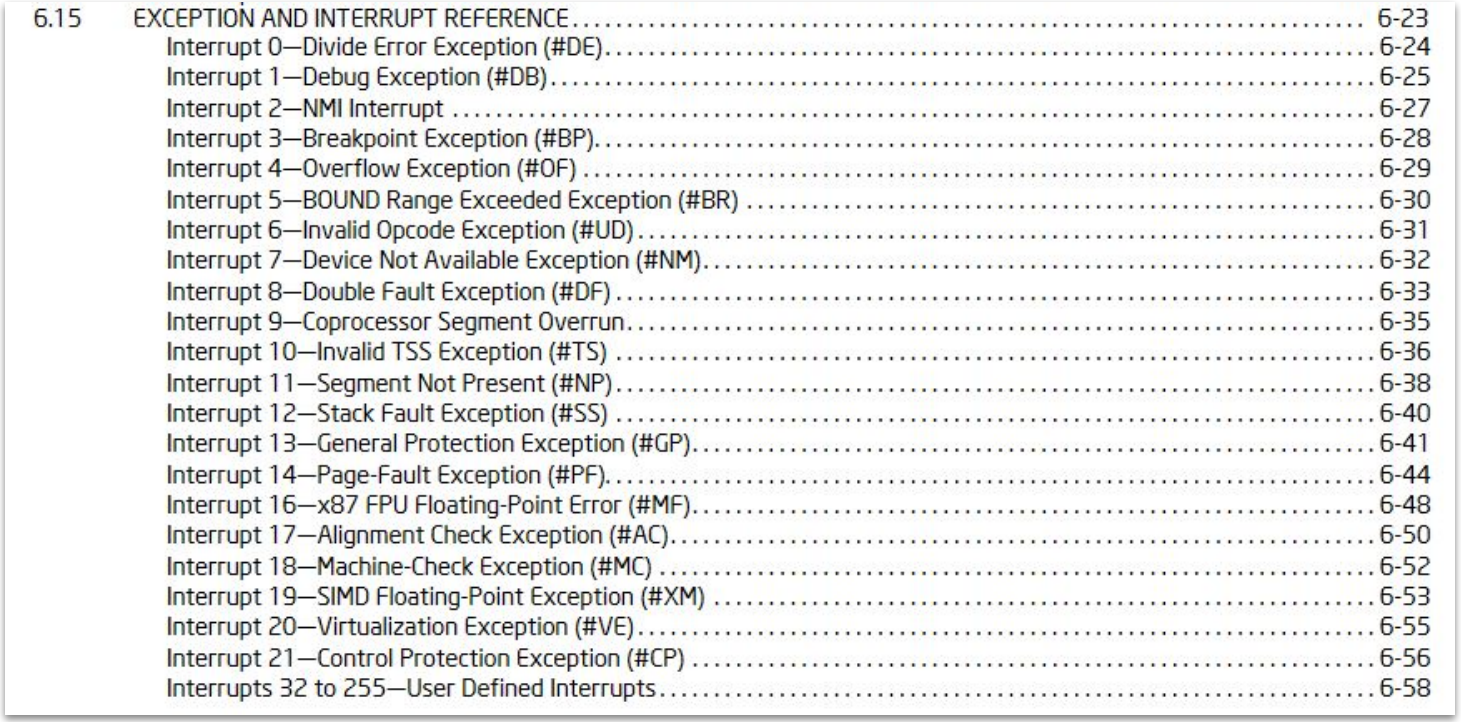
Recomendaciones:

- 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 (30 puntos)

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*.

