MEMORIA

Índice

DATOS

→ Nombre: Raúl Beltrán Marco

→ Email: rbm61@alu.ua.es/raulbeltmarc@gmail.com

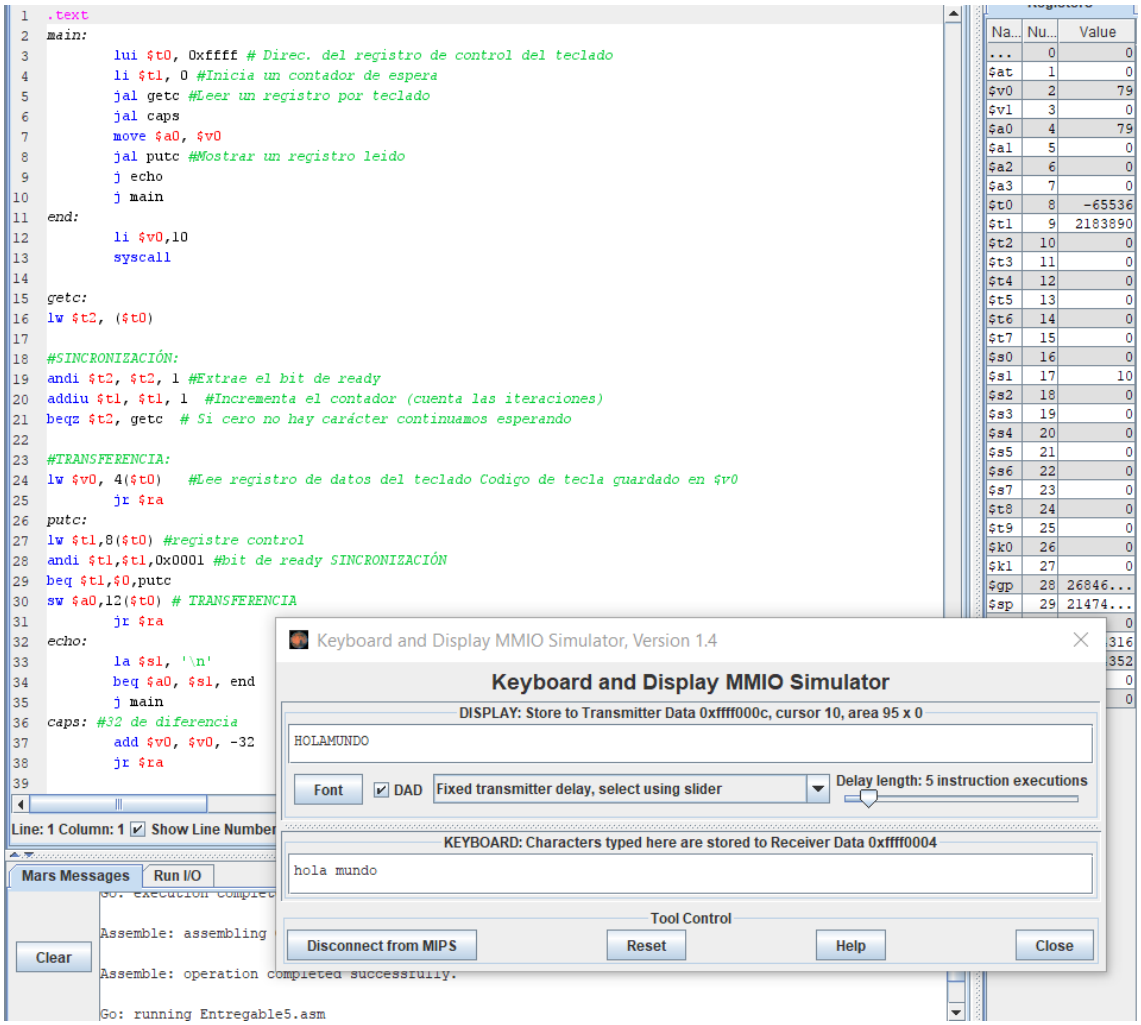
→ Grupo: lunes /05/19

PRÁCTICA 10

Transforma el programa echo en el programa caps que muestra por la consola la mayúscula del carácter introducido por el teclado. Supón que todos los caracteres introducidos están en minúscula.

Como podemos observar, para realizar este entregable hemos creado la función que se llama “caps”, la cual la llamamos después de la función getc, que es la que se encarga de obtener el dato introducido por teclado. En la función caps lo que hacemos es restar 32 al dato introducido debido a que es la diferencia que existe según el Código ASCII entre las letras y sus respectivas mayúsculas.

El programa acaba cuando hacemos un salto de línea, el cual lo comprobamos en la función echo con la instrucción beq.



Dado el siguiente código, complétalo escribiendo la función read_string. Esta función tiene que leer del teclado la cadena de caracteres que introduzca el usuario y tiene que almacenarla en un buffer denominado cadena. La cadena finaliza cuando el usuario teclee un salto de línea. Posteriormente el programa muestra la cadena en la consola. Al escribir la función read_string no olvidéis meter en el buffer el carácter de salto de línea.

En este entregable mostramos la función read_string, la cual nos pide el ejercicio que creemos.

En esta función lo **primero** que hacemos es guardar la dirección de cadena en $t3 e inicializar el contador a 0 ($t4).

