Práctica 11
Entrada/salida 2

Objetivos

- Profundizar en el sistema de Entada/Salida.
- Conocer como trabaja el mecanismo de excepciones en MIPS.
- Entender el mecanismo de sincronización por interrupciones.
- Ser capaz de escribir una rutina simple de tratamiento de excepciones en MIPS.

Material

Simulador MARS y un código fuente de partida.

Teoría:

1. Introducción

El procesador tiene la capacidad de responder a cualquier acontecimiento imprevisto que tenga lugar externamente a él. Podemos observar que, en situaciones normales, por ejemplo, siempre que se pulse una tecla del teclado o se mueva el ratón, el procesador responderá adecuadamente.
Como podéis suponer, el hecho de que el programa esté constantemente indagando en los dispositivos de entrada y salida para ver si está preparado para hacer una transferencia resulta en una pérdida de los recursos del procesador. Existe un mecanismo alternativo, las interrupciones, para realizar las operaciones de entrada y salida. Una interrupción es como una llamada imprevista a un procedimiento que será invocado exclusivamente cuando el dispositivo de E/S se encuentra preparado para hacer la operación. Las interrupciones eliminan la necesidad de estar consultando continuamente al teclado o al ratón. Para poder utilizar esta opción, los dispositivos han de estar habilitados para aceptar interrupciones.
Cuando un dispositivo activa una interrupción, el hardware del procesador interrumpe la ejecución del programa, se entra en modo *kernel* y el control se transfiere a un espacio especial de direcciones dentro del kernel. El código allí localizado realiza todas las acciones necesarias para dar servicio a la interrupción. Una vez hecho se devuelve el control en el programa interrumpido y la ejecución continúa.
Estudiamos a continuación como el MIPS trata este mecanismo.

2. Excepciones e interrupciones en el MIPS

En prácticas anteriores habéis estudiado como las instrucciones de salto y llamada a una función provocan un cambio en la ejecución secuencial de los programas. En esta

práctica estudiaremos un nuevo mecanismo que produce una alteración en el flujo normal de instrucciones, como son las excepciones.
Una excepción es como una llamada imprevista a un procedimiento, provocando pues, un salto a una nueva dirección. Las excepciones pueden ser causadas tanto por software como por hardware. Las excepciones causadas por software son debidas a errores en la ejecución de instrucciones y por hardware son debidas a interrupciones externas producidas por dispositivos de E/S.
Se dice que una excepción es **síncrona** si está provocada por una instrucción durante la ejecución del programa. Una excepción síncrona la puede causar una excepción aritmética (debido a un desbordamiento o una división por cero), una instrucción de lectura o escritura en memoria al generar una dirección invalida (por ejemplo, un acceso a una palabra no alineada), el intento de ejecución de una instrucción ilegal (un código máquina que no tiene sentido), o por una instrucción syscall (cuando se necesita un servicio del sistema operativo). Por ejemplo, si ocurre un error como una división por cero, el programa salta a una sección de código dentro del sistema operativo que decidirá las acciones a tomar. A las excepciones de tipos software a veces se las conoce como ***traps***.

Se dice que una excepción es **asíncrona** si un dispositivo de E/S solicita la atención del procesador. Por ejemplo, el procesador puede recibir una señal que indicó que el usuario ha presionado una tecla, entonces el procesador detiene lo que estaba haciendo para determinar cual es la tecla presionada y realizar las acciones oportunas. Una vez ha acabado, el procesador continúa el programa por donde se había quedado. Una excepción como esta se denomina ***interrupción*** puesto que no está relacionada con el programa que está ejecutándose. Las interrupciones las pueden provocar una gran variedad de dispositivos de E/S como por ejemplo el teclado, la consola o el controlador de disco.

Cuando ocurre la excepción se transfiere el control a la rutina de tratamiento de excepciones (***exception handler***), que es un programa diseñado exclusivamente con el propósito de tratar las excepciones. Una vez ejecutado se vuelve al programa original que continúa como si no hubiera pasado nada. Se puede ver como un procedimiento al que se llama sin argumentos y que no devuelve ningún valor.

