Práctica 11 Entrada/ salida 2

Jordi Blasco Lozano

Arquitectura de comuputadores

Grado en Inteligencia Artificial

Indice:

1. Actividad 1

Observa el código ejemplo de la actividad 1. ¿Qué instrucción causará la excepción?

La excepción la causará la instrucción addi ya que generará un desbordamiento al sumar 1, mientras que addiu no detectará el overflow.

Ensambla y ejecuta el código paso a paso. Observa el cambio de los registros del coprocesador 0.

En el addiu no lo detecta

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$8 (vaddr)	8	0
$12 (st...	12	65297
$13 (ca...	13	0
$14 (epc)	14	0

Mientras que con el addi si

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$8 (vaddr)	8	0
$12 (st...	12	65299
$13 (ca...	13	48
$14 (epc)	14	4194316

2. Cuestión 1

¿Cuáles son los valores de los registros del coprocesador 0 antes y después de producirse la excepción en el código de la actividad 1?

Vaddr muestra 0 ya que no hay ninguna referencia de memoria inválida

Status muestra 65297 al principio, 0 en el bit 1 y 65299 al final, 1 en el bit 1

Cause muestra 48 y sus bits de (2 a 6) 12

Epc muestra 4194316 la direccion de momoria donde se ha producido la excepción

3. Cuestión 2

¿Cuál es el significado de los distintos campos de los registros del coprocesador 0 después de producirse la excepción en el código de la actividad 1?

Vaddr no tiene valor porque no existe ninguna referencia de memoria inválida

Status indicaba en su bit 1 (el bit de nivel de excepción) un 0 pero al ocurrir una excepción cambia a 1 previniendo así que una nueva excepción interrumpa la rutina de tratamiento de excepciones, es por eso por lo que se incrementa el valor en 2, 2 elevado a la posición(1).

En el registro de "causa" (que indica las interrupciones pendientes y el tipo de excepción) muestra 48 que representa excepción por overflow aricmético ya que contando bit 2 al 6 (que son dos divisiones entre 2) nos da 12 (overflow)

En "epc" muestra la direccion de la instrucción donde ocurrió la excepción

4. Actividad 2

Analiza, ensambla y ejecuta el código.

El codigo proporcionado nos genera una excepción debido a que la dirección de memoria a la que queremos acceder pertenece al Kernel y no la encuentra

```
Runtime exception at 0x00400004: address out of range 0x0000007c
```

5. Cuestión 3

¿Cuáles son los valores de los registros del coprocesador 0 antes y después de producirse la excepción en el código de la actividad 2?

Antes de producirse la excepción

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$8 (vaddr)	8	0
$12 (status)	12	65297
$13 (cause)	13	0
$14 (epc)	14	0

Despues

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$8 (vaddr)	8	124
$12 (status)	12	65299
$13 (cause)	13	20
$14 (epc)	14	4194308

6. Cuestión 4

¿Cuál es el significado de los distintos campos de los registros del coprocesador 0 después de producirse la excepción en el código de la actividad 2?

El valor de vaddr en este caso si que nos da la referencia a la memoria invalida, viene a ser la 124 que habiamos ingresado anteriormente

El valor de Status cambia de nuevo su valor del bit 1 de 0 a 1 para prevenir que una nueva excepción interrumpa la rutina de tratamiento de excepciones

El valor de causa es de 20, si contamos del bit 2 al 6 nos da 5 (La excepcion por dirección errónea)

Epc nos vuelve a mostrar la dirección de la instrucción donde se produjo la excepción

7. Actividad 3

Analiza, ensambla y ejecuta el código.

El código proporcionado causa una excepción debido a que la palabra de 4 bytes que se intenta cargar desde la dirección de memoria no está alineada correctamente, ya que se trata de acceder a una dirección de memoria que no es multiplo de 4, por lo que nos salta la siguiente excepción.

```
Runtime exception at 0x00400008: fetch address not aligned on word boundary 0x10010003
```

8. Cuestión 5

¿Cuáles son los valores de los registros del coprocesador 0 antes y después de producirse la excepción en el código de la actividad 3?

Antes:

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$8 (vaddr)	8	0
$12 (status)	12	65297
$13 (cause)	13	0
$14 (epc)	14	0

Después:

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$8 (vaddr)	8	268500995
$12 (status)	12	65299
$13 (cause)	13	16
$14 (epc)	14	4194312

9. Cuestión 6

¿Cuál es el significado de los distintos campos de los registros del coprocesador 0 después de producirse la excepción en el código de la actividad 3?