Después, comenzamos nuestro **bucle "loop"**, en el cual guardaremos en $t5 la suma de la dirección y el contador anteriormente mencionados. Luego, guardamos en $t0 la dirección 0xFFFF, que es el control de teclado e inicializamos el contador de espera a 0.

```
jal read_string
la $a0,cadena
jal print_string
li $v0,10
syscall

############################################################
print_string:
        la $t0,0xFFFF0000
sync:
        lw $t1, ControlDisplay($t0)
        andi $t1,$t1,1
        beqz $t1,sync

        lbu $t1,0($a0)
        beqz $t1,final
        sw $t1, BufferDisplay($t0)
        addi $a0,$a0,1
        j sync
final:
        jr $ra
############################################################
read_string:
        la $t3, cadena
        li $t4, 0
loop:
        add $t5, $t3, $t4
        lui $t0, 0xffff #Dirección del regitro de control del teclado
        li $t1, 0 #Inicia el contador de espera
esperar:
        lw $t2, ($t0) #Lee Registro introducido por teclado
        andi $t2, $t2, 1 #Extrae el bit de ready
        addiu $t1, $t1, 1 #Aumentamos el contador

        beqz $t2, esperar #Si $t2 vale 0, no hay caracter sigue esperando
        lw $v0, 4($t0)

        move $a0, $v0
        sb $a0, 0($t5)
        addi $t4, $t4, 1
        bne $a0, 10, loop # El salto de linea equivale al 10
        ##Ha introducido \n, acabamos el bucle
        jr $ra
```

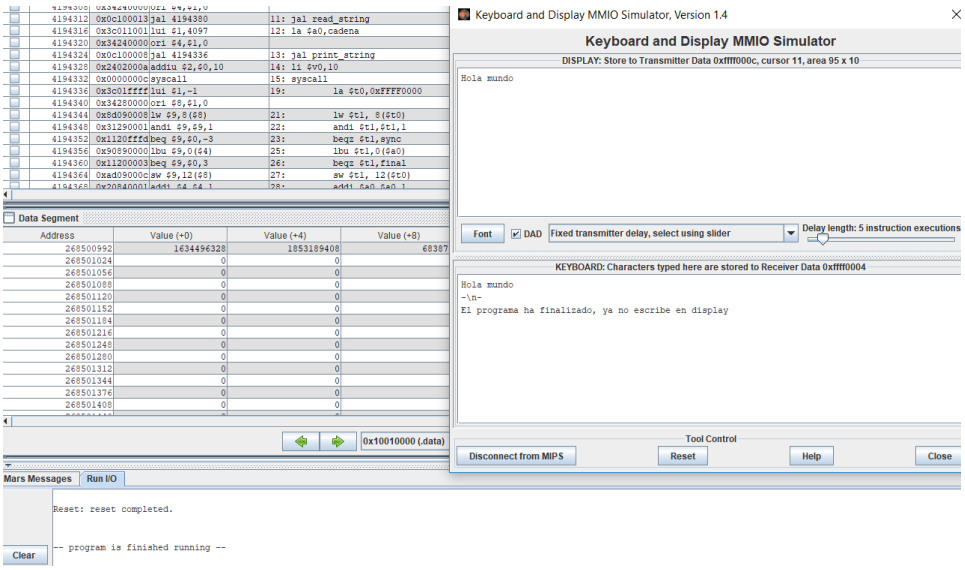
Creamos la etiqueta **esperar**, porque después de leer el registro introducido con la instrucción lw y aumentar en 1 tanto el contador como el bit de ready, haremos una comparación con la **instrucción beqz**, que se encarga de crear un bucle infinito mientras el usuario no introduzca ningún registro por teclado.

Una vez se ha introducido un registro, usamos la instrucción sb para guardar en $t5 el valor de $a0 (El valor que había en $v0, el cual era el registro introducido por teclado)

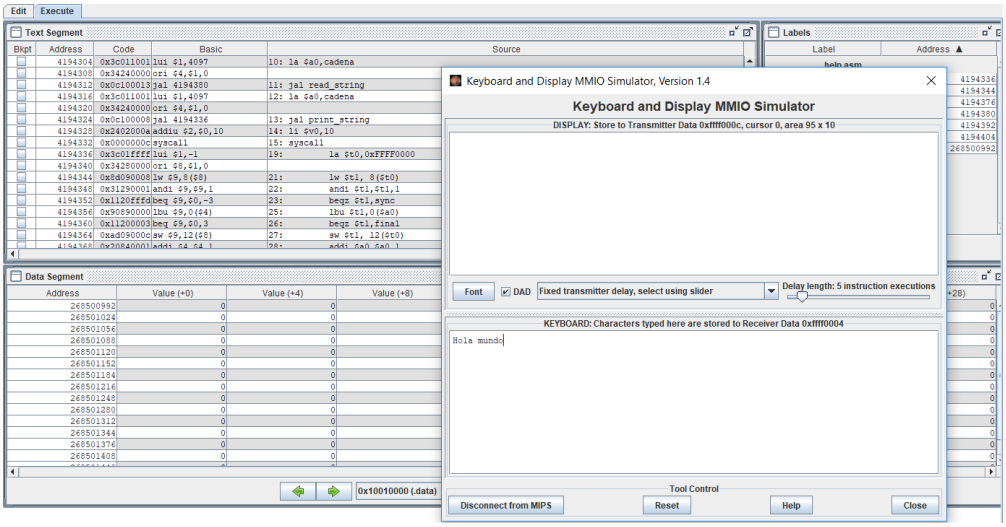
Al final del bucle aumentamos el contador $t4 y comprobamos que el registro introducido no es un salto de línea (equivale al 10 que aparece) con la instrucción bne. En caso de que sí lo sea, el bucle acabará y se llamará a print_string