3. Excepciones e interrupciones en el MARS

Los procesadores MIPS además de la CPU y del coprocesador 1, tiene el llamado coprocesador 0 que contiene la información que los programas necesitan para tratar con las excepciones y las interrupciones.
El simulador MARS no implementa todos los registros del coprocesador 0, sólo aquellos que le serán útiles. Los registros simulados por el MARS se muestran en la tabla 1 y serán los únicos que comentaremos en la práctica. En el simulador los podéis observar pulsando en la pestaña *Coproc 0* dentro de la ventana de registros.

Nombre del registro	Registro	Descripción
VAddr	$8	Referencia inválida a memoria
Status	$12	Controla qué interrupciones están habilitadas
Cause	$13	Interrupciones pendientes y tipos de excepción
EPC	$14	Dirección de la instrucción donde ha ocurrido la excepción

Tabla 1. Registros del coprocesador 0 del MIPS implementados por el MARS

A continuación, comentamos cada uno de estos registros:
Cuando se produce una excepción el registro ***EPC ($14)*** contiene la dirección de la instrucción que se encontraba ejecutándose en el momento de ocurrir la excepción. Si la excepción la causa una interrupción externa significa que la instrucción todavía no ha empezado a ejecutarse. Las interrupciones no pueden darse en mitad de una instrucción. Esto significa que, si la excepción no ha sido ocasionada por una interrupción, el tratamiento de excepciones tiene que encargarse de incrementar el registro EPC en 4 para que el programa retome su ejecución en la instrucción siguiente, puesto que de otra manera se repetiría la instrucción causante de la excepción y se entraría en un bucle infinito. En cualquier caso, al volver de la rutina de tratamiento de excepciones el programa se retomará en la instrucción apuntada por el registro ***EPC***.
Si la excepción lo ha causado una referencia a dirección de memoria inválida, el registro ***VAddr ($8)*** contenderá la dirección errónea.

El registro ***Status ($12)*** se muestra en la figura 1.

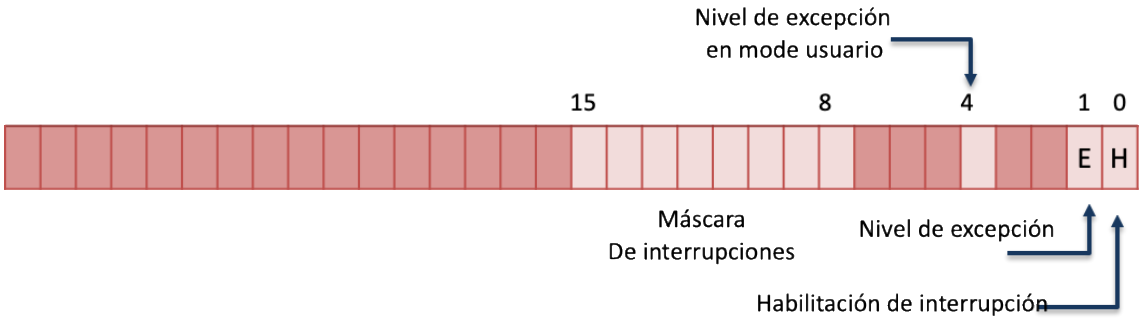

Figura 1. Registros Status ($12)

El registro *Status* consta de los siguientes campos: El bit 0 (H) es el bit de habilitación de interrupción, si está a 1 se permiten las interrupciones, si está a 0 las interrupciones están deshabilitadas. El bit de nivel de excepción (E) (bit 1) normalmente está a 0 y se pone a 1 cuando ocurre la excepción previniendo de este modo que una nueva excepción interrumpa a la rutina de tratamiento de excepciones. El bit de excepción en modo usuario (bit 4) se pone a 0 cuando el programa está ejecutándose en modo *kernel* y a 1 si está en modo usuario. En el MARS este bit está fijado a 1 puesto que no se implementa el modo kernel. La máscara de interrupción contiene 8 bits (bits 8 al 15), uno para cada una de las seis interrupciones de nivel hardware y dos de nivel software. Si el bit de la máscara está a 0 se deshabilita la interrupción correspondiente.

El registro ***Cause ($13)*** se muestra en la figura 2.

