



**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

# **ORGANIZACIÓN DE COMPUTADORES**

**GRADO EN INGENIERÍA DE COMPUTADORES**

## **ACTIVIDADES PRÁCTICAS**

**ACTIVIDAD 2.1:**

**EXCEPCIONES INTERNAS EN ARQUITECTURAS MIPS32**

**CURSO 2019-20**

**Luis Rincón Córcoles**  
[luis.rincon@urjc.es](mailto:luis.rincon@urjc.es)

## Prólogo

En este cuaderno de prácticas se proponen diversas actividades sobre gestión de excepciones en arquitecturas MIPS32. Los ejercicios se realizarán en el simulador de MIPS **MARS** en diferentes condiciones de respuesta ante excepciones:

- Sin cargar ningún manejador de excepciones.
- Con un manejador de excepciones genérico para todas las excepciones.
- Con un manejador que presenta mensajes diferentes en función de la excepción producida.
- Con un manejador que atiende las excepciones de forma específica, con diferentes rutinas de tratamiento para diferentes excepciones.

Junto con estas actividades se adjuntan varios cuestionarios que se responderán a través del aula virtual. La nota obtenida en los mismos computa para el cálculo de la nota final de la asignatura, dentro del apartado de tests sobre actividades de laboratorio. Esta actividad es **optativa**.

## 2. 1. 1. LECTURA PREVIA

### INTRODUCCIÓN A LAS EXCEPCIONES EN MIPS

Las excepciones son situaciones que requieren un tratamiento especial por parte del procesador. Según la causa que las genere, se pueden distinguir dos tipos de excepciones: internas y externas.

Una **excepción interna** es un suceso inesperado del procesador acaecido por causas internas o inherentes a la ejecución de las instrucciones. Entre las causas de una excepción interna se encuentran las siguientes:

- Dirección errónea en lectura de un dato o una instrucción.
- Dirección errónea en escritura de un dato.
- Error de bus en lectura de instrucción.
- Error de bus en acceso a dato.
- Llamada a sistema: se produce cuando el programa en ejecución solicita un servicio al sistema operativo. En MIPS, para ello se emplea la instrucción **syscall**.
- Punto de ruptura.
- Instrucción reservada: se produce cuando hay un intento de ejecutar una instrucción privilegiada por parte de un programa de usuario.
- Instrucción indefinida: esta excepción se produce cuando se intenta ejecutar una instrucción que tiene un código de operación erróneo.
- Trampa (**trap**): en MIPS, las instrucciones **trap** comprueban una condición, y si se cumple, generan una excepción (ver apéndice A).
- Desbordamiento en operación aritmética: se produce cuando hay desbordamiento al sumar o restar dos números en complemento a 2.
- División por 0 en coma flotante: se produce cuando el divisor es 0 en una operación de división.
- Desbordamiento en coma flotante.
- Subdesbordamiento en coma flotante.

Una **excepción externa**, también denominada **interrupción**, es un suceso inesperado producido fuera del procesador, y motivado por la demanda de un dispositivo de E/S que pretende comunicarse con el mismo.

Dependiendo de la causa que la produzca, cada excepción requiere un tratamiento individual, realizado mediante la ejecución de una **rutina de tratamiento de excepción (RTE)**. Las RTE forman parte del sistema operativo.

A veces es preciso abortar la ejecución del programa que provoca la excepción, dado que la misma se produce a causa de un error no recuperable. En otras ocasiones, la rutina de tratamiento de la excepción finaliza devolviendo el control al programa que la generó, para que éste pueda continuar su ejecución con normalidad.

## MODOS DE FUNCIONAMIENTO EN MIPS

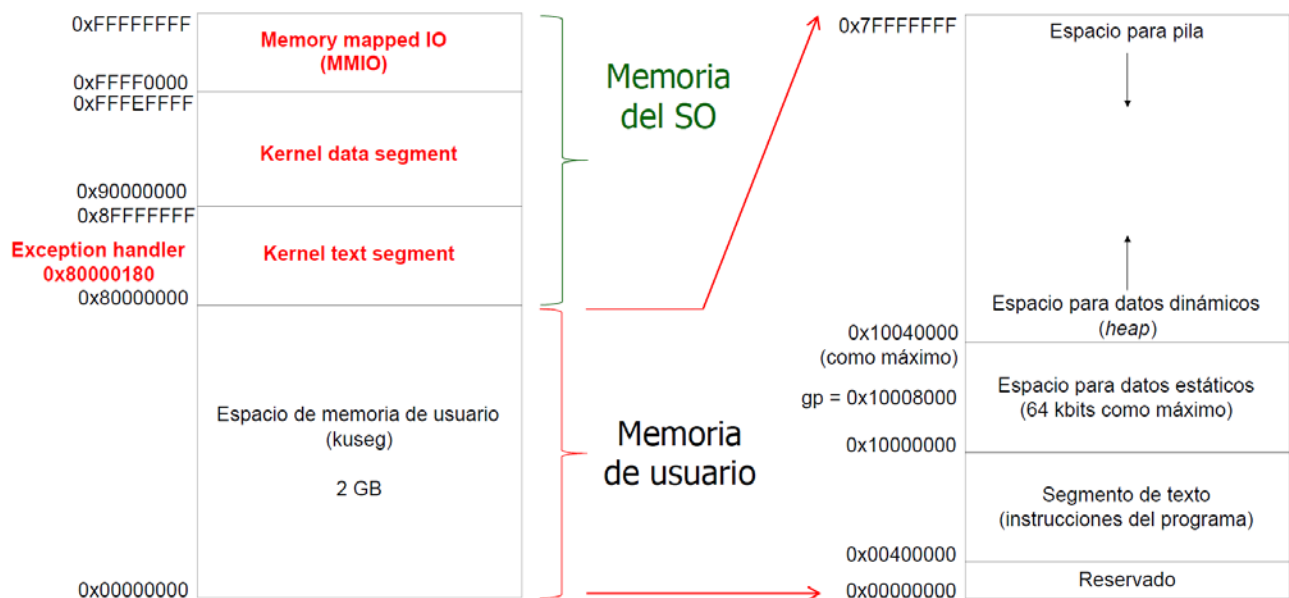
El sistema operativo es un programa que controla el computador, repartiendo los recursos del *hardware* entre los procesos (programas en ejecución) existentes en el sistema, y facilitando que dichos procesos puedan realizar las tareas para las que fueron diseñados.

El *hardware* debe contar con una serie de elementos que den soporte al sistema operativo y le faciliten la realización de su función. Estos elementos son de uso privilegiado por parte del sistema operativo, y no están disponibles para los procesos normales de los usuarios. Para hacer esto posible, el procesador debe disponer de dos modos de funcionamiento:

- **Modo usuario:** es el modo en el que funcionan los programas normales. Este es un modo restringido, en el que no se pueden utilizar los recursos privilegiados de la máquina.
- **Modo núcleo (*kernel*):** es el modo propio del sistema operativo. Cuando el procesador se encuentra en este modo, todos los recursos de la máquina son accesibles. Cuando se produce una excepción, el procesador salta automáticamente al modo núcleo.

## MAPA DE MEMORIA DE MIPS

La siguiente figura muestra el mapa de memoria de MIPS32.



El espacio de memoria por debajo de los 2 GB pertenece a los programas de usuario, y el espacio que se encuentra por encima es de uso privativo por parte del sistema operativo. Una parte de este espacio está reservada para el código del sistema operativo (*kernel text segment*). Dentro de este espacio se encuentra el **manejador de excepciones**, cuyo punto de entrada se encuentra en la dirección 0x80000180. El manejador de excepciones se encarga de identificar la excepción producida, y de invocar a la RTE

correspondiente. También hay un espacio reservado para los datos (*kernel data segment*), y otro para los controladores de periféricos, ubicado en las direcciones superiores a 0xFFFF0000 (recuérdese que MIPS tiene espacio único de direcciones para memoria y E/S).

## EL COPROCESADOR 0 EN LA ARQUITECTURA MIPS

Los **coprocesadores** en MIPS son partes del procesador destinadas a cumplir funciones específicas, y que cuentan con registros propios. Puede haber hasta 4 coprocesadores, con un máximo de 32 registros cada uno.

El **coprocesador 0** (*System Control Coprocessor*) es obligatorio. Entre otras funciones, este coprocesador controla el subsistema de memoria caché, soporta la memoria virtual y la traducción de direcciones virtuales a físicas, el manejo de excepciones, los cambios en el modo de ejecución y proporciona control de diagnóstico y recuperación ante errores de sistema.

## TRATAMIENTO DE EXCEPCIONES

Cuando sucede una excepción, el *hardware* realiza automáticamente la siguiente secuencia de acciones:

- Modificar ciertos bits en el registro de estado (que está en el coprocesador 0) para indicar que se ha producido una excepción.
- Escribir varios bits en el registro de causa (también en el coprocesador 0) para identificar la causa de la excepción.
- Guardar la dirección de la instrucción culpable (*offending*) de la excepción.
- Transferir el control al sistema operativo, que comenzará a ejecutar el manejador de excepciones.

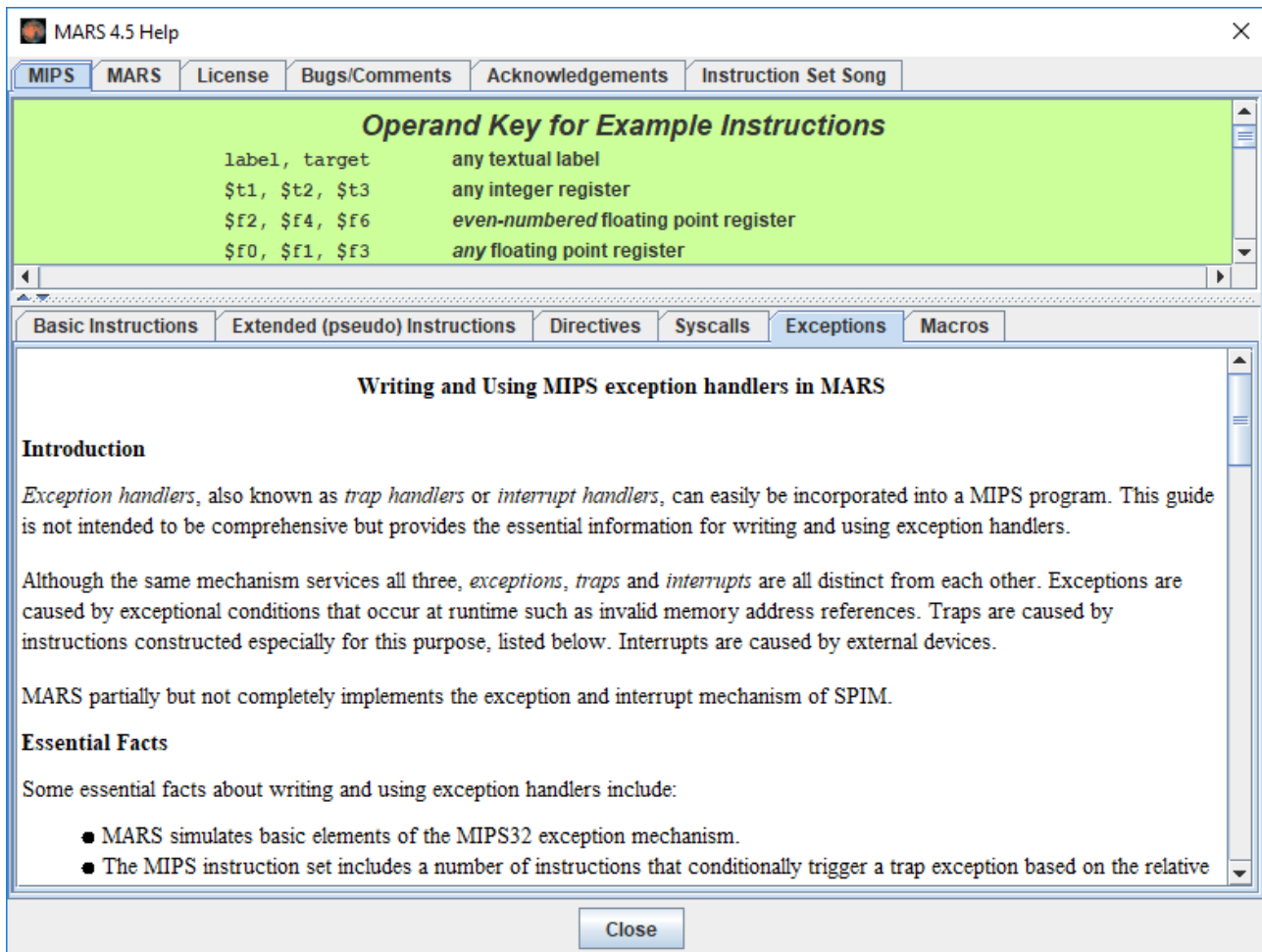
Para realizar estas acciones, los procesadores **MIPS** incorporan ciertos elementos *hardware*, entre los cuales se cuentan varios registros englobados en el coprocesador 0. Vamos a destacar tres:

- El registro de estado (**Status**): contiene varias informaciones, entre las que se cuenta el bit de modo de ejecución, la máscara de interrupciones, etc.
- El registro de causa (**Cause**): entre otras informaciones, se usa para almacenar un código que identifica qué excepción ha sucedido.
- El registro de PC de excepción (**EPC**): este registro contiene la dirección de la instrucción que ha causado la excepción.
- El registro de dirección errónea (VAddr), que contiene la dirección de la instrucción causante de una excepción cuando ésta se produce por un error en un acceso a memoria (en lectura de instrucción o en lectura o escritura de dato).

En el apartado siguiente se presentará la estructura de un manejador de excepciones estándar para **MARS**.

## SOPORTE PARA EXCEPCIONES EN EL SIMULADOR MARS

El simulador **MARS** implementa un tratamiento de excepciones limitado, pero suficiente para desarrollar de esta práctica. Toda la información referente al tratamiento de excepciones en **MARS** se incluye en la ayuda del programa, pero aquí haremos un breve resumen de lo más importante.



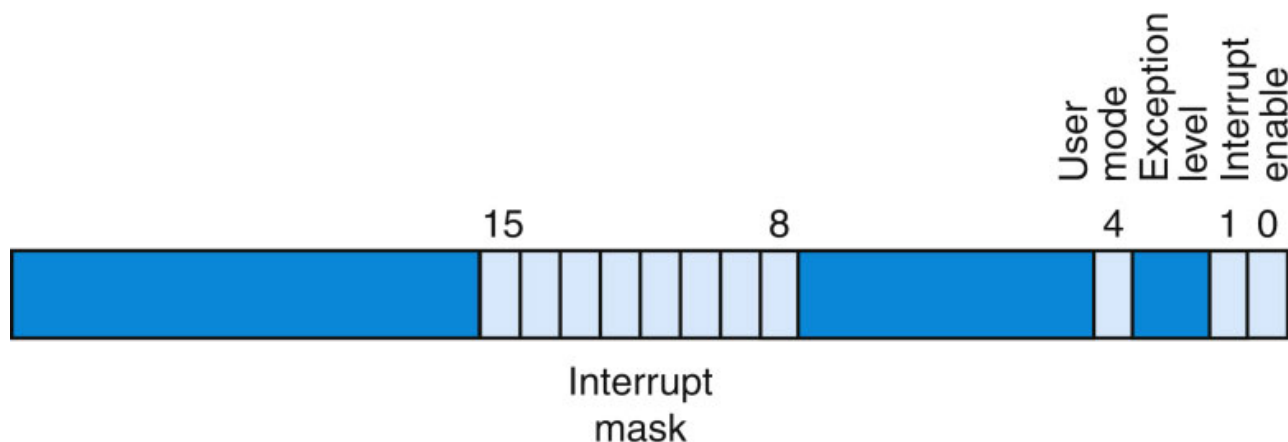
Para el tratamiento de excepciones e interrupciones, **MIPS** dispone de una serie de registros ubicados en el coprocesador 0. Se muestran aquí sólo los que están simulados en **MARS**:

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$8 (vaddr)	8	0x00000000
\$12 (status)	12	0x0000ff11
\$13 (cause)	13	0x00000000
\$14 (epc)	14	0x00000000

El registro **\$8 (BadVAddr)** contiene la dirección de la última instrucción que generó un error de dirección. Este registro indica la dirección errónea

en varias excepciones, incluso si el computador no cuenta con una unidad de manejo de memoria (*memory management unit*, **MMU**).