Ejemplo de ejecución:

Antes del salto de línea



Después del salto de línea



PRÁCTICA 11

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

Para poder realizar este ejercicio simplemente añadiremos la instrucción move para guardar el registro en $v0 y al final la instrucción beq con la cual comprobamos si se ha introducido un salto de línea.

```
.kdata
registros: .word 0,0,0,0 # Espacio para guardar 4 registros
mis1:.asciiz"\nExcepción dirige errónea ocurrida en la dirección:"
mis2:.asciiz "\nExcepción desbordamiento ocurrida en la dirección: "
mis3:.asciiz "\nEn cualquier caso continuamos el programa...\n"
mis4: .asciiz "\nExcepción de dirección no alineada"
.ktext 0x80000180 # Dirección de comienzo de la rutina
# Salvar los registros a utilizar
la $k1, registros
sw $at, 0($k1) # Es importante guardar el registro %at
sw $v0, 4($k1)
sw $a0, 8($k1)
mfc0 $a0, $13 # %a0 <= registro Cause
andi $a0, $a0, 0x3C # extraemos en %a0 el código de excepción
#Detectamos sólo dos excpcions
li $s0, 0x0030 # código Desbordamiento
li $s1, 0x0014 # código error de dirección store
li $s2, 0x00000024 #No alineada
andi $s2, $s2, 100
beq $a0, $s0, Desbordo
bne $a0, $s1, salida
bne $a0, $s2, NoAlineada
la $a0, mis1
li $v0, 4
syscall
mfc0 $a0, $14 # %a0 <= EPC, donde ha ocurrido la excepción
li $v0, 34
syscall # Escribimos EPC en hexadecimal
```

Line: 25 Column: 8 Show Line Numbers

Mars Messages | Run I/O

```
Assemble: assembling C:\Users\IronDarkRL\Desktop\Entregar\13 My\Entregable2.asm

Assemble: operation completed successfully.

Go: running Entregable2.asm


Go: execution terminated by null instruction.
```

Clear

```
syscall
mfc0 $a0, $14 # %a0 <= EPC, donde ha ocurrido la excepción
li $v0, 34
syscall # Escribimos EPC en hexadecimal
Desbordo:
la $a0, mis2
li $v0, 4
syscall
mfc0 $a0, $14 # %a0 <= EPC, donde ha ocurrido la excepción
li $v0, 34
syscall # Escribimos EPC en hexadecimal
salida:
la $a0, mis3
li $v0, 4
syscall
NoAlineada:
la $a0, mis4
li $v0, 4
syscall
#Restauramos los registros
la $k1, registros
lw $at, 0($k1)
lw $v0, 4($k1)
lw $a0, 8($k1)
#Iniciamos registro Vaddr del coprocesador 0
mtc0 $zero, $8
#Cómo se trata de excepciones se actualiza el registro EPC
mfc0 $k0, $14 # $k0 <= EPC
addiu $k0, $k0, 4 # Incremento de $k0 en 4
mtc0 $k0, $14 # Ahora EPC apunta a la siguiente instrucción

eret # Vuelve al programa de usuario
```

Line: 45 Column: 18 Show Line Numbers

Mars Messages | Run I/O

```
En cualquier caso continuamos el programa...

Excepción de dirección no alineada
-- program is finished running (dropped off bottom) --


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

Excepción de dirección no alineada
-- program is finished running (dropped off bottom) --
```