El valor de Vaddr nos muestra la dirección de memoria a la que el programa trata de acceder sin éxito. La (0x10010000) + 3. La primera dirección disponible más 3.

Status sigue habilitando todas las exepciones y cambiando su bit 1 de 0 a 1 cuando se produce la excepción.

En este caso Cause nos muestra el codigo de la exepción 4 (Excepción por dirección errónea) en los bits del 2 al 6

Y Epc nos muestra la dirección de la instrucción donde se ha producido la excepción

10. Cuestión 7

Rellena la tabla indicando cual ha sido la causa que ha provocado la excepción en cada caso

Cause	Fuente de la excepción
0x00000000	Interrupción (0)
0x00000020	Excepción syscall (8 del bit 2 al 6)
0x00000024	Excepción por punto de ruptura (breakpoint, 9 del bit 2 al 6)
0x00000028	Excepción por instrucción reservada (10 del bit 2 al 6)
0x00000030	Excepción por desbordamiento aritmético (12 del bit 2 al 6)

Según la tabla proporcionada en la práctida de codigos de excepciones

11. Actividad 4

Estudia el código de la rutina de tratamiento de excepciones de la actividad 4. ¿Qué hace el programa?

El programa muestra al usuario mensajes en la consola en el caso que se produzcan excepciones de desbordamiento o de error en la dirección de memoria.

En kdata. Se almacenan los mensajes de error y se reserva espacio para almacenar 4 registros.

Se extrae el codigo de la excepción con un andi entre el valor de cause guardado en $a y 0x3C para obtener los bits del valor de excepcion y se guardan de nuevo en $a0.

Si el valor de la excepción es el mismo que el de desbordamiento se lanza el mensaje que corresponde y se aumenta el valor de epc en 4

¿Cuál es la secuencia de instrucciones que permite averiguar el código de excepción que ha causado la excepción?

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

Después de extraer el codigo de los errores se usa beq para comparar y saltar a la etiqueta que muestra el error

¿Qué sucede si ocurre una excepción aritmética por división por 0?

Como el programa no trata esta excepción se mostraría el mismo mensaje que se mostraría en un programa sin la modificación del kernel.

¿Qué conjunto de instrucciones permiten incrementar el registro EPC en 4?

```
mtc0 $zero, $8
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
```

¿Que pasaría si no se incrementara el registro EPC en 4?

Si el registro epc no se incrementa en 4 se volvería a lanzar la instrucción causante de la excepción en bucle.

¿En qué casos se han utilizado los registros $k0 y $k1?

$k0 se utiliza para almacenar la dirección de la instrucción que causó la excepción para el usuario regrese al programa desde la instrucción siguiente a la que causo la excepcción

Y $k1 se utiliza para calcular las direcciones de memoria donde se guardan y restauran los registros necesarios para mantener el contexto de las excepciones.

---

12. Cuestión 8

¿Por qué otra instrucción podrías sustituir la instrucción eret en la rutina de tratamiento de excepciones de la actividad 4? ¿Cómo quedaría?

Podriamos sustituirla por una instrucción de tipo J, sustituimos la instrucción “eret” por ”jr $a0”

---

13. Cuestión 9

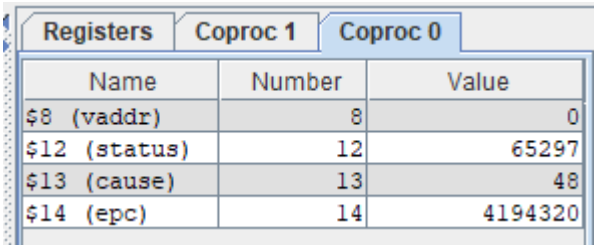
Añade un programa principal a la rutina de tratamiento de excepciones de la actividad 4 que provoque una excepción por desbordamiento o dirección inválida y prueba el funcionamiento de la rutina de tratamiento de excepciones.

Insertamos

```
.text
li $t0, 0x7FFFFFFF
addiu $t1, $t0,1 #Se ignora el desbordamiento
addi $t2, $t0, 1 #Detecta el desbordamiento
```

Al darle play nos mostrará la excepción con el mensaje que guardamos y ademas aparecerá en el registro de causa la excepción pero no se activará el bit 1 de status ya que la excepción la estamos tratando nosotros desde el kernel.

```
Excepción desbordamiento ocurrida en la dirección: 0x0040000c
En cualquier caso continuamos el programa...

-- program is finished running (dropped off bottom) --
```



Registers | Coproc 1 | Coproc 0

Name	Number	Value
$8 (vaddr)	8	0
$12 (status)	12	65297
$13 (cause)	13	48
$14 (epc)	14	4194320

14. Actividad 5

Estudia el código de la rutina de tratamiento de interrupciones de la actividad 5. ¿Qué hace la rutina para dar servicio a la interrupción? ¿De donde proviene la interrupción?

