Organización y Arquitectura de Computadoras
2020-1
Práctica 10: Excepciones

Profesor: José de Jesús Galaviz Casas
Roberto Monroy Argumedo

1. Objetivos

Generales:

- El alumno conocerá la interacción del hardware con el software en el manejo de interrupciones y excepciones.

Particulares:

Al finalizar la práctica, el alumno:

- Tendrá la capacidad de escribir manejadores de excepciones en el lenguaje ensamblador MIPS.

2. Requisitos

- **Conocimientos previos:**
  - Instrucciones aritméticas, lógicas y de manejo de flujo de programas del lenguaje ensamblador MIPS.
  - Convención de llamadas a subrutinas.
  - Llamadas al sistema.
  - Codificación ASCII.
- **Tiempo de realización sugerido:**
  10 horas.
- **Número de colaboradores:**
  Equipos de 2 integrantes.
- **Software a utilizar:**
  - *Java Runtime Environment* versión 5 o superior.
  - El paquete MARS [**Mars**].

3. Planteamiento

Una **excepción**, es una señal enviada al procesador por el hardware o el software originada por una situación inesperada que requiere atención inmediata, alterando el flujo normal de la ejecución de instrucciones. El procesador detiene la actividad que se encuentre ejecutando y cede el control a un **manejador**, una rutina especial que, dependiendo de la situación, realiza las acciones adecuadas para resolver la excepción. Las dividiremos según su fuente:

- Excepciones externas. Para comunicarse con el procesador, los dispositivos de entrada y salida activan una excepción, también se le denomina **interrupción**[1]. Por ejemplo: cuando el usuario presiona alguna tecla.
- Excepciones internas. Cuando ocurre algún tipo de error o el procesador envía un mensaje a un dispositvo, por ejemplo: un desbordamiento aritmético, cuando un programa intenta acceder a secciones de memoria no disponibles o cuando una rutina reproduce sonido.

En esta práctica se desarrollará una calculadora que aprovechará el mecanismo de manejo de excepciones para resolver errores que se puedan presentar[2].

4. Desarrollo

4.1. Excepciones en MARS

En la arquitectura MIPS-32, el **coprocesador 0** es el encargado de registrar la información necesaria para manejar las excepciones. En la tabla **??** se muestran los registros del coprocesador disponibles en el simulador MARS.

Número	Nombre	Uso
`$8`	`vaddr`	Dirección de memoria inválida
`$12`	`status`	Estado del procesador
`$13`	`cause`	Causa de la excepción
`$14`	`EPC`	Dirección de la instrucción causante de la excepción

Tabla 1: Registros del coprocesador 0 disponibles en MARS.

4.1.1. Registro status

El registro `status` describe el estado general del procesador, MARS implementa solo algunos de los bits de la arquitectura, éstos se describen en la tabla

[1] Algunas arquitecturas usan el término *interrupción* para todas las excepciones, seguimos la convención de MIPS en la que el término *interrupcion* se usa sólo para excepciones externas.

[2] En un sistema de cómputo, el sistema operativo es el encargado de resolver las excepciones. Un programa, como la calculadora aquí propuesta, no tiene los permisos necesarios para manejarlas y solo funciona para ilustrar el mecanismo.

??. La máscara de interrupciones cuenta con 8 bits, 6 bits para interrupciones y 2 para excepciones de software, si el valor del bit es 1, la excepción estará habilitada, es decir, el procesador podrá ser interrumpido por la excepción, si el valor es 0 la excepción será ignorada.

Bit	Nombre	Significado
0	`EX`	Control maestro de las interrupciones: ▪ 0 - Todas las interrupciones son ignoradas. ▪ 1 - Las interrupciones están habilidatas.
1	`EXL`	Nivel de excepción: ▪ 0 - Nivel normal. ▪ 1 - Nivel de excepción: • Las interrupciones se deshabilitan. • `EPC` y `Cause` no se actualizan si ocurre otra excepción.
4	`UM`	Modo usuario: ▪ 0 - Modo supervisor (no implementado en MARS). ▪ 1 - Modo usuario.
8-15	`IM0-IM7`	Máscara de interrupción.

Tabla 2: Bits del registro `status` implementados en MARS

4.1.2. Registro cause

En el registro `cause` se describe la causa de la excepción más reciente. En la tabla **??** se describen los bits implementados por MARS y en la tabla **??** los códigos de interrupción.

Bit	Nombre	Significado
2 - 6	`ExcCode`	Código de la excepción
8 - 15	`IP 0 - 7`	Interrupciones pendientes

Tabla 3: Bits del registro `cause` imlementados en MARS

4.1.3. Traps

Además de las excepciones activadas automáticamente por el procesador durante la ejecución del programa, existe un tipo de excepción que puede ser

Código	Mnemónico	Excepción
4	`AdEL`	Dirección inválida (Load)
5	`AdES`	Dirección inválida (Store)
8	`Sys`	Llamada al sistema
9	`Bp`	Breakpoint
10	`RI`	Instrucción reservada
12	`Ov`	Desbordamiento aritmético
13	`Tr`	Trap
15	`DivZ`	División por cero
16	`FPOv`	Desbordamiento en punto flotante
17	`FPUn`	Subdesbordamiento en punto flotante

Tabla 4: Códigos de excepción implementados en MARS.

activada por el programador de ensamblador en cualquier momento. Dichas excepciones se denominan *traps*. MARS implementa las instrucciones descritas en la tabla **??** para causar excepciones de tipo trap.

Mnemónicos	Significado
`teq teqi`	Trap if equal
`tne tnei`	Trap if not equal
`tge tgeu tgei tgeiu`	Trap if greater of equal
`tlt tltu tlti tltiu`	Trap if less than

Tabla 5: Excepciones trap implementadas en MARS.