Clear

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

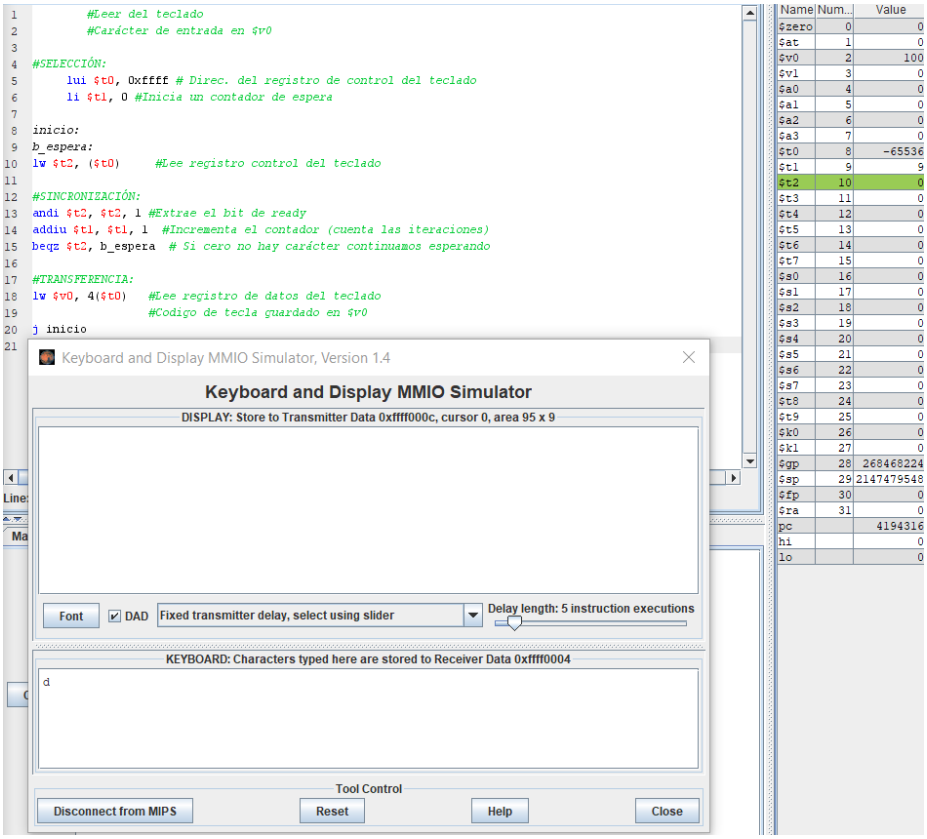
Basicamente, para realizar este ejercicio hemos reutilizado el ejercicio de ejemplo que nos da la Práctica 11, sobre el tratamiento de excepciones. Añadiendo su respectivo mensaje en el .data y su código según la tabla de excepciones y haciendo usa de la instrucción bne para comprobar qué error tenemos que tratar.

```
.kdata
registros: .word 0,0,0,0 # Espacio para guardar 4 registros
mis1:.asciiz"\nExcepción dirige errónea ocurrida en la dirección:"
mis2:.asciiz "\nExcepción desbordamiento ocurrida en la dirección: "
mis3:.asciiz "\nEn cualquier caso continuamos el programa...\n"
mis4: .asciiz "\nExcepción de dirección no alineada"

.ktext 0x80000180 # Dirección de comienzo de la rutina
# Salvar los registros a utilizar
la $k1, registros
sw $at, 0($k1) # Es importante guardar el registro $at
sw $v0, 4($k1)
sw $a0, 8($k1)
mfc0 $a0, $13 # $a0 <= registro Cause
andi $a0, $a0, 0x3C # extraemos en $a0 el código de excepción
#Detectamos sólo dos excpcions
li $s0, 0x0030 # código Desbordamiento
li $s1, 0X0014 # código error de dirección store
li $s2, 0x00000024 #No alineada
beq $a0, $s0, Desbordo
bne $a0, $s1, salida
bne $a0, $s2, NoAlineada
la $a0, mis1
li $v0, 4
syscall
mfc0 $a0, $14 # $a0 <= EPC, donde ha ocurrido la excepción
li $v0, 34
syscall # Escribimos EPC en hexadecimal
Desbordo:
la $a0, mis2
li $v0, 4
syscall
mfc0 $a0, $14 # $a0 <= EPC, donde ha ocurrido la excepción
li $v0, 34
syscall # Escribimos EPC en hexadecimal
salida:
la $a0, mis3
li $v0, 4
syscall
NoAlineada:
la $a0, mis4
li $v0, 4
```

```
Line: 7 Column: 1  Show Line Numbers
Mars Messages  Run I/O
Clear
Excepción desbordamiento ocurrida en la dirección: 0x0040000c
En cualquier caso continuamos el programa...

Excepción de dirección no alineada
```

Ejercicios optativos

Práctica 10



Haz diversas pruebas hasta que comprendas el funcionamiento del programa. Tendrás que teclear un carácter dentro del área de la ventana inferior.