La rutina de tratamiento de interrupciones comprueba si se trata de una interrupción comprobando con una andi si corresponde a al valor de "excepción" de una interrupción.

La interrupción proviene del teclado.

¿Cuál es la secuencia de instrucciones que permite averiguar si la excepción ocurrida se debida a una interrupción?

```
#Comprobación de si se trata de una interrupción
mfc0 $k0, $13
srl $a0, $k0, 2
andi $a0, $a0, 0x1f
bne $a0, $zero, acabamos
```

El andi guaradará en $a0 el valor de la "execpción" en negativo y se comprarará y si da 0 segurá con el tratamiento de la interrupción y si es negativo abará el programa

¿Cuáles diferencias se observan entre la rutina de tratamiento de interrupciones y la rutina de tratamiento de excepciones?

La rutina de execpciones se usa para fallos de ejecución de un programa mientras que el tratamiento de interrupciones se usa para manejar las entradas de dispositivos externos

¿Podrían incluirse los dos tratamientos en una misma rutina?

Si, se podría ya que ambos comparten kernel, además podría pasar que una excepción la cause una interrupción.

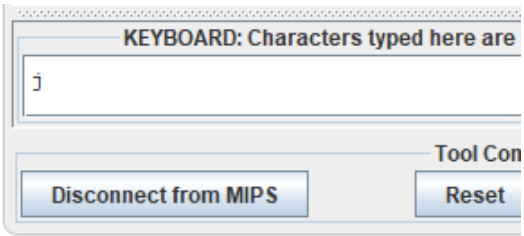
15. Cuestión 10

Utiliza la rutina de tratamiento de interrupciones de la actividad 5 y añádele un programa de prueba como se indica anteriormente. Haz distintas pruebas de ejecución.

```
.data
inicio_mensaje: .asciiz "Programa de prueba de interrupciones en ejecución...\n"

.text
# Habilitar interrupciones del teclado
lui $t0, 0xffff
lw $t1, 0($t0)              # Cargar el contenido del registro de control del receptor
ori $t1, $t1, 0x0002        # Habilitar interrupciones del teclado (bit 1)
sw $t1, 0($t0)              # Escribir de vuelta al registro de control del receptor

# Habilitar todas las interrupciones
mfc0 $a0, $12               # Leer el registro Status
ori $a0, $a0, 0xff11        # Habilitar todas las interrupciones (bits 0-7 y 11-15)
mtc0 $a0, $12               # Escribir de vuelta al registro Status

# Programa principal del usuario

# Mensaje inicial en la consola
li $v0, 4                   # Cargar la llamada al sistema para imprimir cadena
la $a0, inicio_mensaje      # Cargar la dirección del mensaje inicial
syscall                     # Llamar al sistema para imprimir el mensaje

# Bucle infinito esperando interrupciones
bucle:
 j bucle                    # Salto incondicional al principio del bucle

# Fin del programa

# Mensaje de salida en la consola


# Finalizar programa
li $v0, 10                  # Llamada al sistema para salir
syscall
```

El programa finalizó al escribir en el teclado

```
-- program is finished running (dropped off bottom) --

Programa de prueba de interrupciones en ejecución...

-- program is finished running (dropped off bottom) --
```