Por su parte, el registro **\$12 (Status)** contiene el estado de la máquina. No todos sus bits están simulados en **MARS**:



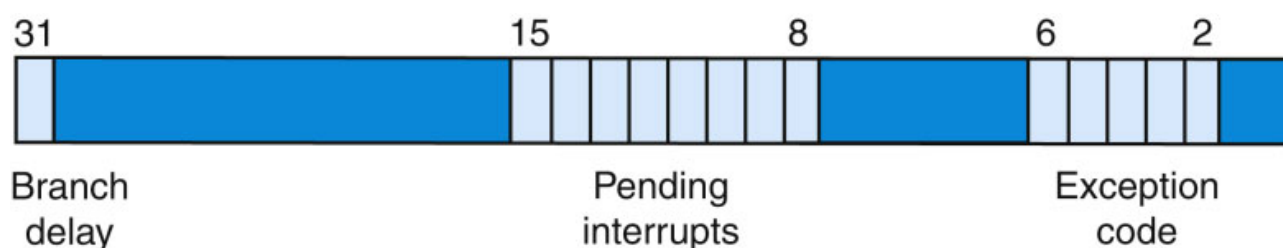
El bit 0 (*interrupt enable*, **IE**) sirve para habilitar o deshabilitar las interrupciones procedentes del *hardware*. Si **IE**=1, las interrupciones están activadas, y si **IE**=0, están desactivadas.

El bit 1 (*exception level*, **EXL**) indica si se ha producido o no una excepción. Si **EXL**=1, entonces se ha producido una excepción que aún no ha sido tratada, o que aún se está tratando. Cuando **EXL** vale 1, las interrupciones están desactivadas. La instrucción de retorno del manejador de excepción (**eret**) pone **EXL** a 0, lo cual indica que ha finalizado el tratamiento de la excepción.

El bit 4 (*user mode*, **UM**) indica el modo de funcionamiento del procesador. Si **UM**=1, el procesador está en modo usuario, mientras que si **UM**=0 entonces se encuentra en modo *kernel*. **MARS** no simula estos dos modos de funcionamiento, así que este bit siempre está a 1.

Los bits 8 a 15 conforman la máscara de interrupciones (*interrupt mask*, **IM**). **MIPS32** tiene 8 niveles de interrupción, 6 de los cuales corresponden con interrupciones procedentes del *hardware*, mientras que los dos restantes se utilizan para generar interrupciones *software*. Si **IE**=1 y un cierto bit de la máscara vale 1, las interrupciones del nivel correspondiente están habilitadas, y si vale a 0 están inhabilitadas.

El registro **\$13 (Cause)** indica la causa que provocó la última excepción o interrupción. No todos sus bits están simulados en **MARS**:



Los bits 2 a 6 contienen el código de excepción (*exception code*, **ExcCode**), que indica la causa que generó la excepción producida. Este código se escribe automáticamente en dichos bits cuando se produce la excepción.

De todas las excepciones posibles, sólo las siguientes están declaradas en **MARS** (y no todas ellas están realmente soportadas):

ExcCode	Nombre	Comentarios
0	INTL	Interrupción externa
4	AdEL	Error de dirección en instrucción de carga
5	AdES	Error de dirección en instrucción de almacenamiento
8	SYSCALL	Llamada a sistema
9	BKPT	Punto de ruptura
10	RI	Instrucción reservada
12	OV	Desbordamiento aritmético
13	TRAP	Trap
15	FPE	Excepción de coma flotante / división por cero
16	FPOV	Excepción de coma flotante / desbordamiento
17	FPUN	Excepción de coma flotante / subdesbordamiento

El bit 31 (*branch delay*, **BD**) se activa cuando la excepción se ha producido en una instrucción ubicada en un hueco de retardo de salto. No nos preocuparemos de este bit en la presente práctica.

Los bits 8 a 15 (*pending interrupts*) se corresponden con los niveles de interrupción del mismo modo en que se asocian a dichos niveles los bits de máscara del registro **Status**. Cuando se ha producido una interrupción, el bit de interrupción pendiente del registro **Cause** que corresponde con su nivel de interrupción se pone a 1. Si la interrupción está enmascarada en el registro **Status**, el bit de interrupción pendiente también se pone a 1, quedando así la interrupción a la espera (pendiente) de ser atendida con posterioridad.

Se interrumpirá al procesador si se cumplen todas las condiciones citadas a continuación:

- Hay un bit de interrupción pendiente activo (con valor 1) en el registro **Cause**.



- El bit de máscara correspondiente en el registro **Status** también está activo (también vale 1).
- El procesador no se encuentra en el nivel de excepción, es decir, **EXL=0** en el registro **Status**.
- Las interrupciones están habilitadas, es decir, **IE=1** en el registro **Status**.

La prioridad de las interrupciones la determina el manejador de interrupción. Así, si hay varias interrupciones pendientes, el manejador decidirá cuál de ellas es más prioritaria y será la próxima en ser atendida.

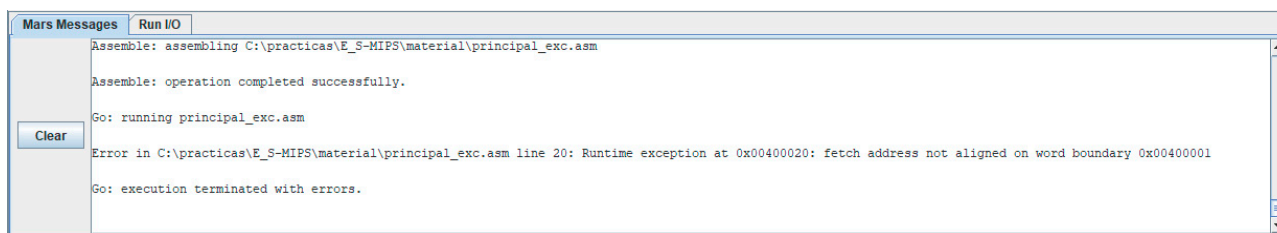
El registro **\$14 (EPC: *exception program counter*)** contiene la dirección de la instrucción que se estaba ejecutando cuando se produjo la excepción. Si la excepción es una interrupción externa, el **EPC** contiene la dirección de la instrucción siguiente, que es por donde debe reanudarse la ejecución del programa en curso tras ejecutar la rutina de tratamiento.

## Tratamiento de excepciones en MARS

Cuando se produce una excepción interna o externa (interrupción) durante una simulación, en **MARS** se realizan automáticamente las siguientes acciones:

- El valor del registro **PC** se copia automáticamente en el registro **EPC**.
- Se escribe en el registro **Cause** el código de la excepción (**ExcCode**). Si se ha producido una interrupción, **ExcCode=0**, y además se activa el bit de interrupción pendiente.
- Se activa el bit de nivel de excepción (**EXL**) en el registro **Status** para indicar que hay una excepción que está pendiente de ser tratada o que aún se está tratando. Esto produce que se deshabiliten las interrupciones hasta que dicho bit se desactive (normalmente al retornar del manejador de excepción con **eret**).
- El procesador debería entrar en modo *kernel*, lo que significa que el manejador de excepciones tendría acceso a todo el mapa de memoria y a todos los registros. En realidad, **MARS** no diferencia entre modos de ejecución, pero en un procesador real este es un hecho importante que es preciso resaltar.
- El contador de programa pasa a valer **0x800000180**, por lo que comenzará la ejecución de la primera instrucción que haya a partir de esa dirección de memoria, que es la primera del manejador de excepciones.

Si no hay ningún manejador de excepciones cargado, **MARS** presenta por consola un mensaje de error estandarizado indicando el tipo de excepción producida, y la ejecución del programa terminará.



Si cuando se produce una excepción no se presenta por consola el mensaje estándar de **MARS** para la excepción producida, es porque hay un manejador de excepción cargado en memoria. Entonces, la ejecución de instrucciones continuará por el punto de entrada del manejador. Cuando la excepción no se deba a un error fatal ante el cual se deba abortar el programa en curso, la ejecución del manejador terminará con la instrucción **eret**. Esta instrucción devuelve a **PC** el valor que se almacenó en el **EPC**, por lo que se volvería al flujo de ejecución de instrucciones del programa original.

## MANEJADOR DE EXCEPCIONES NO REENTRANTE PARA MIPS

El manejador de excepciones puede ser reentrante o no. Si es reentrante, permitirá que el tratamiento de las excepciones y las interrupciones se vayan anidando según su prioridad. Si es no reentrante, no se debe permitir el anidamiento de excepciones y/o interrupciones.

La estructura de un manejador de excepciones no reentrante es la siguiente:

```

# Manejador de excepciones
.ktext 0x80000180

# Guardar estado
# Salvar registros que vaya a utilizar la rutina
# Comprobar origen de la excepción
# Tratar la excepción en cada caso
# Restaurar registros
# Restaurar estado

eret

```

Lo primero es guardar el contenido de los registros que queramos conservar. No utilizaremos la pila para ello, sino que, como el manejador no es reentrante, usaremos para ello variables residentes en memoria. Es preciso tener en cuenta que los registros **\$k0** y **\$k1** pueden ser utilizados libremente dentro del manejador, ya que por convenio están reservados para el *kernel*.

Lo siguiente es comprobar el origen de la excepción. Para ello se consultará el registro **Cause**, aislando el código de la excepción (**ExcCode**) mediante una máscara de bits.

Una vez identificada la excepción, se realizará su tratamiento, que será diferente dependiendo de la excepción acaecida. Si es una interrupción (**ExcCode=0**), será preciso averiguar su nivel para poder atenderla.

Después, devolveremos su valor original a los registros que hayamos modificado. En ciertas excepciones, será preciso incrementar el **EPC** en 4 para no volver a ejecutar la instrucción que la causó.

Lo último es ejecutar la instrucción **eret**, que escribe en el **PC** el valor del **EPC**, y además pone a 0 el bit **EXL** del registro **Status** para abandonar el nivel de excepción.

## ALGUNAS INSTRUCCIONES MIPS32 RELACIONADAS CON LAS EXCEPCIONES

Seguidamente se describen algunas instrucciones de la arquitectura MIPS32 que tienen que ver con la generación o el tratamiento de excepciones.

### Instrucciones de acceso a registros del coprocesador 0

#### **mfc0** Lectura de un registro del coprocesador 0

Esta instrucción copia un registro del coprocesador 0 en un registro de propósito general. El registro de propósito general es el primer operando, y el registro del coprocesador 0 es el segundo.

Uso: **mfc0** *rt, rd*       $rt \leftarrow rd$

Formato:

cod_op: 010000	0	rt	rd	0
31-26	25-21	20-16	15-11	10-0

#### **mtc0** Escritura en un registro del coprocesador 0

Esta instrucción copia un registro de propósito general en un registro del coprocesador 0. El registro de propósito general es el primer operando, y el registro del coprocesador 0 es el segundo.

Uso: **mtc0** *rt, rd*       $rd \leftarrow rt$

Formato:

cod_op: 010000	00100	rt	rd	0
31-26	25-21	20-16	15-11	10-06

### Instrucción de retorno de una excepción

#### **eret** Retorno de excepción

Esta instrucción restaura el **PC** con el valor contenido en el registro \$14 del coprocesador 0, que contiene el **PC** de retorno de excepción o **EPC**. Además, pone a 0 el bit **EXL** del registro \$12 del coprocesador 0 (registro **Status**), deshabilitando el modo de excepción.

Uso: **eret**       $PC \leftarrow EPC$

Formato:

cod_op: 010000	1	0	011000
31-26	25	24-6	5-0

## Instrucciones de generación de traps

### teq Trap si iguales

Es una instrucción de tipo R (con código de operación 000000) que tiene dos registros como operandos. Si sus contenidos son iguales, genera una excepción por trap.

Uso: **teq rs,rt** Si  $rs = rt$ , generar trap

Formato:

cod_op: 000000	rs	rt	0	funct: 110100
31-26	25-21	20-16	15-6	5-0

### teqi Trap si iguales con inmediato

Esta instrucción tiene como operandos un registro y un dato inmediato. Si sus contenidos son iguales, genera una excepción por trap.

Uso: **teqi rs,inm** Si  $rs = inm$ , generar trap

Formato:

cod_op: 000001	rs	01100	inmediato
31-26	25-21	20-16	15-0

### tne Trap si no iguales

Es una instrucción de tipo R (con código de operación 000000) que tiene dos registros como operandos. Si sus contenidos son distintos, genera una excepción por trap.

Uso: **tne rs,rt** Si  $rs \neq rt$ , generar trap

Formato:

cod_op: 000000	rs	rt	0	funct: 110110
31-26	25-21	20-16	15-6	5-0

### tnei Trap si no iguales con inmediato

Esta instrucción tiene como operandos un registro y un dato inmediato. Si sus contenidos son distintos, genera una excepción por trap.

Uso: **tnei rs,inm** Si  $rs \neq inm$ , generar trap

Formato:

cod_op: 000001	rs	01110	inmediato
31-26	25-21	20-16	15-0

### tge Trap si mayor o igual

Es una instrucción de tipo R (con código de operación 000000) que tiene dos registros como operandos. Si el contenido del primer operando es mayor o igual que el del segundo, genera una excepción por trap.

Uso: **tge rs,rt** Si  $rs \geq rt$ , generar trap

Formato:

cod_op: 000000	rs	rt	0	funct: 110000
31-26	25-21	20-16	15-6	5-0

### tgeu Trap si mayor o igual sin signo

Es una instrucción de tipo R (con código de operación 000000) que tiene

dos registros como operandos, ambos conteniendo enteros sin signo. Si el contenido del primer operando es mayor o igual que el del segundo, genera una excepción por trap.

Uso: **tgeu rs,rt** Si  $rs \geq rt$ , generar trap

Formato:

cod_op: 000000	rs	rt	0	funct: 110001
31-26	25-21	20-16	15-6	5-0

### **tgei** Trap si mayor o igual con inmediato

Esta instrucción tiene como operandos un registro y un dato inmediato. Si el contenido del registro es mayor o igual que el del inmediato, genera una excepción por trap.

Uso: **tgei rs,inm** Si  $rs \geq inm$ , generar trap

Formato:

cod_op: 000001	rs	01000	inmediato
31-26	25-21	20-16	15-0

### **tgeiu** Trap si mayor o igual sin signo con inmediato

Esta instrucción tiene como operandos un registro y un dato inmediato, ambos conteniendo enteros sin signo. Si el contenido del registro es mayor o igual que el del inmediato, genera una excepción por trap.

Uso: **tgeiu rs,inm** Si  $rs \geq inm$ , generar trap

Formato:

cod_op: 000001	rs	01001	inmediato
31-26	25-21	20-16	15-0

### **tlr** Trap si menor

Es una instrucción de tipo R (con código de operación 000000) que tiene dos registros como operandos. Si el contenido del primer operando es menor que el del segundo, genera una excepción por trap.