Comprobarás el carácter introducido mirando lo registro $v0

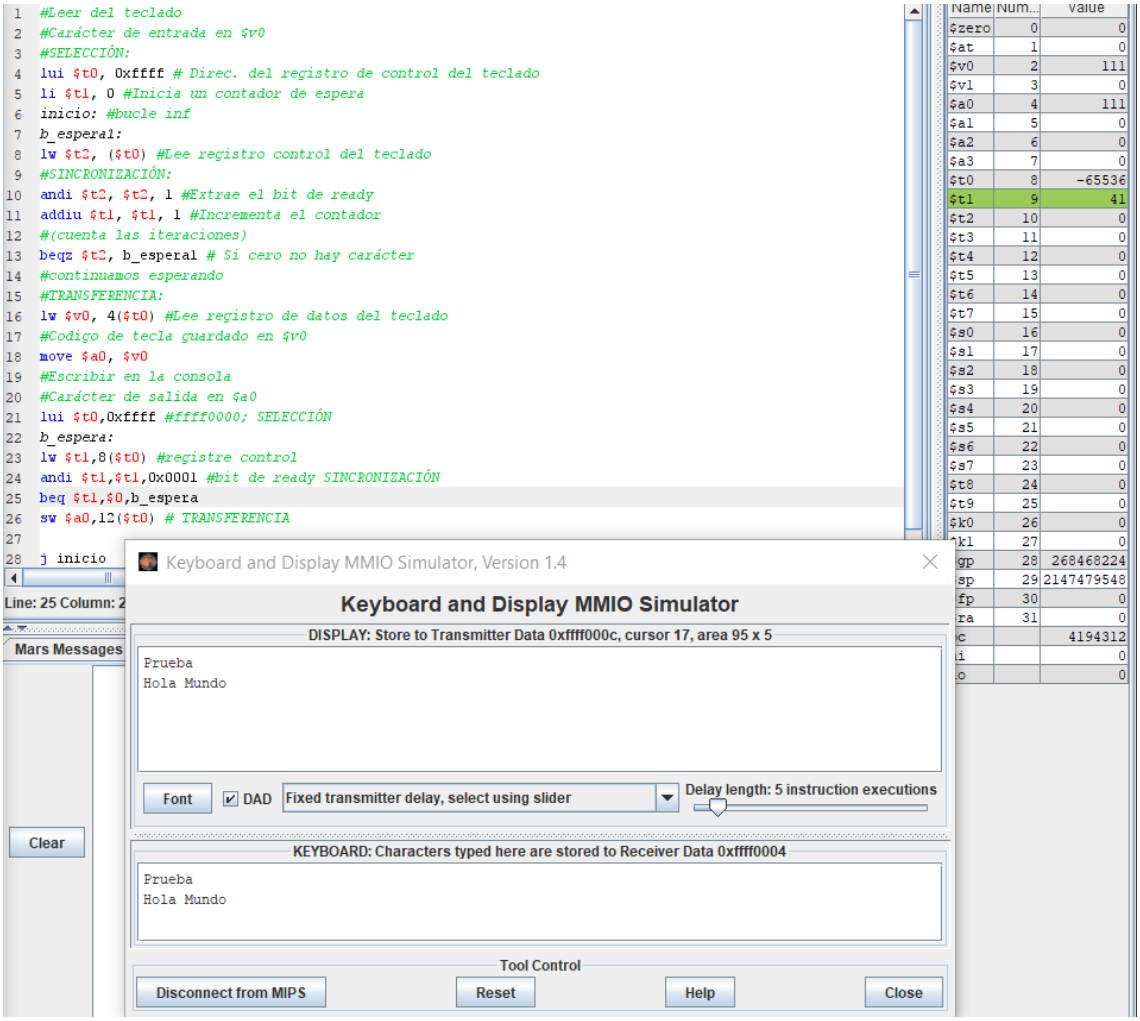
➔ Si introducimos d, $v0 vale 100.

Observa el contador para comprobar la diferencia de velocidad del programa y el usuario

➔ Aumenta de 1 en uno el valor de $t1, hasta que escribimos una tecla, si escribimos.

Elimina momentáneamente la instrucción que lee el carácter del registro de datos del teclado y comprueba que el bit de ready permanece con el valor 1. Sólo pasará a cero si el programa lee el carácter.

➔ Sigue aumentando a pesar de eliminar la instrucción de la línea 10.