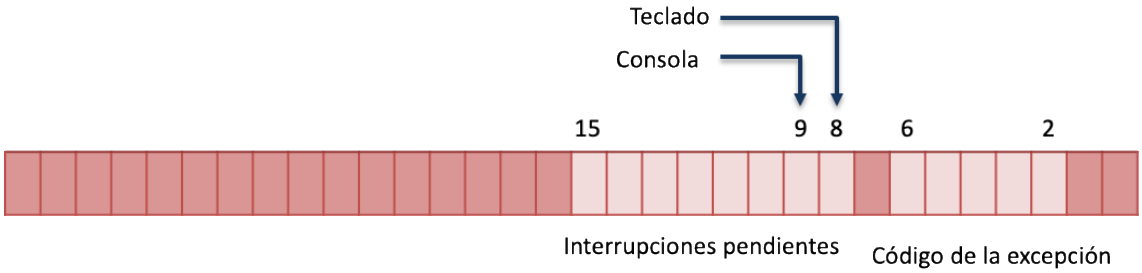

Figura 2. Registro Cause ($13)

El registro ***Cause*** proporciona información de la fuente que ha provocado la excepción en el campo Código de excepción (bits 2 al 6). Si se activa una interrupción se pone el bit correspondiente a 1 dentro del campo de las interrupciones pendientes (bit 8 al 15) aunque el correspondiente bit de la máscara (en el registro Satuts) esté deshabilitado. En el MARS sólo se puede utilizar el teclado y la consola. El bit 8 corresponde a la interrupción del teclado y el bit 9 a la interrupción de la consola.

Algunos de los códigos de excepción que pueden darse y que están implementados en MARS se encuentran en la tabla 2.

Número	Nombre	Causa de la excepción
0	Int	Interrupción (Hardware)
4	AdEL	Excepción por dirección errónea (*Load* o busca de instrucción)
5	AdES	Excepción por dirección errónea (*Store*)
8	SyS0	Excepción syscall
9	Bp	Excepción por punto de ruptura (*breakpoint*)
10	RI	Excepción por instrucción reservada
12	Ov	Excepción por desbordamiento aritmético
13	Tr	*Trap*
15	FPE	Excepción de coma flotante

Tabla 2: Códigos de excepción.

Observamos a continuación distintos ejemplos de excepciones:

1. Dado el siguiente código en MIPS:

```
# Excepción por desbordamiento aritmético

li $t0, 0x7FFFFFFF
addiu $t1, $t0,1 #Se ignora el desbordamiento
```

```
addi $t2, $t0, 1 #Detecta el desbordamiento
```

- Observa el código. ¿Qué instrucción causará la excepción?
- Ensambla y ejecuta el código a pasos. Observa el cambio de los registros del coprocesador 0.
- ¿Cuáles son los valores de los registros del coprocesador 0 antes y después de producirse la excepción?
- ¿Cuál es el significado de los distintos campos de los registros del coprocesador 0 después de producirse la excepción?

2. Dado el siguiente código en MIPS:

```
# Excepción por dirección errónea

li $a0, 5
sw $a0, 124($zero)
```

- Ensambla y ejecuta el código.
- ¿Cuáles son los valores de los registros del coprocesador 0 antes y después de producirse la excepción?
- ¿Cuál es el significado de los distintos campos de los registros del coprocesador 0 después de producirse la excepción?

3. Dado el siguiente codigo en MIPS:

```
# Excepción por dirección no alineada en Load
.data
vector: .word 1, 3, 5, 7, 11, 13

.text
#. . .
la $t0, vector
lw $t0, 3($t0)
```

- Ensambla y ejecuta el código.
- ¿Cuáles son los valores de los registros del coprocesador 0 antes y después de producirse la excepción?
- ¿Cuál es el significado de los distintos campos de los registros del coprocesador 0 después de producirse la excepción?