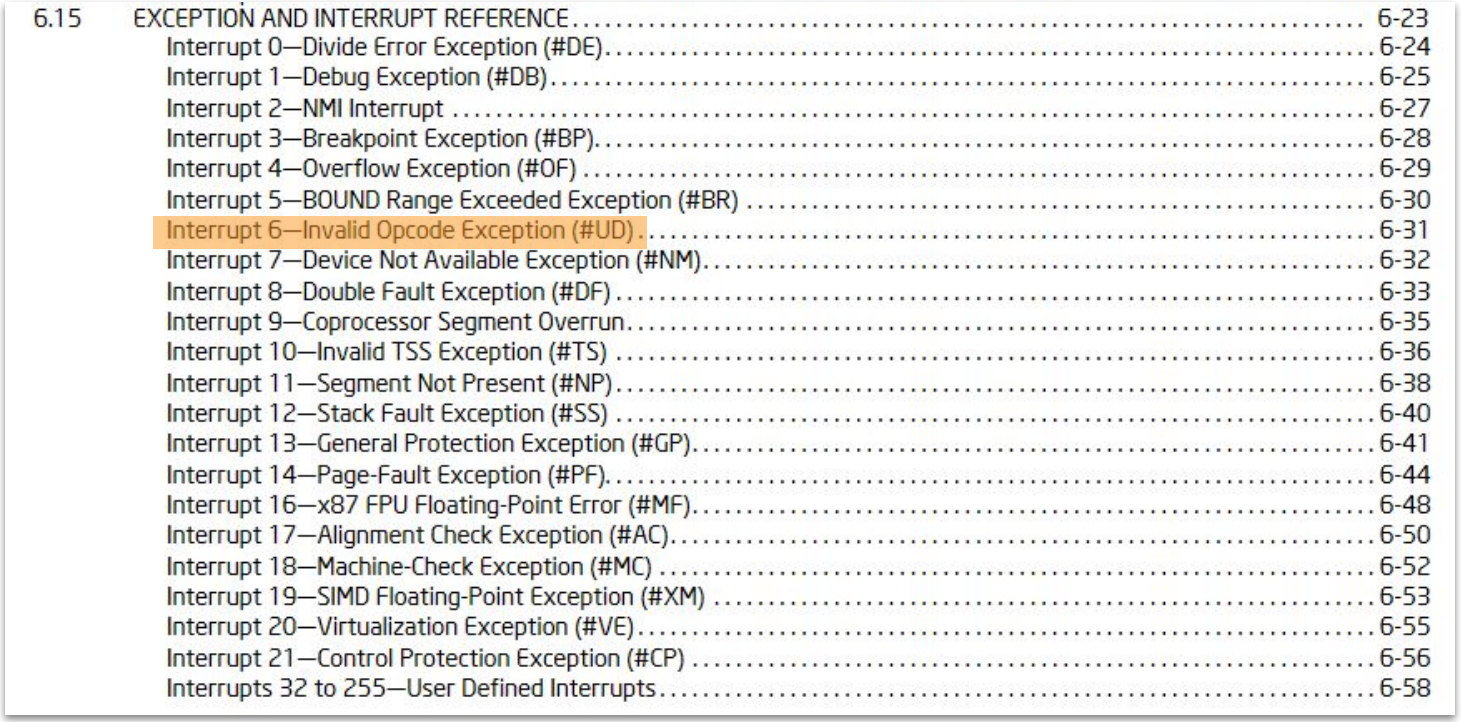
A. ¿**Qué excepción** ocurre cuándo un procesador x86 intenta ejecutar una instrucción no soportada?



Ejercicio 2 (30 puntos)

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 (30 puntos)

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 (30 puntos)

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 (30 puntos)

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?