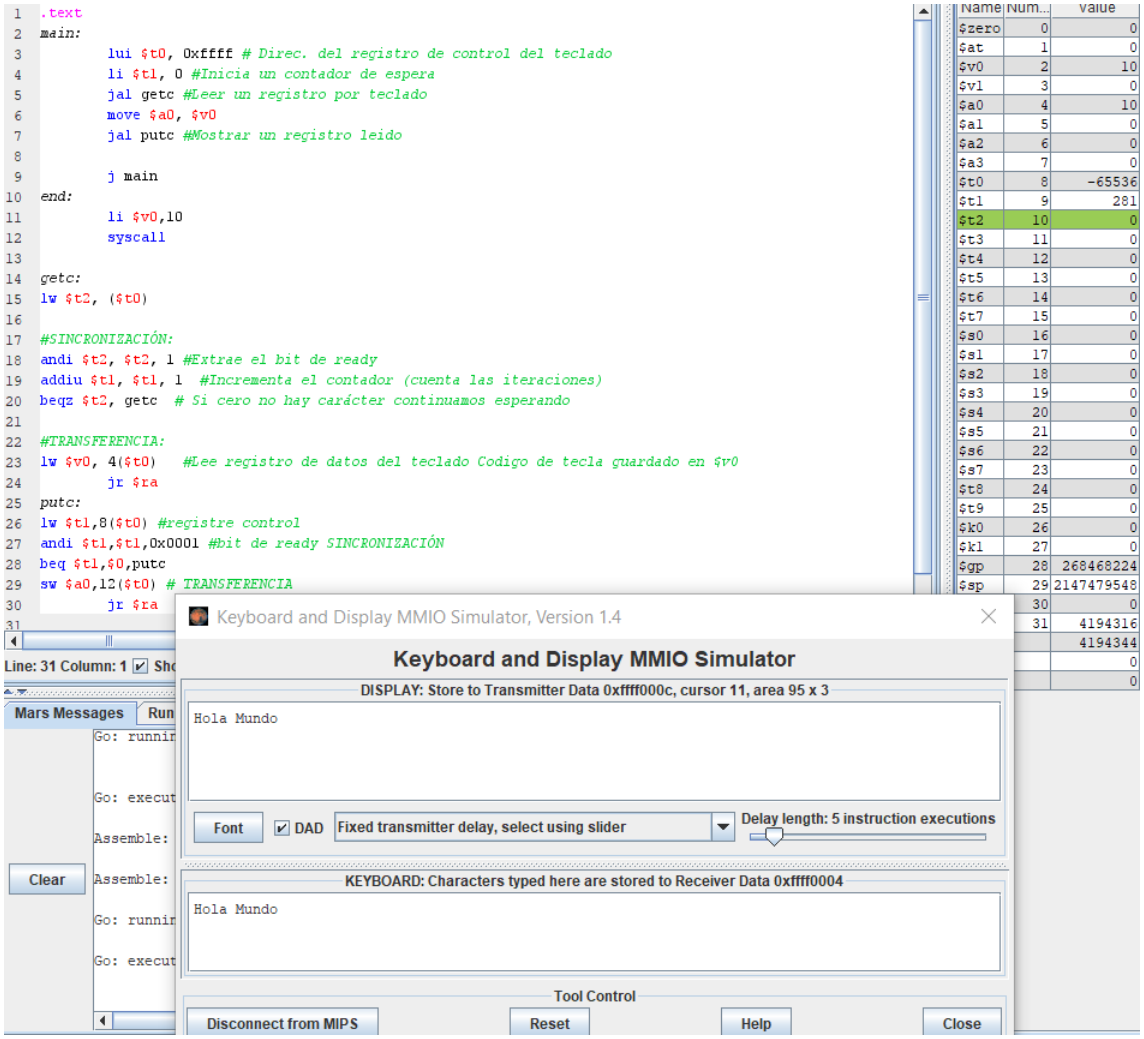


¿Cómo cambiaría el código si sustituyéramos la primera instrucción por le 0xffff0008?

➔ Da error

Añade el fragmento al programa de leer de la consola y comprueba su funcionamiento, pero desactivad previamente la casilla DAD en la herramienta del MIPS

➔ Al desactivar DAD nos aseguramos de que cuando introduzcas algo lo detecte al momento.



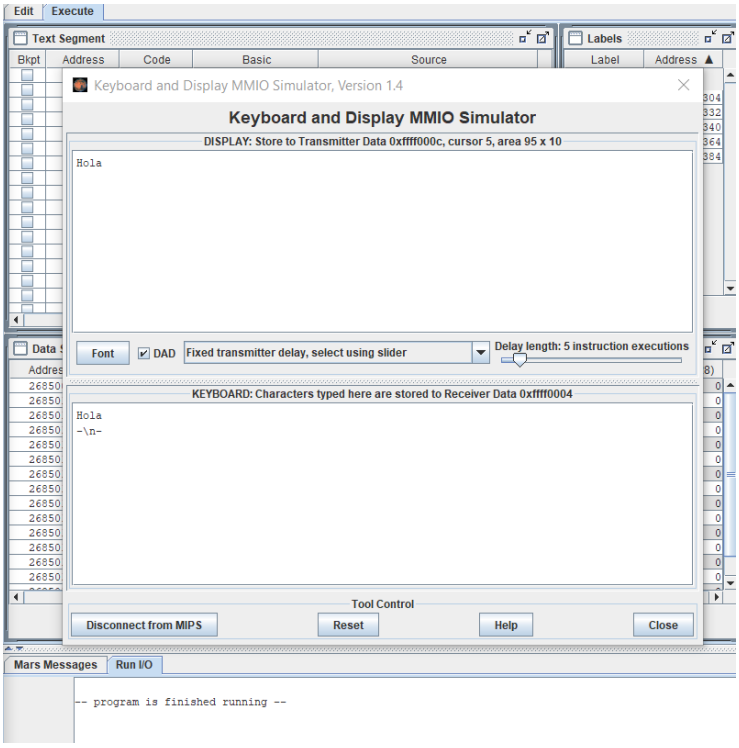
Probad su funcionamiento y comprobad cómo varían los contenidos de los registros y la memoria MMIO.

➔ Como podemos observar, con este programa todo lo que escribamos por teclado se escribe directamente en el display y no para nunca el bucle en el que lee el registro introducido por teclado.

```
.text
main:
        lui $t0, 0xffff # Direc. del registro de control del teclado
        li $t1, 0 #Inicia un contador de espera
        jal getc #Leer un registro por teclado
        move $a0, $v0
        jal putc #Mostrar un registro leido
        j echo
        j main
end:
        li $v0,10
        syscall

getc:
lw $t2, ($t0)

#SINCRONIZACIÓN:
andi $t2, $t2, 1 #Extrae el bit de ready
addiu $t1, $t1, 1  #Incrementa el contador (cuenta las iteraciones)
beqz $t2, getc  # Si cero no hay carácter continuamos esperando

#TRANSFERENCIA:
lw $v0, 4($t0)   #Lee registro de datos del teclado Codigo de tecla guardado en $v0
        jr $ra
putc:
lw $t1,8($t0) #registre control
andi $t1,$t1,0x0001 #bit de ready SINCRONIZACIÓN
beq $t1,$0,putc
sw $a0,12($t0) # TRANSFERENCIA
        jr $ra
echo:
        la $s1, '\n'
        beq $a0, $s1, end
        j main