4. Supón que el contenido del registro **Cause ($13)** tiene los siguientes valores después de haberse producido una excepción. Rellena la tabla indicando cual ha sido la causa que ha provocado la excepción en cada caso:

Cause	Fuente de la excepción
0x00000000	
0x00000020	
0x00000024	

0x00000028	
0x00000030	

- Rellena la tabla indicando cuál ha sido la causa que ha provocado la excepción en cada caso:

4. Rutina de tratamiento de excepciones

Cuando se está ejecutando un programa de usuario y tiene lugar una excepción o una interrupción, el procesador salta a un espacio determinado de la memoria para ejecutar la rutina de tratamiento de excepciones independientemente de la causa que la provoca. En el MIPS esta rutina de tratamiento está localizada en la dirección 0x80000180 dentro del kernel.
La rutina de tratamiento tiene que examinar los registros ***Cause*** y Status para determinar el origen de la excepción y decidir como tratarla. Algunas arquitecturas en lugar de utilizar el registro *Cause* para determinarlo, saltan directamente a diferentes rutinas de tratamiento para cada una de las diferentes fuentes de excepción.
El procesador después de tratar con la excepción vuelve a la dirección contenida en el registro ***EPC*** utilizando la instrucción `eret`. Esta instrucción tiene un efecto similar a la instrucción `jal` en la utilización del registro $ra cuando se ejecuta una función en MIPS.

En MIPS hay unas instrucciones que son concernientes al coprocesador 0 y que nos ayudarán en el diseño de la rutina de tratamiento de excepciones, se muestran en la tabla 3:

Instrucción	Significado	Comentarios
mfc0 rt,rs	rt $_{(CPU)}$← rs $_{(c0)}$	Copia el registro rs del coprocesador 0 al registro rt de la CPU
mtc0 rs, rt	rt $_{(C0)}$← rs $_{(CPU)}$	Copia el registro rs de la CPU al registro rt del coprocesador 0
eret	PC← EPC	Regreso de la rutina de tratamiento

Tabla 3: Instrucciones asociadas al coprocesador 0..

Se tiene que tener también con cuenta que la rutina de tratamiento se definirá en el segmento del kernel de la memoria por lo cual se tendrán que utilizar las directivas: `.kdata` y .ktext para definir los datos y el código de la rutina de tratamiento de excepciones. Además, se utilizarán los registres $k0 y $k1 como variables temporales en las rutinas de tratamiento de excepciones al estar almacenadas en el segmento ktext. La rutina de tratamiento se puede escribir en el mismo fichero donde está el programa de usuario o en un fichero separado.

Desde el punto de vista de la CPU, el proceso seguido cuando ocurre una excepción es el siguiente:
Al recibir el procesador una petición de excepción pone el bit 0 del registro *Status* a 1 para habilitar la interrupción. A continuación, rellena los bits del 2 al 6 del registro *Cause* con el código correspondiente de la tabla 1 que indica la fuente de la excepción. En el registro *EPC* almacenará la dirección de la instrucción donde se ha desencadenado

la excepción (si se trata de una interrupción, la instrucción todavía no ha empezado a ejecutarse). Si la excepción ocurre por acceso a una dirección inválida, en el registro *Vaddr* se guarda la dirección errónea. Por último, realiza un salto a la dirección 0x80000180 situada en el kernel donde se encuentra la rutina de tratamiento de excepciones. Si no hay ninguna instrucción en esa dirección, el MARS acaba el programa con un mensaje de error.

Desarrollo de la práctica

El propósito de esta práctica es tener la capacidad de diseñar e implementar una rutina de tratamiento de excepciones. Así, estudiaremos como funciona una rutina de tratamiento de excepciones software y como escribir una rutina de tratamiento de interrupciones. Las rutinas de tratamiento de excepciones se localizan en la parte de memoria kernel, dentro del sistema operativo. El kernel tiene privilegios extra y de protección sobre el código de usuario. Con el MARS se nos permitirá la posibilidad de acceder a este espacio protegido.