Uso: **tlr rs,rt** Si  $rs < rt$ , generar trap

Formato:

cod_op: 000000	rs	rt	0	funct: 110010
31-26	25-21	20-16	15-6	5-0

### **tlru** Trap si menor sin signo

Es una instrucción de tipo R (con código de operación 000000) que tiene dos registros como operandos, ambos conteniendo enteros sin signo. Si el contenido del primer operando es menor que el del segundo, genera una excepción por trap.

Uso: **tlru rs,rt** Si  $rs < rt$ , generar trap

Formato:

cod_op: 000000	rs	rt	0	funct: 110011
31-26	25-21	20-16	15-6	5-0

### **tlri** Trap si menor con inmediato

Esta instrucción tiene como operandos un registro y un dato inmediato. Si el contenido del registro es menor que el del inmediato, genera una

excepción por trap.

Uso: **tlti rs,inm** Si  $rs < inm$ , generar trap

Formato:

cod_op: 000001	Rs	01000	inmediato
31-26	25-21	20-16	15-0

**tltiu** Trap si menor sin signo con inmediato

Esta instrucción tiene como operandos un registro y un dato inmediato, ambos conteniendo enteros sin signo. Si el contenido del registro es menor que el del inmediato, genera una excepción por trap.

Uso: **tltiu rs,inm** Si  $rs < inm$ , generar trap

Formato:

cod_op: 000001	rs	01001	inmediato
31-26	25-21	20-16	15-0

## SERVICIOS DEL SISTEMA OPERATIVO

Los sistemas operativos proporcionan a los programas de usuario una serie de servicios relacionados con dispositivos de entrada/salida, operaciones privilegiadas del sistema, etc. Estos servicios están numerados y tabulados, y desde programas en bajo nivel es posible invocarlos a través de una instrucción específica. En **MIPS** esa instrucción es **syscall**.

El simulador **MARS** proporciona múltiples llamadas a sistema que implementan diversos servicios de entrada/salida. Estos servicios están implementados directamente en alto nivel en el lenguaje de programación del simulador (**Java**). Para invocar un servicio, será preciso realizar las siguientes acciones:

- Cargar en **\$v0** el número correspondiente al servicio invocado.
- Cargar en los registros **\$a0**, **\$a1**, **\$a2** y/o **\$a3** los parámetros del servicio, según corresponda.
- Ejecutar la instrucción **syscall**.

La ayuda de **MARS** contiene una tabla, mostrada más abajo, con todas las llamadas a sistema de que dispone. En la primera columna aparece el número de servicio; en la segunda, el código del servicio, que debe ser escrito en el registro **\$v0** por el programa solicitante antes de ejecutar **syscall**; en la tercera columna se indican los registros utilizados como argumentos, cuyos valores deben haber sido escritos en ellos también por el programa solicitante antes de la llamada; y en la cuarta se indica cuál es el valor de retorno y dónde puede recogerse.

En realidad, la instrucción **syscall** debería generar una excepción, lo que dispararía la ejecución del manejador de excepciones, que terminaría invocando una rutina de tratamiento que ejecutaría el servicio solicitado. Sin embargo, al invocar uno de los servicios incluidos en la tabla de

**MARS**, el simulador captura la llamada y ejecuta una rutina escrita en Java que implementa dicho servicio sin generar ninguna excepción. Pero si se invoca un servicio con un número que no esté en la lista, dicha excepción sí se producirá en **MARS**, y podrá ser tratada mediante la correspondiente rutina de servicio, que tendrá que estar programada en ensamblador.

A continuación se incluye la tabla de servicios implementados en **MARS**.

Service	Code in \$v0	Arguments	Result
<b>print integer</b>	1	\$a0 = integer to print	
<b>print float</b>	2	\$f12 = float to print	
<b>print double</b>	3	\$f12 = double to print	
<b>print string</b>	4	\$a0 = address of null-terminated string to print	
<b>read integer</b>	5		\$v0 contains integer read
<b>read float</b>	6		\$f0 contains float read
<b>read double</b>	7		\$f0 contains double read
<b>read string</b>	8	\$a0 = address of input buffer \$a1 = maximum number of characters to read	<i>See note below table</i>
<b>sbrk (allocate heap memory)</b>	9	\$a0 = number of bytes to allocate	\$v0 contains address of allocated memory
<b>exit (terminate execution)</b>	10		
<b>print character</b>	11	\$a0 = character to print	<i>See note below table</i>
<b>read character</b>	12		\$v0 contains character read
<b>open file</b>	13	\$a0 = address of null-terminated string containing filename \$a1 = flags \$a2 = mode	\$v0 contains file descriptor (negative if error). <i>See note below table</i>
<b>read from file</b>	14	\$a0 = file descriptor \$a1 = address of input buffer \$a2 = maximum number of characters to read	\$v0 contains number of characters read (0 if end-of-file, negative if error). <i>See note below table</i>
<b>write to file</b>	15	\$a0 = file descriptor \$a1 = address of output buffer \$a2 = number of characters to write	\$v0 contains number of characters written (negative if error). <i>See note below table</i>
<b>close file</b>	16	\$a0 = file descriptor	
<b>exit2 (terminate with value)</b>	17	\$a0 = termination result	<i>See note below table</i>
<b>time (system time)</b>	30		\$a0 = low order 32 bits of system time \$a1 = high order 32 bits of system time. <i>See note below table</i>
<b>MIDI out</b>	31	\$a0 = pitch (0-127) \$a1 = duration in milliseconds \$a2 = instrument (0-127) \$a3 = volume (0-127)	Generate tone and return immediately. <i>See note below table</i>
<b>sleep</b>	32	\$a0 = the length of time to sleep in milliseconds.	Causes the MARS Java thread to sleep for (at least) the specified number of milliseconds. This timing will not be precise, as the Java implementation will add some overhead.
<b>MIDI out synchronous</b>	33	\$a0 = pitch (0-127) \$a1 = duration in milliseconds	Generate tone and return upon tone completion. <i>See note below table</i>

		\$a2 = instrument (0-127) \$a3 = volume (0-127)	
<b>print integer in hexadecimal</b>	34	\$a0 = integer to print	Displayed value is 8 hexadecimal digits, left-padding with zeroes if necessary.
<b>print integer in binary</b>	35	\$a0 = integer to print	Displayed value is 32 bits, left-padding with zeroes if necessary.
<b>print integer as unsigned</b>	36	\$a0 = integer to print	Displayed as unsigned decimal value.
<b>(not used)</b>	37-39		
<b>set seed</b>	40	\$a0 = i.d. of pseudorandom number generator (any int). \$a1 = seed for corresponding pseudorandom number generator.	No values are returned. Sets the seed of the corresponding underlying Java pseudorandom number generator ( <code>java.util.Random</code> ). <i>See note below table</i>
<b>random int</b>	41	\$a0 = i.d. of pseudorandom number generator (any int).	\$a0 contains the next pseudorandom, uniformly distributed int value from this random number generator's sequence. <i>See note below table</i>
<b>random int range</b>	42	\$a0 = i.d. of pseudorandom number generator (any int). \$a1 = upper bound of range of returned values.	\$a0 contains pseudorandom, uniformly distributed int value in the range $0 = [\text{int}] [\text{upper bound}]$ , drawn from this random number generator's sequence. <i>See note below table</i>
<b>random float</b>	43	\$a0 = i.d. of pseudorandom number generator (any int).	\$f0 contains the next pseudorandom, uniformly distributed float value in the range $0.0 = f 1.0$ from this random number generator's sequence. <i>See note below table</i>
<b>random double</b>	44	\$a0 = i.d. of pseudorandom number generator (any int).	\$f0 contains the next pseudorandom, uniformly distributed double value in the range $0.0 = f 1.0$ from this random number generator's sequence. <i>See note below table</i>
<b>(not used)</b>	45-49		
<b>ConfirmDialog</b>	50	\$a0 = address of null-terminated string that is the message to user	\$a0 contains value of user-chosen option 0: Yes 1: No 2: Cancel
<b>InputDialogInt</b>	51	\$a0 = address of null-terminated string that is the message to user	\$a0 contains int read \$a1 contains status value 0: OK status -1: input data cannot be correctly parsed -2: Cancel was chosen -3: OK was chosen but no data had been input into field
<b>InputDialogFloat</b>	52	\$a0 = address of null-terminated string that is the message to user	\$f0 contains float read \$a1 contains status value 0: OK status -1: input data cannot be correctly parsed -2: Cancel was chosen -3: OK was chosen but no data had been input into field
<b>InputDialogDouble</b>	53	\$a0 = address of null-terminated string that is the message to user	\$f0 contains double read \$a1 contains status value 0: OK status -1: input data cannot be correctly parsed -2: Cancel was chosen -3: OK was chosen but no data had been input into field
<b>InputDialogString</b>	54	\$a0 = address of null-terminated string that is the message to user \$a1 = address of input	<i>See Service 8 note below table</i> \$a1 contains status value 0: OK status. Buffer contains the input string. -2: Cancel was chosen. No change to buffer.



		buffer \$a2 = maximum number of characters to read	-3: OK was chosen but no data had been input into field. No change to buffer. -4: length of the input string exceeded the specified maximum. Buffer contains the maximum allowable input string plus a terminating null.
<b>MessageDialog</b>	55	\$a0 = address of null-terminated string that is the message to user \$a1 = the type of message to be displayed: 0: error message, indicated by Error icon 1: information message, indicated by Information icon 2: warning message, indicated by Warning icon 3: question message, indicated by Question icon other: plain message (no icon displayed)	N/A
<b>MessageDialogInt</b>	56	\$a0 = address of null-terminated string that is an information-type message to user \$a1 = int value to display in string form after the first string	N/A
<b>MessageDialogFloat</b>	57	\$a0 = address of null-terminated string that is an information-type message to user \$f12 = float value to display in string form after the first string	N/A
<b>MessageDialogDouble</b>	58	\$a0 = address of null-terminated string that is an information-type message to user \$f12 = double value to display in string form after the first string	N/A
<b>MessageDialogString</b>	59	\$a0 = address of null-terminated string that is an information-type message to user \$a1 = address of null-terminated string to display after the first string	N/A

**NOTES: Services numbered 30 and higher are not provided by SPIM**

**Service 8** - Follows semantics of UNIX 'fgets'. For specified length n, string can be no longer than n-1. If less than that, adds newline to end. In either case, then pads with null byte. If n = 1, input is ignored and null byte placed at buffer address. If n < 1, input is ignored and nothing is written to the buffer.

**Service 11** - Prints ASCII character corresponding to contents of low-order byte.

**Service 13** - MARS implements three flag values: 0 for read-only, 1 for write-only with create, and 9 for write-only with create and append. It ignores mode. The returned file descriptor will be negative if the operation failed. The underlying file I/O implementation uses

`java.io.FileInputStream.read()` to read and `java.io.FileOutputStream.write()` to write. MARS maintains file descriptors internally and allocates them starting with 3. File descriptors 0, 1 and 2 are always open for: reading from standard input, writing to standard output, and writing to standard error, respectively (new in release 4.3).

**Services 13,14,15** - In MARS 3.7, the result register was changed to `$v0` for SPIM compatibility. It was previously `$a0` as erroneously printed in Appendix B of *Computer Organization and Design*,

**Service 17** - If the MIPS program is run under control of the MARS graphical interface (GUI), the exit code in `$a0` is ignored.

**Service 30** - System time comes from `java.util.Date.getTime()` as milliseconds since 1 January 1970.

**Services 31,33** - Simulate MIDI output through sound card. Details below.

**Services 40-44** use underlying Java pseudorandom number generators provided by the `java.util.Random` class. Each stream (identified by `$a0` contents) is modeled by a different `Random` object. There are no default seed values, so use the Set Seed service (40) if replicated random sequences are desired.

Seguidamente se muestran ejemplos de uso de algunos de los servicios anteriores.

## Escritura de una cadena de caracteres

La cadena de caracteres podría incluir el carácter de fin de línea.

```

cadena:      .data
             .asciiz    "Hol a"

             .text
li           $v0, 4
la          $a0, cadena
syscall

```

## Lectura de una cadena de caracteres

```

cadena:      .data
             .eqv       LONGITUD, 32
             .space     LONGITUD

             .text
li           $v0, 8
la          $a0, cadena
li          $a1, LONGITUD
syscall

```

## Escritura de un entero

Supondremos que el entero se encuentra en una variable llamada **num**.

```

num:         .data
             .space     4
             .text
li           $v0, 1
lw          $a0, num
syscall

```

## Lectura de un entero

Supondremos que el entero se guardará en una variable llamada **num**.

```

num:         .data
             .space     4
             .text
li           $v0, 5

```

```

    syscall
    sw      $v0, num

```

## Terminación de un programa indicando valor de retorno

Si el programa termina correctamente se devuelve un 0, y si termina de forma incorrecta, se devolverá un número distinto de 0 que indicará un código de error. El código devuelto es útil si **MARS** se está ejecutando desde la línea de mandatos sin estar controlado por el interfaz gráfico.

```

    .text
    li      $v0, 17
    li      $a0, 0
    syscall

```

## CUESTIONARIO 2. 1. 1

1. Indicar en cuáles de las siguientes situaciones se produce una excepción:
  - a) Que el resultado de una operación aritmética sea negativo.
  - b)** Que se produzca desbordamiento en una operación aritmética.
  - c) Que el resultado de una operación lógica tenga todos sus bits a 0.
  - d)** Que la dirección especificada como destino en una operación de salto sea errónea.
  - e) Que el código de operación de una instrucción sea incorrecto.
  - f) Que el desplazamiento en una instrucción de ramificación sea negativo.
  - g) Que se ejecute una instrucción syscall.
  - h)** Que una instrucción de carga o almacenamiento acceda a una dirección desalineada.
  - i)** Que se produzca un fallo de TLB.
  - j) Que se pretenda calcular la raíz cuadrada de un número negativo.
2. ¿Cuál es el punto de entrada del **manejador de excepción**?
  - a) 0x00400000.
  - b) 0x80000000.
  - c)** 0x90000000.
  - d) Ninguna de las restantes respuestas es cierta.
3. ¿Cuál es la instrucción que se utiliza para **terminar la ejecución del manejador de excepciones**?
  - a) jr
  - b) jal
  - c) bgez
  - d)** eret
  - e) jalr
  - f) Ninguna de las restantes respuestas es cierta.
4. ¿Cuáles de las siguientes acciones se ejecutan **automáticamente** cuando se produce una excepción?
  - a)** Se ejecuta la instrucción eret.