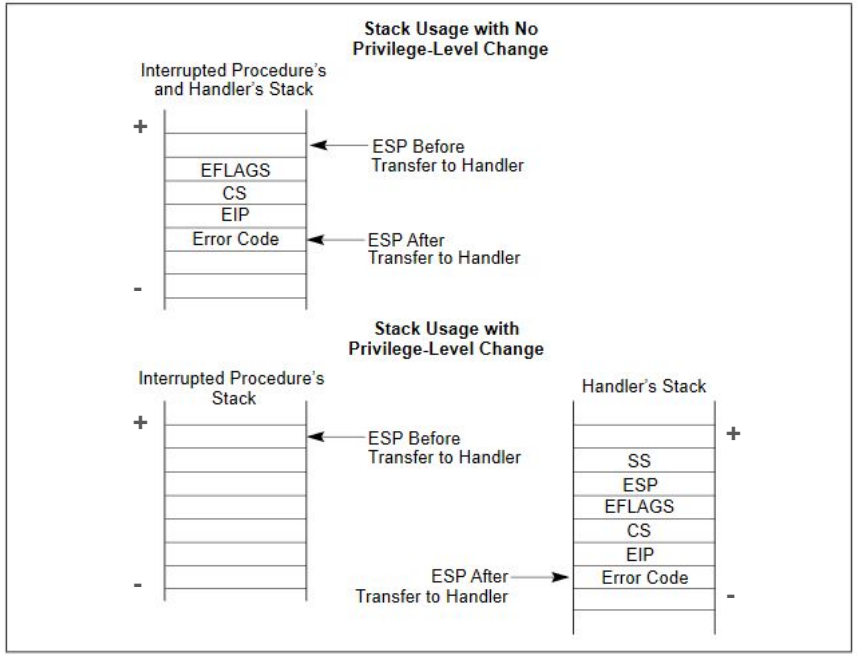


Figure 6-4. Stack Usage on Transfers to Interrupt and Exception-Handling Routines

Ejercicio 2 (30 puntos)

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?

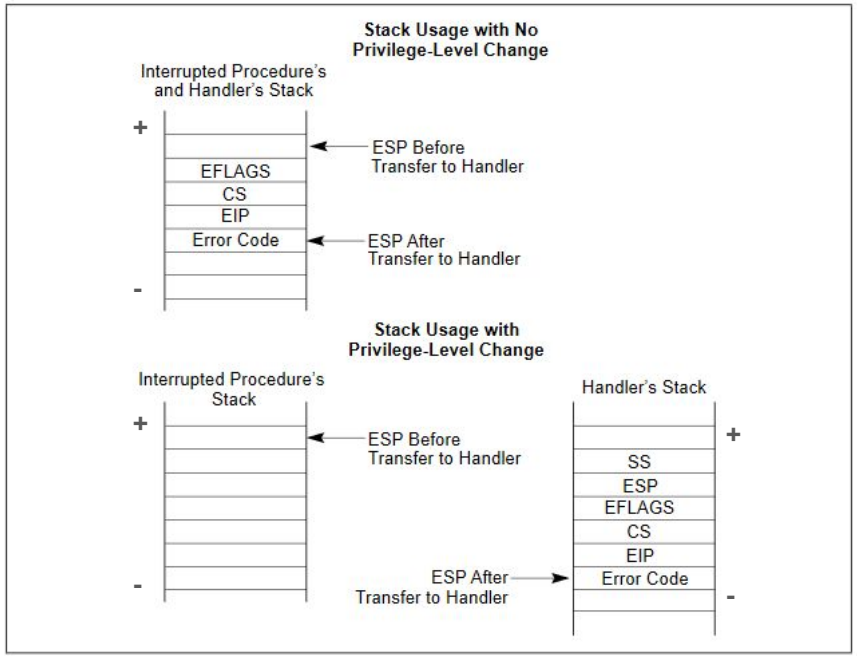


Figure 6-4. Stack Usage on Transfers to Interrupt and Exception-Handling Routines

Ejercicio 2 (30 puntos)

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?

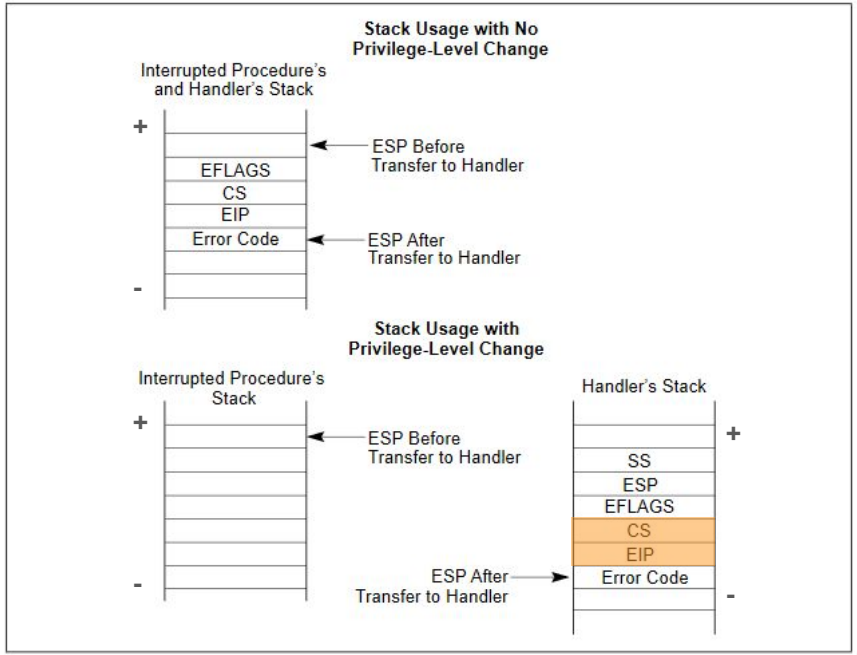


Figure 6-4. Stack Usage on Transfers to Interrupt and Exception-Handling Routines

Ejercicio 2 (30 puntos)

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.