16. Cuestión 11

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.

```
#rutina de tratamiento del teclado arriba

teclado_interrupt:
    # Leer el carácter del registro del receptor
    lw $v0, 0($a0)          # Guardar el carácter leído en $v0

    # Escribir el carácter en el display del transmisor
    sw $v0, 0($a1)          # Escribir el carácter en el display del transmisor

    # Comprobar si el carácter leído es un salto de línea
    li $t0, '\n'            # Cargar el carácter de salto de línea
    beq $v0, $t0, fin_interrupt # Salir de la rutina si el carácter leído es un salto de línea

    jr $ra                  # Retornar de la interrupción

fin_interrupt:
    # Finalizar la rutina de interrupción
    mtc0 $zero, $13         # Limpiar la bandera de interrupción
    jr $ra                # Retornar de la interrupción

.data
receptor_addr: .word 0x10010000    # Dirección del registro del receptor
transmisor_addr: .word 0x10010004  # Dirección del registro del transmisor

# Mensaje inicial en la consola
inicio_mensaje: .asciiz "Programa de prueba de interrupciones en ejecución...\n"
fin_mensaje: .asciiz "Fin del programa.\n"

# Habilitar interrupciones del teclado
lui $t0, 0xffff
lw $t1, 0($t0)                # Cargar el contenido del registro de control del receptor
ori $t1, $t1, 0x0002            # Habilitar interrupciones del teclado (bit 1)
sw $t1, 0($t0)                # Escribir de vuelta al registro de control del receptor
mfc0 $a0, $12                 # Leer el registro Status
ori $a0, $a0, 0xff11            # Habilitar todas las interrupciones (bits 0-7 y 11-15)
mtc0 $a0, $12                 # Escribir de vuelta al registro Status


# Direcciones de los registros del receptor y del transmisor

.text
# Llamada a la rutina de tratamiento de interrupciones
la $a0, receptor_addr           # Pasar la dirección del registro del receptor como argumento
la $a1, transmisor_addr          # Pasar la dirección del registro del transmisor como argumento
li $v0, 10                    # Llamada al sistema para habilitar interrupciones
syscall

# Bucle esperando interrupciones
bucle:
```

```
 j bucle                    # Bucle infinito

# Finalizar programa
fin:
  # Imprimir mensaje de salida en la consola
  li $v0, 4                 # Llamada al sistema para imprimir cadena
  la $a0, fin_mensaje           # Cargar la dirección del mensaje de salida
  syscall

  # Salir del programa
  li $v0, 10                # Llamada al sistema para salir
  syscall
```

17. Cuestión 12

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.

```
.data
overflow_msg: .asciiz "¡Excepción de desbordamiento aritmético!\n"
alignment_error_msg: .asciiz "¡Excepción de dirección no alineada!\n"
keyboard_interrupt_msg: .asciiz "¡Interrupción de teclado!\n"
unknown_exception_msg: .asciiz "¡Excepción desconocida!\n"

#rutina de tratamiento del teclado arriba

exception_handler:
    # Identificar el tipo de excepción
    mfc0 $t0, $13               # Leer el registro de causa de excepción
    li $t1, 0x80000000          # Bit de excepción de interrupción de teclado
    and $t2, $t0, $t1           # Comprobar si la excepción es de interrupción de teclado
    bnez $t2, handle_keyboard_interrupt # Si es una excepción de teclado, ir a manejar la interrupción

    li $t1, 0x2                 # Bit de excepción de desbordamiento aritmético
    and $t2, $t0, $t1           # Comprobar si la excepción es de desbordamiento aritmético
    bnez $t2, handle_overflow     # Si es una excepción de desbordamiento, ir a manejar el desbordamiento

    li $t1, 0x4                 # Bit de excepción de dirección no alineada
    and $t2, $t0, $t1           # Comprobar si la excepción es de dirección no alineada
    bnez $t2, handle_alignment_error # Si es una excepción de dirección no alineada, ir a manejar el error

    # Si no es ninguno de los tipos de excepción anteriores, imprimir un mensaje de excepción desconocida
    la $a0, unknown_exception_msg # Cargar la dirección del mensaje de excepción desconocida
    li $v0, 4                   # Llamada al sistema para imprimir cadena
    syscall

    j end_exception_handler       # Salir de la rutina de manejo de excepciones

handle_keyboard_interrupt:
    # Imprimir mensaje de excepción de interrupción de teclado
    la $a0, keyboard_interrupt_msg # Cargar la dirección del mensaje de interrupción de teclado
```

```
    li $v0, 4                   # Llamada al sistema para imprimir cadena
    syscall

    j end_exception_handler     # Salir de la rutina de manejo de excepciones

handle_overflow:
    # Imprimir mensaje de excepción de desbordamiento aritmético
    la $a0, overflow_msg        # Cargar la dirección del mensaje de desbordamiento aritmático
    li $v0, 4                   # Llamada al sistema para imprimir cadena
    syscall

    j end_exception_handler     # Salir de la rutina de manejo de excepciones

handle_alignment_error:
    # Imprimir mensaje de excepción de dirección no alineada
    la $a0, alignment_error_msg # Cargar la dirección del mensaje de error de alineación
    li $v0, 4                   # Llamada al sistema para imprimir cadena
    syscall

end_exception_handler:
    # Finalizar la rutina de manejo de excepciones
    mtc0 $zero, $13             # Limpiar la bandera de excepción
    jr $ra                      # Retornar
```