- b) Se copia en el registro EPC la dirección de la instrucción causante de la excepción.
  - c) Se graba el código de la excepción en el registro Status.
  - d) Se activa el bit EXL de nivel de excepción en el registro Status.
  - e) Se inhabilitan las interrupciones en el registro Cause.
  - f) El procesador se pone en modo *kernel*.
  - g) Se copia la dirección del punto de entrada del manejador de excepciones en el PC.
  - h) Se aborta la ejecución del programa en curso.
5. **Emparejar cada registro con la función que desempeña:**
- 1) VAddr                      a) Indica la causa de la excepción.
  - 2) Status                    b) Contiene la dirección de la instrucción causante de la excepción.
  - 3) Cause                    c) Contiene el estado del procesador.
  - 4) EPC                      d) Contiene la dirección de memoria en la que se ha producido el acceso incorrecto.
6. **Emparejar los campos del registro Status con su utilidad:**
- 1) Bits 8 a 15              a) Habilitación de interrupciones.
  - 2) Bit 0                    b) Nivel de excepción (EXL).
  - 3) Bit 1                    c) Modo de usuario.
  - 4) Bit 4                    d) Máscara de interrupciones.
7. **En MARS, ¿cuál es valor inicial del registro Status antes de comenzar a ejecutar un programa?**
- a) 0x0000ff11.
  - b) 0x0000ff13.
  - c) 0x00000000.
  - d) Ninguna de las restantes respuestas es cierta.
8. **Emparejar los campos del registro Cause con su utilidad:**
- 1) Bits 2 al 6              a) Excepción en hueco de retardo de salto retardado.
  - 2) Bits 8 a 15              b) Código de excepción.
  - 3) Bit 31                    c) Interrupciones pendientes.
9. **En MARS, ¿cuál es valor inicial del registro Cause antes de comenzar a ejecutar un programa?**
- a) 0x0000ff11.
  - b) 0x0000ff13.
  - c) 0x00000000.
  - d) Ninguna de las restantes respuestas es cierta.
10. **De las siguientes excepciones, indicar cuáles de ellas revelan un error fatal en el programa y provocan que aborte.**

- a) Desbordamiento aritmético.
- b) Llamada a sistema.
- c) Instrucción ilegal.
- d) Fallo de TLB.
- e) Interrupción.
- f) Acceso incorrecto a memoria.

11. Indicar cuál de las siguientes afirmaciones es CIERTA:

- a) En las instrucciones mfc0 y mtc0, el primer operando es un registro del coprocesador 0, y el segundo es un registro de propósito general.
- b) En las instrucciones mfc0 y mtc0, el primer operando es un registro de propósito general, y el segundo es un registro del coprocesador 0.
- c) En ambas instrucciones, el registro destino es el primer operando y el registro origen es el segundo.
- d) Ninguna de las restantes respuestas es cierta.

12. (VERDADERO 0 FALSO) ¿La instrucción syscall produce una excepción?

13. (VERDADERO 0 FALSO) Indicar si la siguiente pareja de instrucciones genera una excepción:

```
addi $s0, $zero, 0x7fffffff
addiu $s0, $s0, 10
```

14. (VERDADERO 0 FALSO) ¿La instrucción break produce una excepción?

15. (VERDADERO 0 FALSO) Indicar si la siguiente pareja de instrucciones genera una excepción:

```
addi $s0, $zero, 0x80000000
addi $s0, $s0, -10
```

16. (VERDADERO 0 FALSO) ¿La instrucción eret produce una excepción?

17. (VERDADERO 0 FALSO) Indicar si la siguiente pareja de instrucciones genera una excepción:

```
addi $s0, $zero, 0x80000000
tge $zero, $s0
```

18. (VERDADERO 0 FALSO) Indicar si la instrucción tnei \$0,0 genera una excepción.

19. (VERDADERO 0 FALSO) Indicar si la siguiente pareja de instrucciones genera una excepción:

```
addi $s0, $zero, 0x80000000
tgeu $zero, $s0
```

20. (VERDADERO 0 FALSO) Indicar si la siguiente pareja de instrucciones genera una excepción:

```
addi $s0, $zero, 0x80000000
tlt $zero, $s0
```

## 2. 1. 2. TRATAMIENTO DE EXCEPCIONES POR DEFECTO EN MARS

### INTRODUCCIÓN

En este apartado de la práctica se probarán varios programas que generan distintas excepciones. Los ficheros fuente son **caso01.asm**, **caso02.asm**, **caso03.asm**, **caso04.asm**, **caso05.asm** y **caso06.asm**.

Para realizar el presente apartado no se utilizará ningún manejador de excepciones, sino que en cada caso **MARS** responderá con la acción por defecto.


A lo largo del ejercicio se planteará un cuestionario con una serie de preguntas que se responderán a través del Aula Virtual.







### REALIZACIÓN DEL CUESTIONARIO 2. 1. 2

#### Ejercicio 1

Abrir **MARS**, y verificar que no hay ningún manejador de excepciones instalado. Para ello, entrar en **Exception Handler ...** dentro del menú **Settings**, y desactivar la opción **Include this exception handler file in all assemble operations**.

A continuación, cargar el fichero **caso01.asm** con la opción **File -> Open**, y ensamblarlo con la opción **Run -> Assemble**. Después, ejecutar el programa mediante la opción **Run -> Go**.

1. ¿Cuál es el mensaje que aparece en la zona de mensajes de **MARS** (**Mars Messages**) al ejecutar el código del fichero **caso01.asm**?
  - a) Error in C:\...\sesion1\caso01.asm line 16: Runtime exception at 0x00400020: trap
  - b) Error in C:\...\sesion1\caso01.asm line 15: Runtime exception at 0x00400018: fetch address not aligned on word boundary 0x10010006**
  - c) Error in C:\...\sesion1\caso01.asm line 13: Runtime exception at 0x00400018: store address not aligned on word boundary 0x00400002
  - d) Error in C:\...\sesion1\caso01.asm line 14: Runtime exception at 0x0040001c: arithmetic overflow
  - e) Error in C:\...\sesion1\caso01.asm line 15: Runtime exception at 0x0040001c: invalid or unimplemented syscall service: 100
  - f) Error in C:\...\sesion1\caso01.asm line 12: Runtime exception at 0x00400010: break instruction executed; no code given.
2. ¿Cuál es el contenido del registro del coprocesador 0 **BadVaddr (\$8)** en hexadecimal? (escribirlo con el prefijo **0x**, tal como aparece en **MARS**)

3.  ¿Cuál es el contenido del registro del coprocesador 0 Status (\$12) en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)
4.  ¿Cuál es el contenido del registro del coprocesador 0 Cause (\$13) en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)
5.  ¿Cuál es el contenido del registro del coprocesador 0 EPC (\$14) en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)
6.  ¿Cuál es el código de la causa de la excepción, expresado en base 10?
7. De acuerdo con el mensaje ofrecido por MARS, y teniendo en cuenta el código de causa, ¿de qué tipo es la excepción producida?
  - a) OV: excepción por desbordamiento aritmético.
  - b) TRAP: excepción por *trap*.
  - c) AdEL: excepción por dirección errónea (load o fetch).**
  - d) AdES: excepción por dirección errónea (store).**
  - e) SYSCALL: excepción por llamada a sistema.
  - f) BKPT: excepción por punto de ruptura.
8. Indicar cuál de las siguientes afirmaciones es cierta en relación al contenido del registro Status:
  - a) El modo núcleo está activo.
  - b) El bit de habilitación de interrupciones está desactivado.
  - c) La máscara de interrupciones enmascara las interrupciones de cualquier nivel.
  - d) El modo de nivel de excepción está activado.**
  - e) Ninguna de las restantes respuestas es cierta.
9.  ¿Cuál es la dirección de la instrucción causante de la excepción, en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)
10.  ¿Cuál es la dirección de memoria cuyo acceso ha causado la excepción, en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)








Pulsar el botón **Clear** de la zona de mensajes y de la consola, cerrar el fichero **caso01.asm** mediante la opción **File -> Close**, y continuar con el siguiente ejercicio.

## Ejercicio 2

Aún sin tener ningún manejador de excepciones cargado, abrir el fichero **caso02.asm** con la opción **File -> Open**, y ensamblarlo con la opción **Run -> Assemble**. Después, ejecutar el programa mediante la opción **Run -> Go**.

11. ¿Cuál es el mensaje que aparece en la zona de mensajes de MARS (Mars Messages) al ejecutar el código del fichero **caso02.asm**?
  - a) Error in C:\...\sesion1\caso02.asm line 16: Runtime exception at 0x00400020: trap
  - b) Error in C:\...\sesion1\caso02.asm line 15: Runtime exception at 0x00400018: fetch address not aligned on word boundary 0x10010006
  - c) Error in C:\...\sesion1\caso02.asm line 13: Runtime exception at 0x00400018: store address not aligned on word boundary 0x00400002**



- d) Error in C:\...\sesion1\caso02.asm line 14: Runtime exception at 0x0040001c: arithmetic overflow
- e) Error in C:\...\sesion1\caso02.asm line 15: Runtime exception at 0x0040001c: invalid or unimplemented syscall service: 100
- f) Error in C:\...\sesion1\caso02.asm line 12: Runtime exception at 0x00400010: break instruction executed; no code given.
12.  ¿Cuál es el contenido del registro del coprocesador 0 BadVaddr (\$8) en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)
13.  ¿Cuál es el contenido del registro del coprocesador 0 Status (\$12) en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)
14.  ¿Cuál es el contenido del registro del coprocesador 0 Cause (\$13) en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)
15.  ¿Cuál es el contenido del registro del coprocesador 0 EPC (\$14) en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)
16.  ¿Cuál es el código de la causa de la excepción, expresado en base 10?
17. De acuerdo con el mensaje ofrecido por MARS, y teniendo en cuenta el código de causa, ¿de qué tipo es la excepción producida?
- 0V: excepción por desbordamiento aritmético.
  - TRAP: excepción por *trap*.
  - c)** AdEL: excepción por dirección errónea (load o fetch).
  - SYSCALL: excepción por llamada a sistema.
  - e)** AdES: excepción por dirección errónea (store).
  - BKPT: excepción por punto de ruptura.
18. Indicar cuál de las siguientes afirmaciones es cierta en relación al contenido del registro Status:
- El modo núcleo está activo.
  - El bit de habilitación de interrupciones está desactivado.
  - c)** La máscara de interrupciones enmascara las interrupciones de cualquier nivel.
  - d)** El modo de nivel de excepción está activado.
  - Ninguna de las restantes respuestas es cierta.
19.  ¿Cuál es la dirección de la instrucción causante de la excepción, en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)
20.  ¿Cuál es la dirección de memoria cuyo acceso ha causado la excepción, en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)

Pulsar el botón **Clear** de la zona de mensajes y de la consola, cerrar el fichero **caso02.asm** mediante la opción **File -> Close**, y continuar con el siguiente ejercicio.





## Ejercicio 3


Aún sin tener ningún manejador de excepciones cargado, abrir el fichero **caso03.asm** con la opción **File -> Open**, y ensamblarlo con la opción **Run -> Assemble**. Después, ejecutar el programa mediante la opción **Run -> Go**.


21. ¿Cuál es el mensaje que aparece en la zona de mensajes de MARS (Mars Messages) al ejecutar el código del fichero **caso03.asm**?

- a) Error in C:\...\sesion1\caso03.asm line 16: Runtime exception at 0x00400020: trap
- b) Error in C:\...\sesion1\caso03.asm line 15: Runtime exception at 0x00400018: fetch address not aligned on word boundary 0x10010006
- c) Error in C:\...\sesion1\caso03.asm line 13: Runtime exception at 0x00400018: store address not aligned on word boundary 0x00400002
- d) Error in C:\...\sesion1\caso03.asm line 14: Runtime exception at 0x0040001c: arithmetic overflow
- e)** Error in C:\...\sesion1\caso03.asm line 15: Runtime exception at 0x0040001c: invalid or unimplemented syscall service: 100
- f) Error in C:\...\sesion1\caso03.asm line 12: Runtime exception at 0x00400010: break instruction executed; no code given.

22.  ¿Cuál es el contenido del registro del coprocesador 0 BadVaddr (\$8) en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)

23.  ¿Cuál es el contenido del registro del coprocesador 0 Status (\$12) en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)

24.  ¿Cuál es el contenido del registro del coprocesador 0 Cause (\$13) en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)

25.  ¿Cuál es el contenido del registro del coprocesador 0 EPC (\$14) en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)


26.  ¿Cuál es el código de la causa de la excepción, expresado en base 10?

27. De acuerdo con el mensaje ofrecido por MARS, y teniendo en cuenta el código de causa, ¿de qué tipo es la excepción producida?

- a) OV: excepción por desbordamiento aritmético.
- b) TRAP: excepción por *trap*.
- c) AdEL: excepción por dirección errónea (load o fetch).
- d)** SYSCALL: excepción por llamada a sistema.
- e) AdES: excepción por dirección errónea (store).
- f) BKPT: excepción por punto de ruptura.

28. Indicar cuál de las siguientes afirmaciones es cierta en relación al contenido del registro Status:





- a) El modo núcleo está activo.
- b) El bit de habilitación de interrupciones está desactivado.
- c) La máscara de interrupciones enmascara las interrupciones de cualquier nivel.
- d)** El modo de nivel de excepción está activado.
- e) Ninguna de las restantes respuestas es cierta.

29.  ¿Cuál es la dirección de la instrucción causante de la excepción, en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)
30. Indicar cuál de las siguientes afirmaciones es CIERTA:
- a) Con un manejador de excepciones cargado, el tratamiento de una excepción con este código de causa debería producir siempre que el programa abortase.
  - b)** Con un manejador de excepciones cargado, el tratamiento de una excepción con este código de causa debería producir siempre que el programa continuase su ejecución.
  - c)** En este caso concreto, la excepción debería abortar la ejecución del programa en curso.
  - d) Ninguna de las restantes respuestas es cierta.

Pulsar el botón **Clear** de la zona de mensajes y de la consola, cerrar el fichero **caso03.asm** mediante la opción **File -> Close**, y continuar con el siguiente ejercicio.

## Ejercicio 4

Aún sin tener ningún manejador de excepciones cargado, abrir el fichero **caso04.asm** con la opción **File -> Open**, y ensamblarlo con la opción **Run -> Assemble**. Después, ejecutar el programa mediante la opción **Run -> Go**.

31. ¿Cuál es el mensaje que aparece en la zona de mensajes de MARS (Mars Messages) al ejecutar el código del fichero **caso04.asm**?
- a) Error in C:\...\sesion1\caso04.asm line 16: Runtime exception at 0x00400020: trap
  - b) Error in C:\...\sesion1\caso04.asm line 15: Runtime exception at 0x00400018: fetch address not aligned on word boundary 0x10010006
  - c) Error in C:\...\sesion1\caso04.asm line 13: Runtime exception at 0x00400018: store address not aligned on word boundary 0x00400002
  - d) Error in C:\...\sesion1\caso04.asm line 14: Runtime exception at 0x0040001c: arithmetic overflow
  - e) Error in C:\...\sesion1\caso04.asm line 15: Runtime exception at 0x0040001c: invalid or unimplemented syscall service: 100
  - f)** Error in C:\...\sesion1\caso04.asm line 12: Runtime exception at 0x00400010: break instruction executed; no code given.
32.  ¿Cuál es el contenido del registro del coprocesador 0 Status (\$12) en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)
33.  ¿Cuál es la dirección de la instrucción causante de la excepción, en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)
34.  ¿Cuál es el contenido del registro del coprocesador 0 Cause (\$13) en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)
35.  ¿Cuál es el código de la causa de la excepción, expresado en base 10?
36. De acuerdo con el mensaje ofrecido por MARS, y teniendo en cuenta el código de causa, ¿de qué tipo es la excepción producida?

- a) OV: excepción por desbordamiento aritmético.
- b) TRAP: excepción por *trap*.
- c) AdEL: excepción por dirección errónea (load o fetch).
- d) SYSCALL: excepción por llamada a sistema.
- e) AdES: excepción por dirección errónea (store).
- f) BKPT: excepción por punto de ruptura.**


Pulsar el botón **Clear** de la zona de mensajes y de la consola, cerrar el fichero **caso04.asm** mediante la opción **File -> Close**, y continuar con el siguiente ejercicio.


## Ejercicio 5


Aún sin tener ningún manejador de excepciones cargado, abrir el fichero **caso05.asm** con la opción **File -> Open**, y ensamblarlo con la opción **Run -> Assemble**. Después, ejecutar el programa mediante la opción **Run -> Go**.