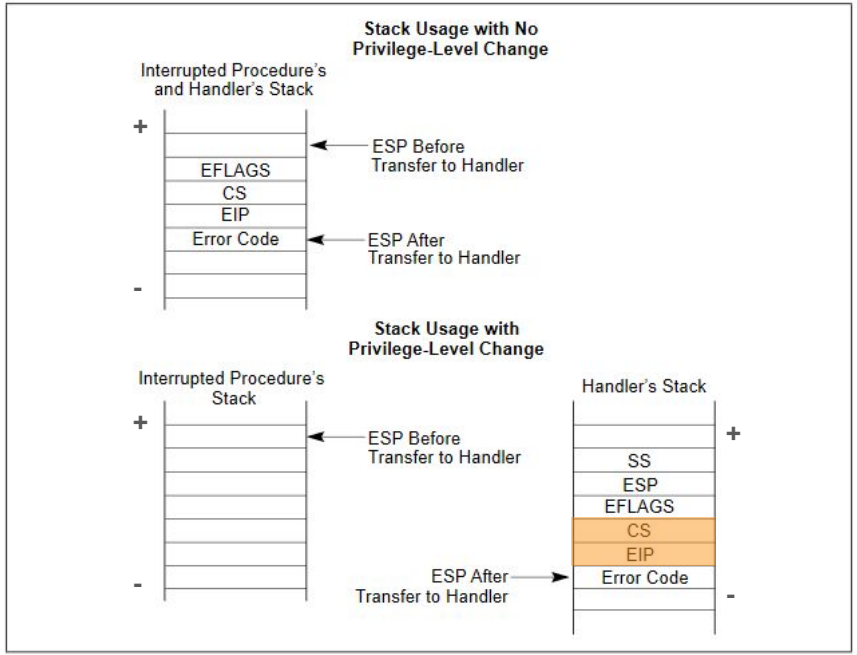


Figure 6-4. Stack Usage on Transfers to Interrupt and Exception-Handling Routines

Ejercicio 2 (30 puntos)

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 (30 puntos)

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 (30 puntos)

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 (30 puntos)

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 (30 puntos)

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`**.**

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

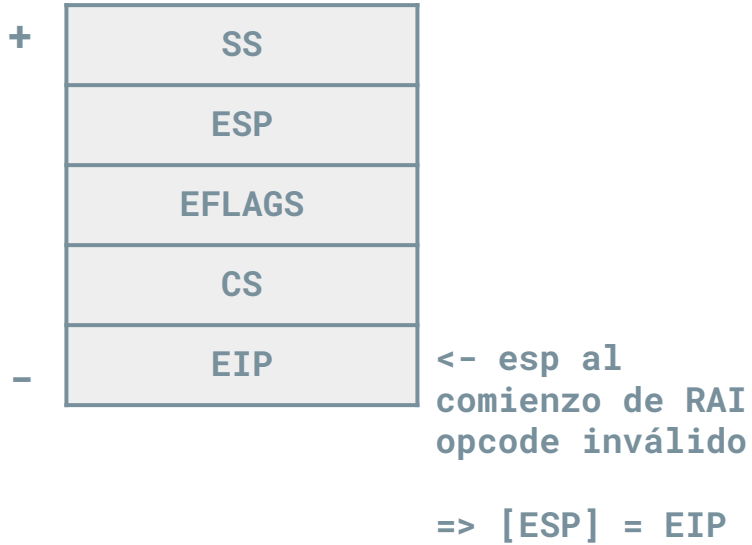
F. Muestre código para la rutina de atención de interrupciones descrita en (d) y todo otro cambio de comportamiento que haya visto necesario.

Ejercicio 2 (30 puntos)

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`.

E. 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)**

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



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 (30 puntos)

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. 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

Preguntas?