1. Rutina de tratamiento de excepciones software (traps)

Las acciones básicas de la rutina de tratamiento de excepciones son en primer lugar salvar el estado en el que se encuentra el procesador y después ejecutar la acción interna apropiada para la excepción ocurrida o bien, tratar con la fuente externa que ha provocado la interrupción. Una vez finalizado se tiene que restaurar el estado interno del procesador y volver al programa que estaba ejecutándose antes de que tuviera lugar la excepción.
Dicho de otro modo, la rutina de tratamiento de excepciones en primer lugar ha de preservar el contexto de ejecución del programa del usuario para que no haya ninguna interferencia con él. Es decir, tendrá que guardar en la pila el contenido del banco de registros, el contador de programa y cualquier otro dato que defina el estado en el que se encuentra el procesador en ese instante. En nuestro caso, cuando diseñamos una rutina de tratamiento será suficiente con que guardemos en la memoria los contenidos de los registros que se utilizarán a continuación. Este hecho podría suponer un problema en MIPS puesto que se necesita de un registro base para formar la dirección efectiva de la memoria, esto podría suponer una contradicción porque habría que modificar un registro antes de guardarlo. Por esta razón, el convenio de utilización de los registros del MIPS reserva los registres $k0 y $k1 para que la rutina de tratamiento de excepciones pueda utilizarlos. A partir de ese momento se puede utilizar la instrucción `mfc0` para examinar el registro *Cause* y determinar la causa que ha provocado la excepción y decidir las acciones a llevar a cabo. Cuando la rutina ha acabado y justo antes de volver al programa interrumpido con `eret` se tienen que restaurar los valores de los registros guardados. Conviene recordar que si la causa de la excepción no es una interrupción, se tiene que incrementar el contenido del registro EPC en 4.

Resumiendo, la rutina de tratamiento de excepciones tiene que llevar a cabo los siguientes pasos:

1. Guardar el estado del procesador (guardar registros en la memoria).
2. Identificar qué ha causado la excepción.
3. Tomar las medidas adecuadas en función del origen de la excepción.
4. Restaurar el valor de los registros guardados.
5. Regreso de la rutina de tratamiento de excepciones

A continuación, se muestra un ejemplo sencillo de una rutina de tratamiento de excepciones:

```
  ##################################################################
  ##        ÁREA DE DATOS DE LA RUTINA DE TRATAMIENTO         ##
  ##################################################################

.kdata
 registros: .word 0,0,0,0 # Espacio para guardar 4 registros

mis1:.asciiz"\nExcepción dirección errónea ocurrida en la
dirección: "
mis2:.asciiz "\nExcepción desbordamiento ocurrida en la
dirección: "
mis3:.asciiz "\nEn cualquier caso continuamos el programa...\n"

  ##################################################################
  ##  EMPIEZA CÓDIGO DE LA RUTINA DE TRATAMIENTO De EXCEPCIONES
##
  ##################################################################

.ktext 0x80000180  # Dirección de comienzo de la rutina

 # Salvar los registros a utilizar

 la $k1, registros
 sw $at, 0($k1) # Es importante guardar el registro $at
 sw $v0, 4($k1)
 sw $a0, 8($k1)

 mfc0 $a0, $13    # $a0 <= registro Cause
 andi $a0, $a0, 0x3C # extraemos en $a0 el código de excepción
 #Detectamos sólo dos excpcions
 li $s0, 0x0030 # código Desbordamiento
 li $s1, 0X0014 # código error de dirección store

 beq $a0, $s0, Desbordo
 bne $a0, $s1, salida
 la $a0, mis1
 li $v0, 4
 syscall

 mfc0 $a0, $14 # $a0 <= EPC, donde ha ocurrido la excepción
 li $v0, 34
 syscall # Escribimos EPC en hexadecimal

 Desbord:
 la $a0, mis2
 li $v0, 4
 syscall
 mfc0 $a0, $14 # $a0 <= EPC, donde ha  ocurrido la excepción
 li $v0, 34
 syscall # Escribimos EPC en hexadecimal

 salida:
 la $a0, mis3
 li $v0, 4
 syscall

 #Restauramos los registros
```

```
la $k1, registros
lw $at, 0($k1)
lw $v0, 4($k1)
lw $a0, 8($k1)

#Iniciamos registro Vaddr del coprocesador 0
mtc0 $zero, $8

#Cómo se trata de excepciones se actualiza el registro EPC
mfc0 $k0, $14 # $k0 <= EPC
addiu $k0, $k0, 4 # Incremento de $k0 en 4
mtc0 $k0, $14  # Ahora EPC apunta a la siguiente instrucción
eret # Vuelve al programa de usuario
```