37. ¿Cuál es el mensaje que aparece en la zona de mensajes de MARS (Mars Messages) al ejecutar el código del fichero **caso03.asm**?

- a) Error in C:\...\sesion1\caso05.asm line 16: Runtime exception at 0x00400020: trap
- b) Error in C:\...\sesion1\caso05.asm line 15: Runtime exception at 0x00400018: fetch address not aligned on word boundary 0x10010006
- c) Error in C:\...\sesion1\caso05.asm line 13: Runtime exception at 0x00400018: store address not aligned on word boundary 0x00400002
- d) Error in C:\...\sesion1\caso05.asm line 14: Runtime exception at 0x0040001c: arithmetic overflow**
- e) Error in C:\...\sesion1\caso05.asm line 15: Runtime exception at 0x0040001c: invalid or unimplemented syscall service: 100
- f) Error in C:\...\sesion1\caso05.asm line 12: Runtime exception at 0x00400010: break instruction executed; no code given.


38.  ¿Cuál es el contenido del registro del coprocesador 0 Status (\$12) en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)

39.  ¿Cuál es el contenido del registro del coprocesador 0 Cause (\$13) en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)

40.  ¿Cuál es el código de la causa de la excepción, expresado en base 10?

41. De acuerdo con el mensaje ofrecido por MARS, y teniendo en cuenta el código de causa, ¿de qué tipo es la excepción producida?

- a) OV: excepción por desbordamiento aritmético.**
- b) TRAP: excepción por *trap*.
- c) AdEL: excepción por dirección errónea (load o fetch).
- d) SYSCALL: excepción por llamada a sistema.
- e) AdES: excepción por dirección errónea (store).
- f) BKPT: excepción por punto de ruptura.

42.  ¿Cuál es la dirección de la instrucción causante de la excepción, en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)


Pulsar el botón **Clear** de la zona de mensajes y de la consola, cerrar el fichero **caso05.asm** mediante la opción **File -> Close**, y continuar con el siguiente ejercicio.


## Ejercicio 6

Aún sin tener ningún manejador de excepciones cargado, abrir el fichero **caso06.asm** con la opción **File -> Open**, y ensamblarlo con la opción **Run -> Assemble**. Después, ejecutar el programa mediante la opción **Run -> Go**.

43. ¿Cuál es el mensaje que aparece en la zona de mensajes de MARS (Mars Messages) al ejecutar el código del fichero **caso06.asm**?

- a) Error in C:\...\sesion1\caso06.asm line 16: Runtime exception at 0x00400020: trap
- b) Error in C:\...\sesion1\caso06.asm line 15: Runtime exception at 0x00400018: fetch address not aligned on word boundary 0x10010006
- c) Error in C:\...\sesion1\caso06.asm line 13: Runtime exception at 0x00400018: store address not aligned on word boundary 0x00400002
- d) Error in C:\...\sesion1\caso06.asm line 14: Runtime exception at 0x0040001c: arithmetic overflow
- e) Error in C:\...\sesion1\caso06.asm line 15: Runtime exception at 0x0040001c: invalid or unimplemented syscall service: 100
- f) Error in C:\...\sesion1\caso06.asm line 12: Runtime exception at 0x00400010: break instruction executed; no code given.

44.  ¿Cuál es el contenido del registro del coprocesador 0 Status (\$12) en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)

45.  ¿Cuál es el contenido del registro del coprocesador 0 Cause (\$13) en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)


46.  ¿Cuál es el código de la causa de la excepción, expresado en base 10?

47. De acuerdo con el mensaje ofrecido por MARS, y teniendo en cuenta el código de causa, ¿de qué tipo es la excepción producida?

- a) 0V: excepción por desbordamiento aritmético.
- b) TRAP: excepción por *trap*.
- c) AdEL: excepción por dirección errónea (load o fetch).
- d) SYSCALL: excepción por llamada a sistema.
- e) AdES: excepción por dirección errónea (store).
- f) BKPT: excepción por punto de ruptura.

48. ¿Por qué se ha producido esta excepción?

- a) Porque se ha ejecutado la instrucción **trap**.
- b) Porque se ha ejecutado una instrucción de generación de **trap**.
- c) Porque se ha cumplido una cierta condición al ejecutar una instrucción de generación de **trap**.
- d) Ninguna de las restantes respuestas es cierta.

49.  ¿Cuál es la dirección de la instrucción causante de la excepción, en hexadecimal? (escribirlo con el prefijo 0x, tal como aparece en MARS)

50. ¿Cuál es la instrucción concreta que ha producido la excepción?

- a) **tnei \$zero, 0**
- b) **tlti \$s0, 0**
- c) **teqi \$s0, 0**
- d) tgei \$s0, 0**
- e) Ninguna de las restantes respuestas es cierta.

Pulsar el botón **Clear** de la zona de mensajes y de la consola, cerrar el fichero **caso06.asm** mediante la opción **File -> Close**, y cerrar **MARS**.

## 2. 1. 3. MANEJADOR DE EXCEPCIONES GENÉRICO

### INTRODUCCIÓN

En este apartado de la práctica se probarán los mismos programas que se utilizaron en el apartado anterior, pero teniendo cargado un manejador de excepción sencillo, que realizará un tratamiento genérico de las excepciones producidas.

Junto con este ejercicio se planteará un cuestionario con una serie de preguntas que se responderán a través del Aula Virtual.

### DESCRIPCIÓN DEL MANEJADOR

La estructura del manejador propuesto es la siguiente:

```
# MANEJADOR DE EXCEPCIONES: VARIABLES Y DEFINICIONES
    .kdata
# Zona de salvaguarda de registros
# Mensajes

# MANEJADOR DE EXCEPCIONES: CÓDIGO
    .ktext      0x80000180
EXCEPTION_HANDLER_ENTRY:
# Salvar registros por si hubiera que retornar del manejador
# Identificar la causa de la excepción
# Tratamiento genérico de las excepciones
    # Se imprimen diversos mensajes
    # Abortar la ejecución del programa que generó la excepción
# Restaurar registros salvados
# Retornar del manejador de excepción
return_exception:
    eret
```

Cuando se produzca una excepción al ejecutar un programa en MARS, saltará automáticamente el manejador de excepciones, que en esta primera versión se limitará a:

- Identificar la causa de la excepción producida.
- Presentar un mensaje en la consola informando de que se ha producido una excepción.
- Imprimir en la consola el código de la excepción y el contenido de los registros **VAddr**, **Status**, **Cause** y **EPC**.
- Terminar la ejecución, abortando el programa en curso.

Es decir, el presente manejador no distinguirá unas excepciones de otras al realizar su tratamiento, presentando en todo caso una serie de mensajes informativos que sirven de ayuda para identificar la excepción producida, y abortando la ejecución del programa que la provocó.

Además, este manejador no es reentrante, es decir, no permite anidamiento en el tratamiento de excepciones.

## ANÁLISIS DEL CÓDIGO FUENTE DEL MANEJADOR

### Salvaguardar registros

Para poder realizar esta tarea, se crea un espacio en memoria utilizando la directiva `.word` una vez por cada registro:

```
# Zona de salvaguarda de registros
        .align 2
sv_at:   .word 0
sv_a0:   .word 0
sv_v0:   .word 0
```

Los registros se salvaguardan nada más entrar en el manejador mediante el siguiente código:

```
# Salvar $at en $k0 (antes de utilizar pseudoinstrucciones)
        move $k0, $at
# Salvar copia de $at, $v0 y $s0 en memoria
        sw    $k0, sv_at
        sw    $v0, sv_v0
        sw    $a0, sv_a0
```

Los registros `$k0` y `$k1` no se salvaguardarán en memoria, dado que son de uso libre por parte del código del *kernel*.

Antes de usar ninguna pseudoinstrucción que pueda modificarlo, se guarda copia del registro `$at` en el registro `$k0`, y desde ahí se copia en el espacio en memoria reservado para ello. Después se salvan los registros `$v0` y `$a0`, utilizados por el manejador. Si tras responder a la excepción producida el manejador devolviera el control al programa que la generó, los tres registros deberían ser restaurados antes de retornar.

Una de los motivos por los que el manejador no es **reentrante** (no puede anidar el tratamiento de excepciones) es que los registros utilizados se salvaguardan en una zona de memoria concreta y no en la pila.

### Identificar la causa de la excepción

Este manejador necesita identificar la causa de la excepción para presentar en la consola los mensajes informativos. Como la causa está en el campo **ExcCode** del registro **Cause**, se utiliza para ello el siguiente código:

```
# Identificar la causa de la excepción
identify_cause:
    # Copiar registro Cause ($13-coprocesador 0) en $k0
        mfc0 $k0, $13
    # Extraer el campo de código de excepción (ExcCode)
        andi $k0, $k0, 0x7c
```

Con este código, en `$k0` está el código de causa seguido de dos bits a 0, ya que el campo **ExcCode** está en los bits 2 a 6 del registro **Cause**, y con la instrucción `andi` se enmascara (se pone a 0) el resto del registro (más bien su copia en `$k0`).

## Mensajes informativos

Lo siguiente es presentar varios mensajes en la consola informando de que se ha producido una excepción, y además presentando el código de dicha excepción y los contenidos de los registros del coprocesador 0 simulados en **MARS** (**VAddr**, **Status**, **Cause** y **EPC**).

Los mensajes se encuentran definidos en la zona de datos del manejador:

```
# Mensajes
__msgExc__:      .asciiz "Ha ocurrido una excepción\n"
__msgExcCode__:  .asciiz "  Código de excepción  : "
__msgVAddr__:    .asciiz "  Registro VAddr   ($8): "
__msgStatus__:   .asciiz "  Registro Status ($12): "
__msgCause__:    .asciiz "  Registro Cause   ($13): "
__msgEPC__:      .asciiz "  Registro EPC     ($14): "
__eoln__:        .byte '\n'
```

La cadena `__msgExc__` se usa para indicar que ha sucedido una excepción, y se imprime delante de los demás mensajes.

El código de la excepción se imprime así:

```
# Imprimir causa de la excepción
# Imprimir código de causa (decimal)
li    $v0, 4          # syscall 4 (print_str)
la    $a0, __msgExcCode__
syscall
li    $v0, 1          # syscall 1 (print_int)
move  $a0, $k0
srl   $a0, $a0, 2
syscall
li    $v0, 11         # syscall 11 (print_char)
lb    $a0, __eoln__
syscall
```

El campo **ExcCode** del registro **Cause** ya estaba copiado en **\$k0**, pero con dos bits a 0 por detrás. Por tanto, para imprimir el código de excepción se copia **\$k0** en **\$a0**, y después se desplaza **\$a0** dos lugares a la derecha.

A continuación se imprime en la consola el contenido de los registros **VAddr**, **Status**, **Cause** y **EPC** en hexadecimal. Como ejemplo se incluye el código necesario para imprimir el contenido de **Status**:

```
li    $v0, 4          # syscall 4 (print_str)
la    $a0, __msgStatus__
syscall
li    $v0, 34         # syscall 34 (print_int_hexadecimal)
mfhc0 $a0, $12
syscall
li    $v0, 11         # syscall 11 (print_char)
lb    $a0, __eoln__
syscall
```

## Abortar la ejecución

La mayoría de las excepciones soportadas por **MARS** corresponden con errores fatales que deberían suponer la finalización de la ejecución del programa



en curso. Como no se va a realizar un tratamiento diferenciado de las excepciones, este manejador no retornará en ningún caso el control al programa que generó la excepción.

Abortar la ejecución de un programa con una excepción fatal es una tarea del sistema operativo que tiene una cierta complejidad. En este manejador sencillo, para que no se retorne al programa en curso simplemente se utiliza una llamada para terminar la ejecución devolviendo un código de error:

```
# Abortar la ejecución del programa que generó la excepción
li    $v0, 4      # syscall 4 (print_str)
la    $a0, __msgAbort__
syscall
li    $v0, 17     # syscall 17 (exit with code)
li    $a0, 1      # Error
syscall
```

Esta operación produce la finalización de ejecución de las instrucciones en **MARS**, con lo cual el resto del manejador nunca se ejecutará. En un caso real, con un sistema operativo real, esto se realizará de otra forma, y el procesador no se parará.

## Restaurar los registros utilizados y retornar

Al abortar el programa en curso, en realidad estamos deteniendo la ejecución de instrucciones en **MARS**. Pero en un caso real será preciso restaurar los registros modificados en el manejador, y después terminar su ejecución.

En el manejador se ha modificado el contenido de los registros **\$at**, **\$v0** y **\$a0**. Por tanto, el contenido de dichos registros debe ser restaurando, para lo cual se empleará el siguiente código:

```
# Restaurar registros salvados
# Restaurar $v0
lw    $v0, sv_v0
# Restaurar $a0
lw    $s0, sv_a0
# Restaurar $at
lw    $at, sv_at
```

Aunque en este manejador sencillo no se contempla tal situación, en ocasiones el manejador de excepción debe devolver el control al programa en curso, Para ello se emplea el siguiente código:

```
# Retornar del manejador de excepción
# (nunca se va a ejecutar en esta versión del manejador)
return_exception:
eret
```

## Uso de syscall para imprimir mensajes en la consola


En **MARS**, la manera de imprimir información en la consola consiste en invocar un servicio del sistema operativo mediante la instrucción **syscall**. Utilizar la instrucción **syscall** desde dentro del manejador de excepciones

no es correcto, ya que supondría generar una excepción desde dentro del propio manejador, y, tal como está diseñado, el manejador no es reentrante. Sin embargo, al invocar un servicio cualquiera entre los ofertados por **MARS** no se genera una excepción, sino que **MARS** captura la solicitud de servicio y ejecuta un código Java que realiza la operación solicitada.

## REALIZACIÓN DEL CUESTIONARIO 2.1.3


Abrir **MARS**, y entrar en **Exception Handler ...** dentro del menú **Settings**. Una vez dentro, activar la opción **Include this exception handler file in all assemble operations** y usar los botones **Browse** y **OK** para cargar el manejador incluido en el fichero **excepciones\_generico.asm**.

A continuación, cargar el fichero **caso01.asm** con la opción **File -> Open**, y ensamblarlo con la opción **Run -> Assemble**. Después, ejecutar el programa mediante la opción **Run -> Go**.

1.  Según el mensaje mostrado en pantalla, ¿cuál es código de la excepción producida al ejecutar el código del fichero **caso01.asm**, dado en base 10?
2. En la zona de salvaguarda de registros del manejador, ¿por qué no se copia el contenido inicial de **\$k0** en memoria?
  - a) No es cierto que no se copie el contenido inicial de **\$k0** en memoria.
  - b)** El contenido inicial de **\$k0** no se copia en memoria porque es un registro de uso libre por parte del *kernel*.
  - c) Sí se copia en memoria: el contenido inicial de **\$k0** es el mismo que el de **\$at**.
  - d) Ninguna de las restantes respuestas es cierta.
3. En las primeras instrucciones del manejador, ¿es importante el orden en que se salvaguardan los registros en memoria?
  - a)** No, el orden de salvaguarda de los registros en memoria es indiferente.
  - b)** Es necesario copiar en memoria **\$at** antes que los demás.
  - c) Podría grabarse primero **\$at** o **\$a0**, da igual. Lo que sí debe hacerse es grabar **\$v0** el último, por el convenio de llamada a subrutina.
  - d) Ninguna de las restantes respuestas es cierta.
4. En las últimas instrucciones del manejador, ¿es importante el orden en que se restauran los registros desde memoria?
  - a) Los registros deben forzosamente ser restaurados en el mismo orden en el que se salvaron.
  - b) Los registros deben forzosamente ser restaurados en el orden inverso del orden en el que fueron salvados.
  - c) El orden da igual siempre que **\$v0** se restaure el primero.
  - d) El orden da igual siempre que **\$at** se restaure el último.