```

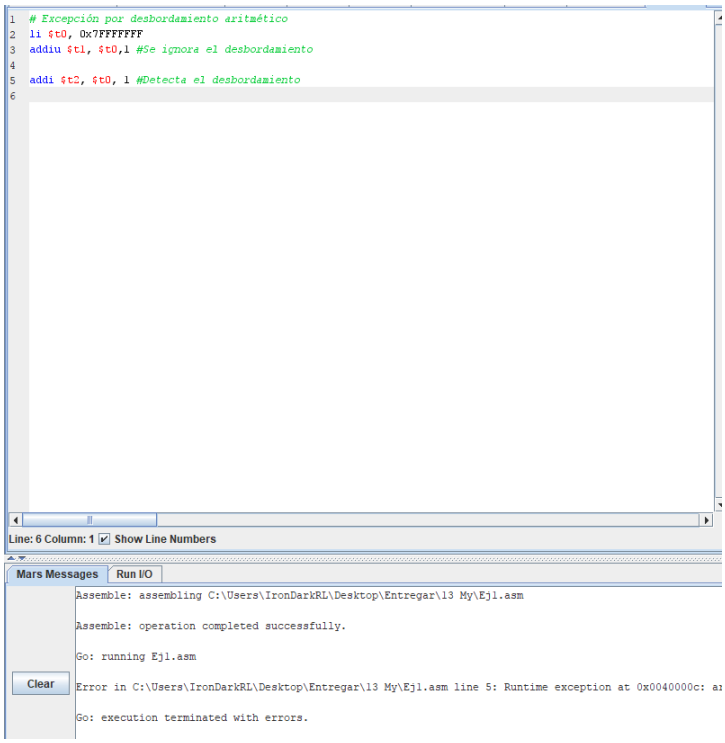
Iterad el código anterior hasta que el carácter introducido sea un salto de línea (‘/’).

Como podemos observar al llamar a echo e introducir un salto de línea, el programa finaliza y deja de copiar lo introducido por teclado en el display.



Ejercicios optativos

Práctica 10



Observa el código. ¿Qué instrucción causará la excepción?

➔ La instrucción addi $t2, $t0, 1

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

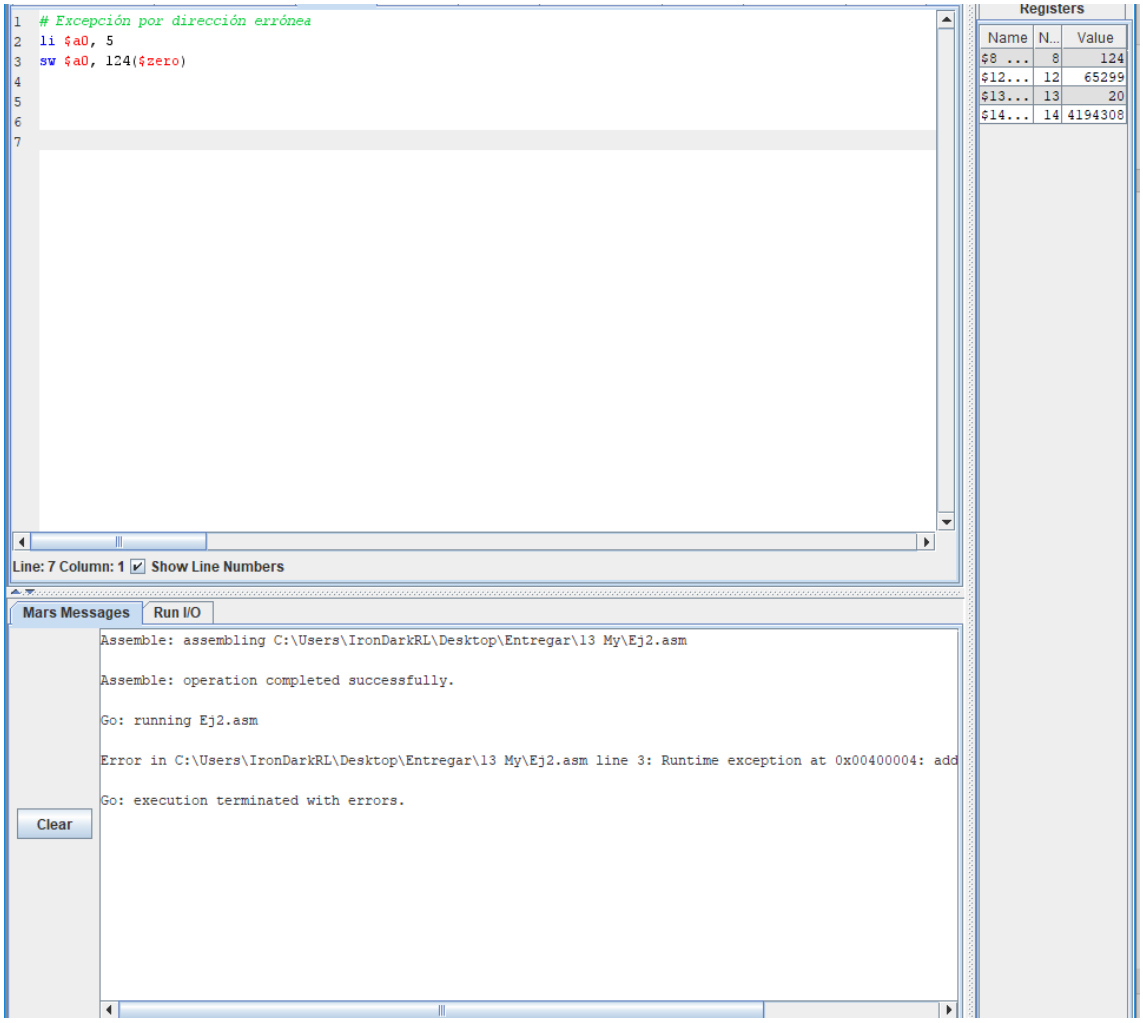
➔ En Coproc 0 ha cambiado $12 de ff11 a ff13(No se permite generar excepciones por el usuario)