- Estudia el código de la rutina de tratamiento de excepciones anterior. ¿Qué hace el programa?
- ¿Cuál es la secuencia de instrucciones que permite averiguar el código de excepción que ha causado la excepción?
- ¿Qué sucede si ocurre una excepción aritmética por división por 0?
- ¿Qué conjunto de instrucciones permiten incrementar el registro EPC en 4?
- ¿Que pasaría si no se incrementara el registro EPC en 4?
- ¿En qué casos se han utilizado los registres $k0 y $k1?
- ¿Por qué otra instrucción podrías sustituir la instrucción eret? ¿Cómo quedaría?
- Añade un programa principal que provoque una excepción por desbordamiento o dirección inválida y prueba el funcionamiento de la rutina de tratamiento de excepciones.

2. Rutina de tratamiento de interrupciones (excepciones hardware)

Como ya estudiamos en la práctica 10, MARS simula un terminal mapeado en memoria conectado al procesador. Recordad que esto significa que se accede a los dispositivos de E/S mediante unas determinadas posiciones de memoria de forma que si, por ejemplo, escribimos en la correspondiente dirección de memoria realmente estamos escribiendo en el dispositivo de salida y si leemos de una posición de memoria realmente estamos leyendo de un dispositivo de entrada.
El terminal de E/S que simula MARS consiste, como ya hemos visto, en dos unidades independientes, el receptor y el transmisor. El receptor lee caracteres desde el teclado y el transmisor escribe caracteres en el display del terminal. El procesador accederá al terminal a través de los dos registros de los dispositivos mapeados en memoria.
Recordad que los registros son los que se reproducen de nuevo en la figura 3:

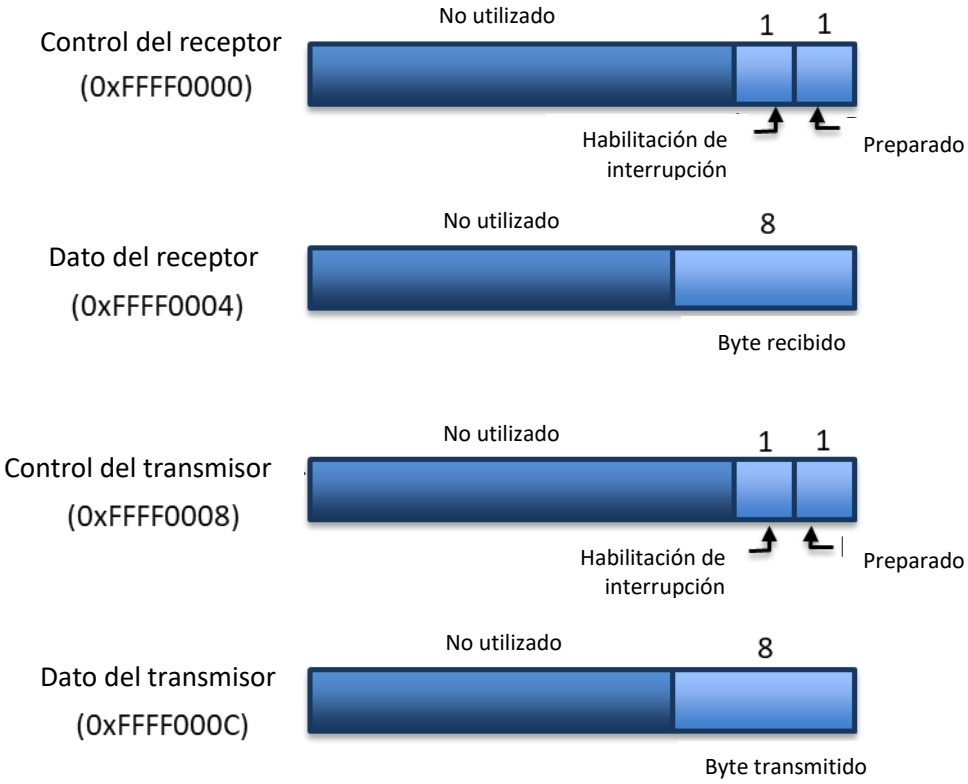

Figura 3. Registros de los dispositivos mapeados en memoria

El bit 0 (*Preparado*) de los registros de control cuando está activado (es decir, está a 1) indica que el dispositivo correspondiente está preparado para recibir o transmitir un carácter. El usuario puede leer o escribir el bit 1 de los registros de control llamado bit de ***Habilitación de interrupción***. Si el usuario la activa (lo pone a 1) entonces se realizará una petición de interrupción al procesador siempre que el bit de preparado esté activado. El byte menos significativo de los registros de datos del receptor y del transmisor se interpreta como un carácter ASCII y se leerá para una entrada desde el teclado (en el caso del receptor) o se escribirá para una salida en la consola (en el caso del transmisor).
La solicitud de una interrupción provocará que el procesador MIPS, al igual que ocurría con las excepciones, salta a la dirección 0x80000180 para ejecutar la rutina de tratamiento de interrupción. La diferencia con las excepciones software está en que al saltar a la rutina de tratamiento de interrupciones el procesador no ha empezado a ejecutar la instrucción en curso, por lo tanto en este caso no hay que incrementar el registro EPC en 4.

Los pasos generales que hay que seguir para diseñar una rutina de tratamiento de interrupciones son los siguientes. Primero habrá que deshabilitar las interrupciones para evitar que una rutina de tratamiento de interrupciones sea interrumpida por acontecimientos inesperados. Cuando se habilitan de nuevo, el hardware tratará las interrupciones pendientes cómo si hubieran acabado de ocurrir y se invocará inmediatamente a la rutina de tratamiento. El paso siguiente consiste en guardar el estado interno del procesador. Mientras el programa de usuario se está ejecutando puede haber una petición de interrupción. El procesador interrumpe el programa y tiene que atender la interrupción. El programa de usuario no sabe nada sobre lo que ha pasado y

tiene que continuar ejecutándose como si no hubiera pasado nada. Si se considera pertinente, a continuación, se pueden habilitar las interrupciones de prioridad más alta. Ahora la rutina realiza las acciones pertinentes que demanda la interrupción. Una vez ha acabado, se restaura el estado del procesador. El paso final es volver de la rutina de tratamiento al programa interrumpido y retomar la ejecución. Es deseable que las rutinas de tratamiento de interrupciones sean cortas puesto que el procesador a menudo tiene muchos de tipos diferentes. Si ocurre que una interrupción solicita un servicio mientras se está procesando otra se tiene que tener la seguridad de que la segunda sea atendida en tiempo para lo cual interesa que el tratamiento de la primera no sea muy largo.

Un ejemplo sencillo de rutina de tratamiento de interrupciones se puede ver a continuación:

```
 # Reserva de espacio para guardar registros en kdata
.kdata

contexto:  .word 0,0,0,0  # espacio para alojar cuatro registros


  .ktext 0x80000180    # Dirección de comienzo de la rutina


 # Guardar registros a utilizar en la rutina.

 la $k1, contexto
 sw $at, 0($k1)          # Guardamos $at
 sw $t0, 4($k1)
 sw  $v0, 8($k1)
 sw $a0, 12($k1)

 #Comprobación de si se trata de una interrupción
 mfc0 $k0, $13           # Registro Cause
 srl $a0, $k0, 2         # Extraemos campo del código
 andi $a0, $a0, 0x1f
 bne $a0, $zero, acabamos       # Sólo procesamos aquí E/S

 #Tratamiento de la interrupción

 li $t0, 0xffff0000   #
 lb $a0, 4($t0)       # Lee carácter del teclado

  # Por ejemplo:
  # Escribe en la consola del MARS el carácter leído
 li $v0, 11
 syscall

 # Antes de acabar se podría dejar todo iniciado:

 acabamos:  mtc0 $0, $13        # Iniciar registro Cause
            mfc0 $k0, $12       # Leer registre Status
            andi $k0, 0xfffd    # Iniciar bit de excepción
            ori  $k0, 0x11      # Habilitar interrupciones
            mtc0 $k0, $12       # reescribir registre Startus

 # Restaurar registros

            lw $at, 0($k1)      # Recupero $at
            lw    $t0, 4($k1)
```

```
            lw    $v0, 8($k1)
            lw    $a0, 12($k1)

# Devolver en el programa de usuario

            eret
```