- e)** Ninguna de las restantes respuestas es cierta.
5. Indicar qué hace el par de instrucciones siguientes, que están en las líneas 45 y 47 del manejador de excepciones:
- ```
mfc0 $k0, $13  
andi $k0, $k0, 0x7c
```
- a) Ponen en \$k0 el código de la excepción producida.  
b) Ponen en \$k0 el campo de interrupciones pendientes del registro Cause.  
**c)** Ponen en \$k0 el contenido del campo ExcCode del registro Cause.  
d) Ninguna de las restantes respuestas es cierta.
6. ¿Para qué sirve la instrucción srl \$a0, \$a0, 2 de la línea 60 del manejador?
- a)** Multiplica por 4 el código de excepción del registro Cause, copiado en \$k0 previamente, para escribirlo después por pantalla.  
b) Divide por 4 el código de excepción del registro Cause, copiado en \$k0 previamente, para escribirlo después por pantalla.  
c) Multiplica por 4 el valor del campo ExcCode del registro Cause, copiado en \$k0 previamente, para calcular el código de excepción y escribirlo después por pantalla.  
**d)** Divide por 4 el valor del campo ExcCode del registro Cause, copiado en \$k0 previamente, para calcular el código de excepción y escribirlo después por pantalla.  
**e)** Ninguna de las restantes respuestas es cierta.
7. ¿Cuáles de los siguientes motivos provocan que el manejador no sea reentrante? (Indicar varios si es preciso)
- a) El manejador no tiene distintos puntos de entrada.  
**b)** Los registros modificados no se salvaguardan en la pila.  
c) El código de excepción se calcula sólo una vez.  
**d)** Las excepciones están inhabilitadas durante toda la ejecución del manejador.  
**e)** Las interrupciones están inhabilitadas durante toda la ejecución del **manejador**.  
f) Ninguno de los restantes motivos influye para que el manejador no sea reentrante.

Pulsar el botón **Clear** de la zona de mensajes y de la consola, y cerrar el fichero **caso01.asm** mediante la opción **File -> Close**. Manteniendo cargado el manejador de excepciones, abrir el fichero **caso02.asm**, y ensamblarlo mediante la opción **Run -> Assemble**. Después, ejecutar el programa mediante la opción **Run -> Go**.


8.  Según el mensaje mostrado en pantalla, ¿cuál es código de la excepción producida al ejecutar el código del fichero **caso02.asm**, dado en base 10?

Pulsar el botón **Clear** de la zona de mensajes y de la consola, y cerrar el fichero **caso02.asm** mediante la opción **File -> Close**. Manteniendo cargado


el manejador de excepciones, abrir el fichero **caso03.asm**, y ensamblarlo mediante la opción **Run -> Assemble**. Después, ejecutar el programa mediante la opción **Run -> Go**.

9.  Según el mensaje mostrado en pantalla, ¿cuál es código de la excepción producida al ejecutar el código del fichero **caso03.asm**, dado en base 10?
10. ¿Por qué se produce una excepción al ejecutar el programa del fichero **caso03.asm**?
- a) Porque la instrucción **syscall** en MARS siempre produce una excepción.
  - b) Porque hay un acceso a memoria incorrectamente alineado.
  - c) Porque el servicio solicitado no está implementado.
  - d) Ninguna de las restantes respuestas es cierta.


Pulsar el botón **Clear** de la zona de mensajes y de la consola, y cerrar el fichero **caso03.asm** mediante la opción **File -> Close**. Manteniendo cargado el manejador de excepciones, abrir el fichero **caso04.asm**, y ensamblarlo mediante la opción **Run -> Assemble**. Después, ejecutar el programa mediante la opción **Run -> Go**.

11. Según el mensaje mostrado en pantalla, ¿cuál es código de la excepción producida al ejecutar el código del fichero **caso04.asm**, dado en base 10? 

Pulsar el botón **Clear** de la zona de mensajes y de la consola, y cerrar el fichero **caso04.asm** mediante la opción **File -> Close**. Manteniendo cargado el manejador de excepciones, abrir el fichero **caso05.asm**, y ensamblarlo mediante la opción **Run -> Assemble**. Después, ejecutar el programa mediante la opción **Run -> Go**.

12.  Según el mensaje mostrado en pantalla, ¿cuál es código de la excepción producida al ejecutar el código del fichero **caso05.asm**, dado en base 10?

Pulsar el botón **Clear** de la zona de mensajes y de la consola, y cerrar el fichero **caso05.asm** mediante la opción **File -> Close**. Manteniendo cargado el manejador de excepciones, abrir el fichero **caso06.asm**, y ensamblarlo mediante la opción **Run -> Assemble**. Después, ejecutar el programa mediante la opción **Run -> Go**.

13.  Según el mensaje mostrado en pantalla, ¿cuál es código de la excepción producida al ejecutar el código del fichero **caso06.asm**, dado en base 10?
14. ¿Qué sucede al ejecutar cada una de las pruebas hasta el final?
- a) Se produce un error de ejecución y MARS interrumpe la simulación.
  - b) El manejador aborta la ejecución del programa que generó la excepción.

- c) El manejador retorna el control al programa que generó la excepción, y éste ejecuta hasta el final.
  - d) Ninguna de las restantes respuestas es cierta.
15. Indicar cuál de las siguientes afirmaciones es **CIERTA** sobre la ejecución de los programas de los ficheros **caso01.asm**, **caso02.asm**, **caso03.asm**, **caso04.asm**, **caso05.asm** y **caso06.asm**:
- a) Todos ellos generarán siempre un error fatal, sea cual sea el manejador que se utilice.
  - b) Uno de los programas que hemos probado no generaría error fatal si el manejador estuviera correctamente escrito.
  - c) Ninguno de los programas que hemos probado generaría error fatal si el manejador estuviera correctamente escrito.
  - d)** Si el servicio solicitado por el programa de **caso03.asm** estuviera implementado, ese programa no generaría error fatal.
  - e) Ninguna de las restantes respuestas es cierta.

Pulsar el botón **Clear** de la zona de mensajes y de la consola, cerrar el fichero **caso06.asm** mediante la opción **File -> Close**, y cerrar **MARS**.

## **2. 1. 4. MANEJADOR CON MENSAJES ESPECÍFICOS**

### **INTRODUCCIÓN**

En este apartado de la práctica se probarán los mismos programas que se utilizaron en los apartados previos, pero utilizando un manejador de excepciones distinto al empleado con anterioridad.

Este manejador, como el anterior, presenta mensajes informativos que ayudan a identificar la excepción, y aborta la ejecución del programa que la provocó. Sin embargo, también presenta un mensaje específico adicional que dependerá de la excepción concreta que se haya producido. Para ello, en la zona de datos del manejador se incluyen dos elementos nuevos:

- Un conjunto de cadenas de caracteres que contienen los mensajes individualizados que se presentarán. Habrá una cadena por cada tipo de excepción existente.
- Una tabla de punteros a dichas tiras de caracteres, ordenados en función del código de la excepción a la que se encuentran asociados.

Además, dentro de la zona de código se añaden varias líneas que imprimen en la consola el mensaje específico.

Junto con este ejercicio se planteará un cuestionario con una serie de preguntas que se responderán a través del Aula Virtual.

### **DESCRIPCIÓN DEL MANEJADOR**

En términos generales, la estructura del manejador propuesto es idéntica a la del anterior. Las diferencias se presentan, sobre todo, en la zona de datos, donde se crean las cadenas de caracteres y la tabla de punteros a las mismas, mientras que en la zona de código aparece como novedad la escritura del mensaje específico para la excepción producida.

### **ANÁLISIS DEL CÓDIGO FUENTE DEL MANEJADOR**

#### **Código compartido con el manejador previo**

El nuevo manejador comparte con el anterior:

- Los recursos y el procedimiento para salvaguardar registros.
- La identificación de la causa de la excepción.
- Los recursos y el procedimiento para la presentación por consola de los mensajes informativos con el contenido de los registros **VAddr**, **Status**, **Cause** y **EPC**.
- El procedimiento para abortar el programa en curso.
- El código para restaurar registros y retornar del manejador.

## Mensajes específicos

En la zona de datos del nuevo manejador se añade un conjunto de cadenas de caracteres con mensajes específicos, con un mensaje para cada excepción posible (32 mensajes en total), aunque no esté soportada en **MARS**. Algunos mensajes están vacíos. Aquí se incluyen los mensajes más relevantes:

# Mensajes específicos para cada excepción

```
INT_MSG:    .asciiz    "[Interrupción]"
ADEL_MSG:    .asciiz    "[Error de dirección en lectura de dato o instrucción]"
ADES_MSG:    .asciiz    "[Error de dirección en almacenamiento]"
SYSCALL_MSG: .asciiz    "[Llamada a sistema no implementada]"
BKPT_MSG:    .asciiz    "[Punto de ruptura]"
OV_MSG:      .asciiz    "[Desbordamiento aritmético]"
TRAP_MSG:    .asciiz    "[Trap]"
```

También se incluye una tabla con punteros a dichas cadenas de caracteres. Esta tabla contiene 32 punteros, cada uno de los cuales ocupa 4 bytes. Los punteros están ordenados de acuerdo con el código de excepción cuya cadena referencian. La tabla punteros se muestra a continuación:

# Tabla de punteros a los mensajes de excepción

```
MSG_TABLE:  .align     2
             .word INT_MSG, MOD_MSG, TLBL_MSG, TLBS_MSG      # Excepciones 0-3
             .word ADEL_MSG, ADES_MSG, IBE_MSG, DBE_MSG      # Excepciones 4-7
             .word SYSCALL_MSG, BKPT_MSG, RI_MSG, CpU_MSG     # Excepciones 8-11
             .word OV_MSG, TRAP_MSG, R14_MSG, DIVZ_MSG        # Excepciones 12-15
             .word ID16_MSG, ID17_MSG, C2E_MSG, R19_MSG       # Excepciones 16-19
             .word R20_MSG, R21_MSG, MDMX_MSG, WATCH_MSG     # Excepciones 20-23
             .word MCHECK_MSG, R25_MSG, R26_MSG, R27_MSG      # Excepciones 24-27
             .word R28_MSG, R29_MSG, CACHE_ERR_MSG, R31_MSG   # Excepciones 28-31
```

El mensaje específico se escribe tras el mensaje inicial que indica que se ha producido una excepción junto con su código, en la misma línea:

```
# Imprimir mensaje genérico de aviso de excepción
li    $v0, 4          # syscall 4 (print_str)
la    $a0, __msgExc__
syscall

# Imprimir causa de la excepción
# Imprimir código de causa (decimal)
li    $v0, 4          # syscall 4 (print_str)
la    $a0, __msgExcCode__
syscall
li    $v0, 1          # syscall 1 (print_int)
srl   $a0, $k0, 2
syscall
# Imprimir mensaje indicando la excepción producida
li    $v0, 4          # syscall 4 (print_str)
la    $a0, MSG_TABLE
addu  $a0, $a0, $k0
lw    $a0, 0($a0)
syscall
li    $v0, 11         # syscall 11 (print_char)
lb    $a0, __eoln__
syscall
```



La parte del código resaltada en **negrita** escribe el mensaje específico en la consola. Cada puntero en la tabla de mensajes se encuentra a **4\*ExcCode**

bytes del comienzo de la misma. Como en \$k0 tenemos precisamente ese valor, basta con cargar un puntero al comienzo de la tabla y sumarle \$k0 para tener la dirección del puntero buscado. Cargando dicho puntero en \$a0, es posible utilizar el servicio de impresión de cadenas de caracteres y escribir por consola el mensaje específico.


## REALIZACIÓN DEL CUESTIONARIO 2.1.4

Abrir **MARS**, y entrar en **Exception Handler ...** dentro del menú **Settings**. Una vez dentro, activar la opción **Include this exception handler file in all assemble operations** y usar los botones **Browse** y **OK** para cargar el manejador incluido en el fichero **excepciones\_especifico.asm**.

A continuación, cargar el fichero **caso01.asm** con la opción **File -> Open**, y ensamblarlo con la opción **Run -> Assemble**. Después, poner un punto de ruptura en la instrucción con dirección **0x800001dc**. Finalmente, ejecutar el programa mediante la opción **Run -> Go**.


1.  detener la ejecución en el punto de ruptura **0x800001dc**, ¿cuál es el contenido del registro **\$a0** en hexadecimal?
2. ¿Con qué corresponde el contenido de **\$a0** en ese momento?
  - a) Con la dirección base de uno de los mensajes específicos.
  - b)** Con la dirección base de comienzo de la tabla de punteros a los mensajes específicos.
  - c) No apunta a ningún sitio en particular.
  - d) Ninguna de las restantes respuestas es cierta.
3.  ¿Cuál es el contenido de **\$k0** en decimal?
4. ¿A qué equivale en ese momento el contenido de **\$k0**?
  - a) Al código de excepción.
  - b)** A la distancia del puntero a mensaje específico respecto de la dirección base de la tabla de punteros a mensajes específicos.
  - c) Al tamaño de la tabla de punteros a mensajes específicos.
  - d) Ninguna de las restantes respuestas es cierta.

Ejecutar la instrucción **addu \$a0, \$a0, \$k0**, que está en la dirección **0x800001dc**.

5.  ¿Cuál es ahora el contenido de **\$a0**?
6. ¿Para qué sirve la instrucción **addu \$a0, \$a0, \$k0** recién ejecutada?
  - a) Para poner a **\$a0** apuntando al puntero al mensaje específico que hay que presentar en la consola.
  - b)** Para poner a **\$a0** apuntando al mensaje específico que hay que presentar por la consola.
  - c) Esa instrucción no hace nada útil y se podría eliminar.
  - d) Ninguna de las restantes respuestas es cierta.

Ejecutar la instrucción **lw \$a0, 0(\$a0)**, que está en la dirección **0x800001e0**.





7.  ¿Cuál es ahora el contenido de \$a0?
8. ¿Para qué sirve la instrucción `lw $a0, 0($a0)` recién ejecutada?
- a) Para poner a \$a0 apuntando al puntero al mensaje específico que hay que presentar en la consola.
  - b) Para poner a \$a0 apuntando al mensaje específico que hay que presentar por la consola.
  - c) Para poner a \$a0 apuntando al comienzo de la tabla de mensajes específicos.
  - d) Ninguna de las restantes respuestas es cierta.
9. ¿A dónde apunta ahora \$a0?
- a) A la dirección base de la tabla de punteros a mensajes específicos.
  - b) A la dirección base de la cadena `SYSCALL_MSG`.
  - c) A la dirección base de la cadena `ADES_MSG`.
  - d) A la dirección base de la cadena `TRAP_MSG`.
  - e) A la dirección base de la cadena `BKPT_MSG`.
  - f) A la dirección base de la cadena `ADEL_MSG`.
  - g) A la dirección base de la cadena `OV_MSG`.
  - h) Ninguna de las restantes respuestas es cierta.

Ejecutar la instrucción que está en la dirección `0x800001e4`.