4.1.4. Manejador de excepciones

Cuando ocurre una exepción, el procesador realiza las siguientes acciones:

- Guarda la dirección de la instrucción causante en el registro `EPC` (*Exception Program Counter*).
- Guarda el código de la excepción en el registro `cause`.
- Cambia el bit de `EXL` a 1.
- En caso de que la excepción fuera causada por un acceso inválido de memoria, guarda la dirección inválida en el registro `vaddr`.
- Finalmente, brinca a la dirección de memoria `0x81000080`.

En esta dirección se encuentra una rutina denominada **manejador de excepciones**. Éste debe analizar los registros del coprocesador 0 para deteminar la causa y tomar las acciones necesarias para solucionar la excepción. Las únicas instrucciones con las que se puede modificar dichos registros son: `mfc0` (*move*

from coprocessor 0), y `mtc0` (*move to coprocessor 0*). El manejador dispone de los registros `$k0` y `$k1` para realizar sus tareas, en caso de necesitar más, puede disponer de los demás registros de propósito general, por lo que es necesario que respalde en la memoria la información contenida en éstos. Una vez concluido el manejo de la excepción, el manejador debe:

- Restaurar la información de los registros que haya modificado.
- Actualizar el registro `EPC` con la dirección de memoria de la siguiente instrucción que se debe ejecutar al terminar el manejo de la excepción.
- Ejecutar la instrucción `eret` (*exception return*), la cual cambia el bit `EXL` a 0 (permitiendo nuevas excepciones) y brinca a la dirección de memoria almacenada en `EPC`.

Los espacios de memoria de texto y datos correspondientes para el manejador son espacios reservados, por lo que se debe usar las directivas `.kdata` y `.ktext` para que el ensamblador guarde la información en el espacio adecuado.

4.2. La calculadora

4.2.1. Notación posfija

En la notación posfija, a diferencia de la notación infija, primero se escriben los operandos y después la opeación que se va a realizar. En la tabala **??** se encuentran varios ejemplos, se puede notar que se elimina por completo la necesidad de usar paréntesis.

Notación infija	**Notación posfija**
$1 + 2$	$1\ 2\ +$
$3 * (1 + 2)$	$3\ 1\ 2\ +\ *$
$3 * 1 + 2$	$3\ 1\ *\ 2\ +$
$(3 * (1 + 2) + 3 - 4)/6$	$3\ 1\ 2\ +\ *\ 3\ 4\ -\ +\ 6\ /$

Tabla 6: Ejemplos de notación posfija.

La evaluación se realiza leyendo la expresión de izquierda a derecha, con ayuda de una pila y usando el siguiente algoritmo:

1. Mientras haya elementos en la expresión:
   a) Si el elemento es un operando, se pone en la pila.
   b) Si el elemento es un operador:
      1) Tomar los primeros dos elementos de la pila como operandos. Si no hay dos elementos en la pila, ocurre un error: el usuario ha introducido mal la expresión.
      2) Evaluar la operación.
      3) Poner el resultado en la pila.

2. Si se ha terminado la cadena y solo hay un elemento en la pila, se regresa como resultado. Si hay más de un elemento, ocurre un error: el usuario ha introducido mal la expresión.

4.2.2. Calculadora

Desarrollaremos un calculadora aritmética que reciba la entrada en notación posfija, realice las operaciónes y muestre el resultado en pantalla. Deberá cumplir las siguientes especificaciones:

1. La entrada y la salida será por medio del simulador de terminal de MARS.
2. El calculadora imprimirá en la terminal un *prompt* (un caracter o secuencia de caracteres) para indicar al usuario que espera una expresión, el usuario podrá escribir la expresión la terminal, la calculadora evaluará la expresión, mostrará el resultado en la terminal y repetirá el proceso.
3. Las operaciones que puede realizar son: suma, resta, multiplicación y división entera.
4. Para terminar la ejecución de la calculadora, el usuario deberá escribir el comando *exit* como entrada.
5. La calculadora deberá resolver los errores que puedan ocurrir por medio del mecanismo de manejo de excepciones, éstos son:

Error	**Excepción**
Desbordamiento aritmético	`Ov`
División por cero	`DivZ`
Expresión invalida	`Tr`

   El manejador deberá imprimir en la terminal un mensaje con la causa del error e información como los operandos que causaron el error o el número de carcater de la cadena de entrada donde ocurrió el problema. Al finalizar la rutina del manejador, la calculadora se deberá encontrar lista para recibir otra expresión.

5. Entrada

La calculadora será interactiva, el usuario introducirá la entrada en el simulador de terminal de MARS.

6. Salida

La calculadora mostrará el resultado en el simulador de terminal de MARS.

7. Variables libres

No hay variables libres para el desarrollo de esta práctica.

8. Procedimento

Escribe las rutinas necesarias en lenguaje ensamblador de MIPS respetando los siguientes lineamientos:

- Usa la convención de llamadas a subrutinas.
- Agrega una rutina *main* en donde se carguen de la memoria los argumentos en los registros adecuados y se llame a la subrutina que resuelva el ejercicio.
- El programa debe terminar con la llamada al sistema *exit*.
- Deberás entregar un único archivo de código fuente.
- No olvides documentar el código.

9. Ejercicios

1. Desarrolla la calculadora propuesta en la sección **??**.

10. Preguntas

1. En un procesador, ¿qué es el *modo supervisor*? ¿Qué funciones tiene? ¿Cómo se implementa?
2. ¿Cuál es la relación entre una llamada al sistema y una excepción?
3. ¿Qué es un vector de interrupciones?