- Estudia el código de la rutina de tratamiento de interrupciones anterior. ¿Qué hace la rutina para dar servicio a la interrupción? ¿De donde proviene la interrupción?
- ¿Cuál es la secuencia de instrucciones que permite averiguar si la excepción ocurrida se debida a una interrupción?
- ¿Cuáles diferencias se observan entre la rutina de tratamiento de interrupciones y la rutina de tratamiento de excepciones?
- ¿Podrían incluirse los dos tratamientos en una misma rutina?

Para probar la rutina de tratamiento de interrupciones diseñada se requiere de un programa de usuario que acepte peticiones de interrupciones. A continuación, se muestran los pasos que se tienen que seguir para escribir uno:

1. Es necesario preparar la interfaz de usuario, lo que supone habilitar las interrupciones de los dispositivos de E/S asociados. Para lo cual se tienen que seguir dos pasos:
    a. Primero hace falta decirle al dispositivo de E/S que puede enviar peticiones de interrupción. En el MARS esto significa habilitado el bit de interrupción en el registro de control del receptor y en el registro de control del transmisor. De momento sólo utilizaremos el teclado entonces se habilitará el bit de interrupciones del registro de control del teclado.

```
lui $t0,0xffff          # Dirige del registro de control
lw $t1,0($t0)           # Registre de control del receptor
ori $t1,$t1,0x0002      # Habilitar interrupciones del teclado
sw $t1,0($t0)           # Actualizamos registro de control
```

    b Segundo hace falta decirle al coprocesador 0 que acepte peticiones de interrupción. Esto se hace habilitando las interrupciones específicas y habilitando el bit global de interrupción en el registro *Status ($12)* del coprocesador 0. Los bits 8 al 15 del registro *Status* permiten 8 niveles diferentes de interrupción a pesar de que el MARS sólo utiliza los bits 3 y 4 de la máscara de interrupciones (los bits 11 y 12 del registro de estado) que habilitan el teclado y la consola respectivamente. Cómo no habrá ningún problema, habilitaremos todos los bits de la máscara.

```
mfc0 $a0, $12           # leer registre Status
ori $a0, 0xff11         # Habilitar todas las interrupciones
mtc0 $a0, $12           # reescribir el registro status
```

2. Hay que escribir un programa principal del usuario.
   Un programa de prueba podría consistir por ejemplo con un mensaje inicial en la consola del MARS seguido de la ejecución de un bucle infinito a la espera de que se produzca una solicitud de interrupción.

3. El último paso es acabar el programa. Conviene acabarlo con un mensaje en la consola del MARS seguido de la secuencia normal de salida:

```
li $v0, 10
 syscall    # syscall 10 (exit)
```

➢ Utiliza la rutina de tratamiento de interrupciones y añádele un programa de prueba como se indica anteriormente. Haz distintas pruebas de ejecución.

Ejercicios a entregar

➢ Modifica la rutina de tratamiento de interrupciones para que escriba en el display del transmisor el carácter leído en el receptor. Haz que guarde en el registro $v0 el carácter leído. Escribe un programa principal apropiado para hacer pruebas que finalice cuando en el receptor se pulse un salto de línea.

➢ Escribe una rutina general de tratamiento de excepciones que permita tratar excepciones por desbordamiento aritmético, error por lectura al intentar el acceso a una dirección no alineada e interrupciones de teclado. En los tres casos se tiene que escribir un mensaje en la consola del MARS de la excepción tratada. Escribe el programa de prueba apropiado para probar los tres casos.

Resumen

- Una excepción software o hardware provoca un salto inesperado para ejecutar la rutina de tratamiento de excepciones.
- La rutina de tratamiento de excepciones se encuentra en la dirección 0x80000180 dentro del kernel.
- La rutina de tratamiento de excepciones tiene que preservar el estado interno del procesador.