10. ¿Cuál es el mensaje que acaba de aparecer en la consola?
- a) Código de excepción : 13 [Trap]
  - b) Código de excepción : 5 [Error de dirección en almacenamiento]
  - c) Código de excepción : 9 [Punto de ruptura]
  - d) Código de excepción : 4 [Error de dirección en lectura de dato o instrucción]
  - e) Código de excepción : 12 [Desbordamiento aritmético]
  - f) Código de excepción : 8 [Llamada a sistema no implementada]

Ejecutar el programa hasta el final. Pulsar el botón **Clear** de la zona de mensajes y de la consola, y cerrar el fichero `caso01.asm` mediante la opción **File -> Close**. Manteniendo cargado el manejador de excepciones, abrir el fichero `caso02.asm`, y ensamblarlo mediante la opción **Run -> Assemble**. Después, poner un punto de ruptura en la instrucción con dirección `0x800001e4`. Finalmente, ejecutar el programa mediante la opción **Run -> Go**.

11.  Al detener la ejecución en el punto de ruptura `0x800001e4`, ¿cuál es el contenido de \$k0 en decimal?
12.  ¿Cuál es en este momento el contenido del registro \$a0 en hexadecimal?
13. ¿A dónde apunta \$a0?

- a) A la dirección base de la tabla de punteros a mensajes específicos.
- b) A la dirección base de la cadena `SYSCALL_MSG`.
- c) A la dirección base de la cadena `ADES_MSG`.
- d) A la dirección base de la cadena `TRAP_MSG`.


- e) A la dirección base de la cadena BKPT\_MSG.
- f)** A la dirección base de la cadena ADEL\_MSG.
- g) A la dirección base de la cadena OV\_MSG.
- h) Ninguna de las restantes respuestas es cierta.

Ejecutar la instrucción que está en la dirección 0x800001e4.

**14. ¿Cuál es el mensaje que acaba de aparecer en la consola?**

- a) Código de excepción : 9 [Punto de ruptura]
- b)** Código de excepción : 5 [Error de dirección en almacenamiento]
- c) Código de excepción : 12 [Desbordamiento aritmético]
- d) Código de excepción : 4 [Error de dirección en lectura de dato o instrucción].
- e) Código de excepción : 13 [Trap]
- f) Código de excepción : 8 [Llamada a sistema no implementada]

Ejecutar el programa hasta el final. Pulsar el botón **Clear** de la zona de mensajes y de la consola, y cerrar el fichero **caso02.asm** mediante la opción **File -> Close**. Manteniendo cargado el manejador de excepciones, abrir el fichero **caso03.asm**, y ensamblarlo mediante la opción **Run -> Assemble**. Después, poner un punto de ruptura en la instrucción con dirección 0x800001e4. Finalmente, ejecutar el programa mediante la opción **Run -> Go**.

**15.**  **1** detener la ejecución en el punto de ruptura 0x800001e4, ¿cuál es el contenido de \$k0 en decimal?

**16.**  **1** ¿Cuál es en este momento el contenido del registro \$a0 en hexadecimal?

**17. ¿A dónde apunta \$a0?**


- a) A la dirección base de la tabla de punteros a mensajes específicos.
- b)** A la dirección base de la cadena SYSCALL\_MSG.
- c) A la dirección base de la cadena ADES\_MSG.
- d) A la dirección base de la cadena TRAP\_MSG.
- e) A la dirección base de la cadena BKPT\_MSG.
- f) A la dirección base de la cadena ADEL\_MSG.
- g) A la dirección base de la cadena OV\_MSG.
- h) Ninguna de las restantes respuestas es cierta.

Ejecutar la instrucción que está en la dirección 0x800001e4.

**18. ¿Cuál es el mensaje que acaba de aparecer en la consola?**

- a) Código de excepción : 9 [Punto de ruptura]
- b) Código de excepción : 5 [Error de dirección en almacenamiento]
- c) Código de excepción : 12 [Desbordamiento aritmético]
- d) Código de excepción : 4 [Error de dirección en lectura de dato o instrucción].
- e) Código de excepción : 13 [Trap]
- f)** Código de excepción : 8 [Llamada a sistema no implementada]

Ejecutar el programa hasta el final. Pulsar el botón **Clear** de la zona de mensajes y de la consola, y cerrar el fichero **caso03.asm** mediante la opción **File -> Close**. Manteniendo cargado el manejador de excepciones, abrir el fichero **caso04.asm**, y ensamblarlo mediante la opción **Run -> Assemble**. Después, poner un punto de ruptura en la instrucción con dirección **0x800001e4**. Finalmente, ejecutar el programa mediante la opción **Run -> Go**.

19.  Al detener la ejecución en el punto de ruptura **0x800001e4**, ¿cuál es el contenido de **\$k0** en decimal?

20.  ¿Cuál es en este momento el contenido del registro **\$a0** en hexadecimal?

21. ¿A dónde apunta **\$a0**?


- a) A la dirección base de la tabla de punteros a mensajes específicos.
- b) A la dirección base de la cadena **SYSCALL\_MSG**.
- c) A la dirección base de la cadena **ADES\_MSG**.
- d) A la dirección base de la cadena **TRAP\_MSG**.
- e)** A la dirección base de la cadena **BKPT\_MSG**.
- f) A la dirección base de la cadena **ADEL\_MSG**.
- g) A la dirección base de la cadena **OV\_MSG**.
- h) Ninguna de las restantes respuestas es cierta.

Ejecutar la instrucción que está en la dirección **0x800001e4**.

22. ¿Cuál es el mensaje que acaba de aparecer en la consola?

- a)** Código de excepción : 9 [Punto de ruptura]
- b) Código de excepción : 5 [Error de dirección en almacenamiento]
- c) Código de excepción : 12 [Desbordamiento aritmético]
- d) Código de excepción : 4 [Error de dirección en lectura de dato o instrucción].
- e) Código de excepción : 13 [Trap]
- f) Código de excepción : 8 [Llamada a sistema no implementada]

Ejecutar el programa hasta el final. Pulsar el botón **Clear** de la zona de mensajes y de la consola, y cerrar el fichero **caso04.asm** mediante la opción **File -> Close**. Manteniendo cargado el manejador de excepciones, abrir el fichero **caso05.asm**, y ensamblarlo mediante la opción **Run -> Assemble**. Después, poner un punto de ruptura en la instrucción con dirección **0x800001e4**. Finalmente, ejecutar el programa mediante la opción **Run -> Go**.

23.  Al detener la ejecución en el punto de ruptura **0x800001e4**, ¿cuál es el contenido de **\$k0** en decimal?

24.  ¿Cuál es en este momento el contenido del registro **\$a0** en hexadecimal?

25. ¿A dónde apunta **\$a0**?

- a) A la dirección base de la tabla de punteros a mensajes específicos.
- b) A la dirección base de la cadena **SYSCALL\_MSG**.
- c) A la dirección base de la cadena **ADES\_MSG**.
- d) A la dirección base de la cadena **TRAP\_MSG**.


- e)** A la dirección base de la cadena BKPT\_MSG.
- f) A la dirección base de la cadena ADEL\_MSG.
- g) A la dirección base de la cadena OV\_MSG.
- h) Ninguna de las restantes respuestas es cierta.

Ejecutar la instrucción que está en la dirección 0x800001e4.

**26. ¿Cuál es el mensaje que acaba de aparecer en la consola?**

- a)** Código de excepción : 9 [Punto de ruptura]
- b) Código de excepción : 5 [Error de dirección en almacenamiento]
- c) Código de excepción : 12 [Desbordamiento aritmético]
- d) Código de excepción : 4 [Error de dirección en lectura de dato o instrucción].
- e) Código de excepción : 13 [Trap]
- f) Código de excepción : 8 [Llamada a sistema no implementada]

Ejecutar el programa hasta el final. Pulsar el botón **Clear** de la zona de mensajes y de la consola, y cerrar el fichero **caso05.asm** mediante la opción **File -> Close**. Manteniendo cargado el manejador de excepciones, abrir el fichero **caso06.asm**, y ensamblarlo mediante la opción **Run -> Assemble**. Después, poner un punto de ruptura en la instrucción con dirección 0x800001e4. Finalmente, ejecutar el programa mediante la opción **Run -> Go**.

**27.**  Al detener la ejecución en el punto de ruptura 0x800001e4, ¿cuál es el contenido de \$k0 en decimal?

**28.**  ¿Cuál es en este momento el contenido del registro \$a0 en hexadecimal?

**29. ¿A dónde apunta \$a0?**

- a) A la dirección base de la tabla de punteros a mensajes específicos.
- b) A la dirección base de la cadena SYSCALL\_MSG.
- c) A la dirección base de la cadena ADES\_MSG.
- d)** A la dirección base de la cadena TRAP\_MSG.
- e) A la dirección base de la cadena BKPT\_MSG.
- f) A la dirección base de la cadena ADEL\_MSG.
- g) A la dirección base de la cadena OV\_MSG.
- h) Ninguna de las restantes respuestas es cierta.

Ejecutar la instrucción que está en la dirección 0x800001e4.

**30. ¿Cuál es el mensaje que acaba de aparecer en la consola?**

- a) Código de excepción : 9 [Punto de ruptura]
- b) Código de excepción : 5 [Error de dirección en almacenamiento]
- c) Código de excepción : 12 [Desbordamiento aritmético]
- d) Código de excepción : 4 [Error de dirección en lectura de dato o instrucción].
- e)** Código de excepción : 13 [Trap]
- f) Código de excepción : 8 [Llamada a sistema no implementada]

Ejecutar el programa hasta el final. Pulsar el botón **Clear** de la zona de mensajes y de la consola, cerrar el fichero **caso06.asm** mediante la opción **File -> Close**, y cerrar **MARS**.

## 2. 1. 5. MANEJADOR CON RUTINAS ESPECÍFICAS

### INTRODUCCIÓN

En un caso real, cada excepción recibe un tratamiento individualizado. Aquí se presenta un manejador capaz de tratar cada excepción por separado, en el que cada excepción cuenta con su propia rutina de tratamiento, para que así el manejador pueda responder de forma diferente según la excepción manejada. El manejador al principio identifica la excepción producida, y después invoca la rutina adecuada. La dirección de comienzo de cada rutina se denomina **vector de excepción**. El manejador incluye una **tabla de vectores de excepción** con 32 punteros (vectores de excepción), cada uno apuntando a la dirección de comienzo de una rutina de tratamiento. La tabla de vectores de excepción está en la zona de datos del manejador. Su funcionamiento se probará con los programas de las secciones anteriores.

Junto con este ejercicio se planteará un cuestionario con una serie de preguntas que se responderán a través del Aula Virtual.

### DESCRIPCIÓN DEL MANEJADOR

La estructura del manejador propuesto es la siguiente:

```
# MANEJADOR DE EXCEPCIONES: VARIABLES Y DEFINICIONES
    .kdata
# Zona de salvaguarda de registros
    ...
# Mensajes genéricos
    ...
# Mensajes específicos
    ...
# Tabla de punteros a mensajes específicos
    ...
# Tabla de vectores de excepción
    ...
# MANEJADOR DE EXCEPCIONES: CÓDIGO
    .ktext      0x80000180
EXCEPTION_HANDLER_ENTRY:
# Salvar registros
    ...
# Identificar la causa de la excepción
    ...
# Cargar vector de excepción
    ...
# Saltar a la rutina de tratamiento de excepción
    ...
# Punto de retorno de las rutinas de tratamiento
continue_exception:
# Restaurar registros salvados
    ...
# Retornar del manejador de excepción
return_exception:
    eret
```

Cada rutina de tratamiento tendrá una estructura diferente. En un primer paso se distinguirán dos casos:

- Una rutina de tratamiento de excepciones de tipo syscall.
- Una rutina de tratamiento genérica para el resto de excepciones.

La rutina genérica de tratamiento presentará una serie de mensajes informativos en la consola y abortará la ejecución del programa en curso.

Por su parte, la rutina de tipo syscall debe hacer lo siguiente:

- Si el servicio invocado está implementado, debe realizarlo y después retornar al manejador para continuar ejecutando el programa que lo solicitó.
- Si el servicio invocado no está implementado, hará lo mismo que la rutina genérica: presentará una serie de mensajes informativos y abortar el programa.

Más adelante también se presentará una rutina específica de tratamiento de interrupciones.

## ANÁLISIS DEL CÓDIGO FUENTE DEL MANEJADOR

### Subrutina de impresión de mensajes

Para no repetir código, la presentación de mensajes informativos por consola, que se realizará en diferentes rutinas de tratamiento, se codificará en una subrutina que podrá ser invocada desde diferentes lugares. El único requisito de esta subrutina es que, antes de ser invocada, el registro **\$k0** debe contener el valor del campo **ExcCode** del registro **Cause** multiplicado por 4. A continuación se presenta el código completo de la subrutina:

**MSG\_PRINT:**

```
# Imprimir mensaje genérico de aviso de excepción
li    $v0, 4          # syscall 4 (print_str)
la    $a0, __msgExc__
syscall

# Imprimir causa de la excepción
# Imprimir código de causa (decimal)
li    $v0, 4          # syscall 4 (print_str)
la    $a0, __msgExcCode__
syscall
li    $v0, 1          # syscall 1 (print_int)
srl   $a0, $k0, 2
syscall
# Imprimir mensaje indicando la excepción producida
li    $v0, 4          # syscall 4 (print_str)
la    $a0, MSG_TABLE
addu  $a0, $a0, $k0
lw    $a0, 0($a0)
syscall
li    $v0, 11         # syscall 11 (print_char)
lb    $a0, __eoln__
syscall
```

```

# Imprimir contenido de los registros Status, Cause, EPC y VAddr
# VAddr
li    $v0, 4          # syscall 4 (print_str)
la    $a0, __msgVAddr__
syscall
li    $v0, 34         # syscall 34 (print_int_hexadecimal)
mfc0  $a0, $8
syscall
li    $v0, 11         # syscall 11 (print_char)
lb    $a0, __eoln__
syscall
# Status
li    $v0, 4          # syscall 4 (print_str)
la    $a0, __msgStatus__
syscall
li    $v0, 34         # syscall 34 (print_int_hexadecimal)
mfc0  $a0, $12
syscall
li    $v0, 11         # syscall 11 (print_char)
lb    $a0, __eoln__
syscall
# Cause
li    $v0, 4          # syscall 4 (print_str)
la    $a0, __msgCause__
syscall
li    $v0, 34         # syscall 34 (print_int_hexadecimal)
mfc0  $a0, $13
syscall
li    $v0, 11         # syscall 11 (print_char)
lb    $a0, __eoln__
syscall
# EPC
li    $v0, 4          # syscall 4 (print_str)
la    $a0, __msgEPC__
syscall
li    $v0, 34         # syscall 34 (print_int_hexadecimal)
mfc0  $a0, $14
syscall
li    $v0, 11         # syscall 11 (print_char)
lb    $a0, __eoln__
syscall
# Retornar
jr    $ra

```

Como la subrutina modifica el registro **\$ra**, será preciso salvarlo en memoria al principio de la rutina de tratamiento específica que la invoque, y restaurarlo al final de la misma.

## Parte genérica del manejador

La zona de datos del manejador contiene la tabla de vectores de excepción:

```

EXV_TABLE:  .word GEN_RTE, GEN_RTE, GEN_RTE, GEN_RTE      # Excepciones 0-3
             .word GEN_RTE, GEN_RTE, GEN_RTE, GEN_RTE      # Excepciones 4-7
             .word SYSCALL_RTE, GEN_RTE, GEN_RTE, GEN_RTE   # Excepciones 8-11
             .word GEN_RTE, GEN_RTE, GEN_RTE, GEN_RTE      # Excepciones 12-15
             .word GEN_RTE, GEN_RTE, GEN_RTE, GEN_RTE      # Excepciones 16-19
             .word GEN_RTE, GEN_RTE, GEN_RTE, GEN_RTE      # Excepciones 20-23
             .word GEN_RTE, GEN_RTE, GEN_RTE, GEN_RTE      # Excepciones 24-27
             .word GEN_RTE, GEN_RTE, GEN_RTE, GEN_RTE      # Excepciones 28-31

```



Además del espacio para guardar copia de **\$at**, **\$v0** y **\$a0**, ahora es preciso reservar hueco también para **\$ra**:

```

        .align 2
sv_at:  .word 0
sv_a0:  .word 0
sv_v0:  .word 0
sv_ra:  .word 0

```

Por otro lado, la parte genérica del manejador sólo guarda la copia de **\$at**, siendo cada rutina específica quien se encargue de guardar los restantes registros que modifique, así que la salvaguarda de registros se efectúa mediante el siguiente fragmento de código:

```

        .ktext      0x80000180
EXCEPTION_HANDLER_ENTRY:
        move $k0, $at      # Copiar $at en $k0
        sw   $k0, sv_at    # Guardar copia de $at en memoria a través de $k0

```

La identificación de la causa de la excepción se realiza del mismo modo que en los manejadores previos:

```

identify_cause:
    # Copiar registro Cause ($13-coprocesador 0) en $k0
    mfc0 $k0, $13
    # Extraer el campo de código de excepción (ExcCode)
    andi $k0, $k0, 0x7c

```

Tras ello, el manejador carga en **\$k1** el vector de excepción necesario mediante el siguiente fragmento de código:

```

get_exception_vector:
    la   $k1, EXV_TABLE
    addu $k1, $k1, $k0
    lw   $k1, 0($k1)

```

Para saltar a la rutina de tratamiento, basta con ejecutar lo siguiente:

```

jump_RTE:
    jr   $k1

```

Las excepciones que se deban a errores no recuperables presentan varios mensajes informativos y abortan la ejecución del programa. Otras excepciones (llamada a servicio **syscall**, por ejemplo) deben retornar el control al programa que las genera, restaurando los registros de propósito general modificados y poniendo a 0 el registro **Cause**. Entonces, la parte final del código del manejador será:

```

# Punto de retorno de las excepciones que no abortan la ejecución del programa
continue_exception:
# Restaurar registros
restore_regs:
    # Restaurar $at
    lw   $at, sv_at
    # Borrar registro Cause ($13-coprocesador 0)
    mtc0 $zero, $13
# Retornar del manejador de excepción
return_exception:
    eret

```

## Rutina genérica de tratamiento de excepción

Esta rutina presenta varios mensajes informativos y luego aborta la ejecución, como en los manejadores previos. Su código es el siguiente:

```
GEN_RTE:
    # Salvar copia de $v0 en memoria
    sw    $v0, sv_v0
    # Salvar copia de $a0 en memoria
    sw    $a0, sv_a0
    # Salvar copia de $ra en memoria
    sw    $ra, sv_ra
    # Imprimir mensajes
    jal   MSG_PRINT
    # Abortar la ejecución del programa que generó la excepción
    li    $v0, 4          # syscall 4 (print_str)
    la    $a0, __msgAbort__
    syscall
    li    $v0, 17         # syscall 17 (exit with code)
    li    $a0, 1          # Error
    syscall
```

No es necesario restaurar registros ni hacer nada más, ya que tras ejecutar esta rutina el simulador se detendrá.

## Rutina de tratamiento de la excepción syscall

Esta rutina deberá identificar el servicio solicitado, y llamar a una rutina aún más específica de servicio del mismo, que deberá retornar a la rutina de **syscall** para restaurar los registros modificados dentro de la misma y volver al manejador genérico, que terminará devolviendo el control al programa que invocó el servicio.

En caso de que el servicio invocado no esté implementado en el manejador, se producirá un error fatal, y la rutina de **syscall** saltará a la rutina genérica de tratamiento de excepción, que escribe en la consola varios mensajes informativos y aborta la ejecución del programa.

Aparte de los servicios soportados directamente por **MARS**, y que son capturados por el mismo y realizados mediante código Java sin generar ninguna excepción, hasta el momento no se ha propuesto la implementación de ningún servicio con **syscall**. Por ello, de momento sólo se puede implementar un esqueleto de la rutina, indicando mediante comentarios las acciones que deberían ser realizadas por la misma.

```
SYSCALL_RTE:
    # Chequear el registro $v0 para identificar el servicio y saltar a rutina
    # Si el servicio no está implementado, presentar mensajes y abortar
    j     GEN_RTE
    # Punto de retorno para los servicios syscall implementados
    continúe_syscall:
    # Sumar 4 al EPC para saltarse la instrucción syscall invocadora
    mfc0  $k0, $14
    addi  $k0, $k0, 4
    mtc0  $k0, $14
    # Volver al cuerpo principal del manejador
```

j      `continue_exception`

## REALIZACIÓN DEL CUESTIONARIO 2.1.5

Abrir **MARS**, y entrar en **Exception Handler ...** dentro del menú **Settings**. Una vez dentro, activar la opción **Include this exception handler file in all assemble operations** y usar los botones **Browse** y **OK** para cargar el manejador incluido en el fichero **excepciones\_rutinas.asm**.

A continuación, cargar el fichero **caso01.asm** con la opción **File -> Open**, y ensamblarlo con la opción **Run -> Assemble**. Después, poner un punto de ruptura en la instrucción con dirección **0x8000019c**. Finalmente, ejecutar el programa mediante la opción **Run -> Go**.

1. Al detener la ejecución en el punto de ruptura **0x8000019c**, ¿cuál es el contenido del registro **\$k1** en hexadecimal?
2. ¿Con qué corresponde el contenido de **\$k1** en ese momento?
  - a) Con la dirección base de comienzo de la tabla de punteros a los mensajes específicos.
  - b) Con la dirección base de la tabla de vectores de excepción.
  - c) Con el vector de excepción correspondiente a la excepción producida.
  - d) Ninguna de las restantes respuestas es cierta.
3. ¿Cuál es el contenido de **\$k0** en decimal?
4. ¿A qué equivale en ese momento el contenido de **\$k0**?
  - a) Al código de excepción.
  - b) A la distancia del vector de excepción buscado respecto de la dirección base de la tabla de vectores de excepción.
  - c) Al tamaño de la tabla de vectores de excepción.
  - d) Ninguna de las restantes respuestas es cierta.

Ejecutar la instrucción **addu \$k1,\$k1,\$k0**, que está en la dirección **0x8000019c**.

5. ¿Cuál es ahora el contenido de **\$k1**?
6. ¿Para qué sirve la instrucción **addu \$k1,\$k1,\$k0** recién ejecutada?
  - a) Para poner a **\$k1** apuntando al puntero al mensaje específico que hay que presentar en la consola.
  - b) Para poner a **\$k1** apuntando al vector de excepción buscado.
  - c) Esa instrucción no hace nada útil y se podría eliminar.
  - d) Ninguna de las restantes respuestas es cierta.

Ejecutar la instrucción **lw \$k1,0(\$k1)**, que está en la dirección **0x800001a0**.

7. ¿Cuál es ahora el contenido de **\$k1**?
8. ¿Para qué sirve la instrucción **lw \$k1,0(\$k1)** recién ejecutada?
  - a) Para poner a **\$k1** apuntando al comienzo de la tabla de vectores de excepción.
  - b) Para poner a **\$k1** apuntando al vector de excepción buscado.

- c)** Para poner en \$k1 el vector de excepción buscado.
- d) Ninguna de las restantes respuestas es cierta.

9. ¿A dónde apunta ahora \$k1?

- a)** Al comienzo de la rutina genérica de tratamiento de excepción.
- b) Al comienzo de la rutina de tratamiento de la excepción syscall.
- c) Al comienzo de la rutina de tratamiento de interrupciones.
- d) Ninguna de las restantes respuestas es cierta.

Ejecutar el programa hasta el final.

10. A juzgar por los mensajes que aparecen por pantalla, ¿qué excepción se ha producido?

- a) Excepción de trap (13).
- b) Excepción por error de dirección en almacenamiento (5).
- c) Excepción por punto de ruptura (9).
- d)** Excepción por error de dirección en lectura de dato o instrucción (4).
- e) Excepción por desbordamiento aritmético (12).
- f) Excepción por llamada a sistema no implementada (8).

Pulsar el botón **Clear** de la zona de mensajes y de la consola, y cerrar el fichero **caso01.asm** mediante la opción **File -> Close**. Manteniendo cargado el manejador de excepciones, abrir el fichero **caso02.asm**, y ensamblarlo mediante la opción **Run -> Assemble**. Después, poner un punto de ruptura en la instrucción con dirección 0x800001a4. Finalmente, ejecutar el programa mediante la opción **Run -> Go**.



Al detener la ejecución en el punto de ruptura 0x800001a4, ¿cuál es el contenido de \$k0 en decimal?

12. ¿Cuál es en este momento el contenido del registro \$k1 en hexadecimal?

13. ¿A dónde apunta \$k1?


- a)** Al comienzo de la rutina genérica de tratamiento de excepción.
- b) Al comienzo de la rutina de tratamiento de la excepción syscall.
- c) Al comienzo de la rutina de tratamiento de interrupciones.
- d) Ninguna de las restantes respuestas es cierta.

Ejecutar el programa hasta el final.

14. A juzgar por los mensajes que aparecen por pantalla, ¿qué excepción se ha producido?

- a) Excepción de trap (13).
- b)** Excepción por error de dirección en almacenamiento (5).
- c) Excepción por punto de ruptura (9).
- d) Excepción por error de dirección en lectura de dato o instrucción (4).
- e) Excepción por desbordamiento aritmético (12).
- f) Excepción por llamada a sistema no implementada (8).

Pulsar el botón **Clear** de la zona de mensajes y de la consola, y cerrar el fichero **caso02.asm** mediante la opción **File -> Close**. Manteniendo cargado el manejador de excepciones, abrir el fichero **caso03.asm**, y ensamblarlo mediante la opción **Run -> Assemble**. Después, poner un punto de ruptura en la instrucción con dirección **0x800001a4**. Finalmente, ejecutar el programa mediante la opción **Run -> Go**.

15.  Al detener la ejecución en el punto de ruptura **0x800001a4**, ¿cuál es el contenido de **\$k0** en decimal?

16.  ¿Cuál es en este momento contenido del registro **\$k1** en hexadecimal?

17. ¿A dónde apunta **\$k1**?

- a) Al comienzo de la rutina genérica de tratamiento de excepción.
- b)** Al comienzo de la rutina de tratamiento de la excepción **syscall**.
- c) Al comienzo de la rutina de tratamiento de interrupciones.
- d) Ninguna de las restantes respuestas es cierta.

Ejecutar el programa hasta el final.

18. A juzgar por los mensajes que aparecen por pantalla, ¿qué excepción se ha producido?

- a) Excepción de trap (13).
- b) Excepción por error de dirección en almacenamiento (5).
- c) Excepción por punto de ruptura (9).
- d) Excepción por error de dirección en lectura de dato o instrucción (4).
- e) Excepción por desbordamiento aritmético (12).
- f)** Excepción por llamada a sistema no implementada (8).

Pulsar el botón **Clear** de la zona de mensajes y de la consola, y cerrar el fichero **caso03.asm** mediante la opción **File -> Close**. Manteniendo cargado el manejador de excepciones, abrir el fichero **caso04.asm**, y ensamblarlo mediante la opción **Run -> Assemble**. Después, poner un punto de ruptura en la instrucción con dirección **0x800001a4**. Finalmente, ejecutar el programa mediante la opción **Run -> Go**.

19.  Al detenerse en el punto de ruptura **0x800001a4**, ¿cuál es el contenido de **\$k0** en decimal?

20.  ¿Cuál es en este momento contenido del registro **\$k0** en hexadecimal?

21. ¿A dónde apunta **\$k1**?

- a)** Al comienzo de la rutina genérica de tratamiento de excepción.
- b)** Al comienzo de la rutina de tratamiento de la excepción **syscall**.
- c) Al comienzo de la rutina de tratamiento de interrupciones.
- d) Ninguna de las restantes respuestas es cierta.



Ejecutar el programa hasta el final.

22. A juzgar por los mensajes que aparecen por pantalla, ¿qué excepción se ha producido?

- a) Excepción de trap (13).

- b) Excepción por error de dirección en almacenamiento (5).
- c)** Excepción por punto de ruptura (9).
- d) Excepción por error de dirección en lectura de dato o instrucción (4).
- e) Excepción por desbordamiento aritmético (12).
- f)** Excepción por llamada a sistema no implementada (8).



Pulsar el botón **Clear** de la zona de mensajes y de la consola, y cerrar el fichero **caso04.asm** mediante la opción **File -> Close**. Manteniendo cargado el manejador de excepciones, abrir el fichero **caso05.asm**, y ensamblarlo mediante la opción **Run -> Assemble**. Después, poner un punto de ruptura en la instrucción con dirección 0x800001a4. Finalmente, ejecutar el programa mediante la opción **Run -> Go**.

23.  Al detenerse en el punto de ruptura 0x800001a4, ¿cuál es el contenido de \$k0 en decimal?
24.  ¿Cuál es en este momento contenido del registro \$k1 en hexadecimal?
25. ¿A dónde apunta \$k1?
- a)** Al comienzo de la rutina genérica de tratamiento de excepción.
  - b) Al comienzo de la rutina de tratamiento de la excepción syscall.
  - c) Al comienzo de la rutina de tratamiento de interrupciones.
  - d) Ninguna de las restantes respuestas es cierta.

Ejecutar el programa hasta el final.

26. A juzgar por los mensajes que aparecen por pantalla, ¿qué excepción se ha producido?
- a) Excepción de trap (13).
  - b) Excepción por error de dirección en almacenamiento (5).
  - c) Excepción por punto de ruptura (9).
  - d) Excepción por error de dirección en lectura de dato o instrucción (4).
  - e)** Excepción por desbordamiento aritmético (12).
  - f) Excepción por llamada a sistema no implementada (8).

Pulsar el botón **Clear** de la zona de mensajes y de la consola, y cerrar el fichero **caso05.asm** mediante la opción **File -> Close**. Manteniendo cargado el manejador de excepciones, abrir el fichero **caso06.asm**, y ensamblarlo mediante la opción **Run -> Assemble**. Después, poner un punto de ruptura en la instrucción con dirección 0x800001a4. Finalmente, ejecutar el programa mediante la opción **Run -> Go**.

27.  Al detenerse en el punto de ruptura 0x800001a4, ¿cuál es el contenido de \$k0 en decimal?
28.  ¿Cuál es en este momento contenido del registro \$k1 en hexadecimal?
29. ¿A dónde apunta \$k1?
- a)** Al comienzo de la rutina genérica de tratamiento de excepción.
  - b) Al comienzo de la rutina de tratamiento de la excepción syscall.

- c) Al comienzo de la rutina de tratamiento de interrupciones.
- d) Ninguna de las restantes respuestas es cierta.

Ejecutar el programa hasta el final.

**30. A juzgar por los mensajes que aparecen por pantalla, ¿qué excepción se ha producido?**

- a)** Excepción de trap (13).
- b) Excepción por error de dirección en almacenamiento (5).
- c) Excepción por punto de ruptura (9).
- d) Excepción por error de dirección en lectura de dato o instrucción (4).
- e) Excepción por desbordamiento aritmético (12).
- f) Excepción por llamada a sistema no implementada (8).

Pulsar el botón **Clear** de la zona de mensajes y de la consola, cerrar el fichero **caso06.asm** mediante la opción **File -> Close**, y cerrar **MARS**.