¿Cuáles son los valores de los registros del coprocesador 0 antes y después de producirse la excepción?

➔ FF11

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

➔ FF13, no se permite al usuario generar excepciones y podemos ver que es overflow aritmetico si pasamos a binario. Además $14 guarda la dirección: 40000c

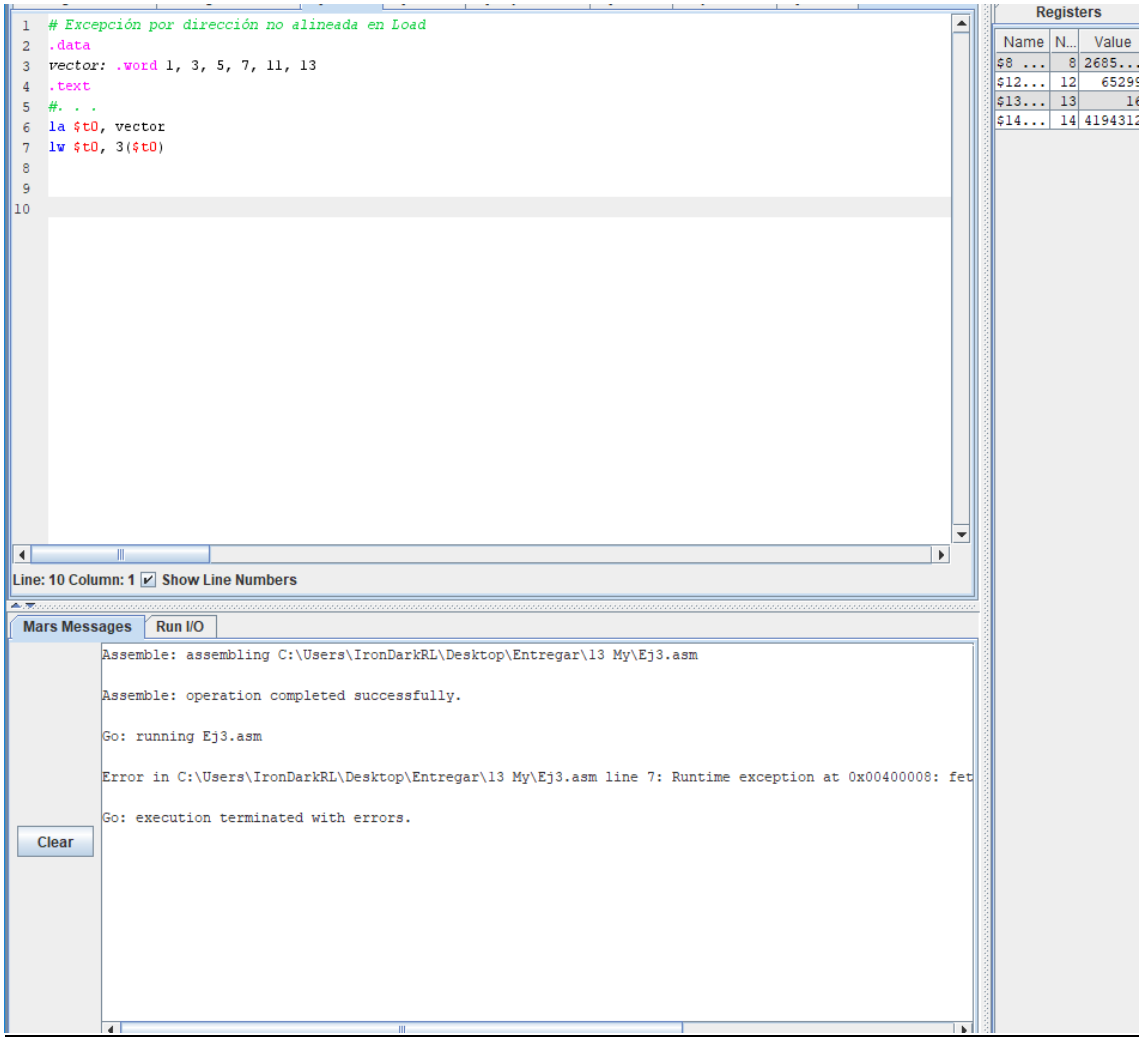


¿Cuáles son los valores de los registros del coprocesador 0 antes y después de producirse la excepción?

➔ ANTES: $8= 0 ; $12= FF11; $13= 0; $14= 0
➔ DESPUES: $8= 7C ; $12= FF13; $13= 14 ; $14= 400004

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

➔ Dirección errónea



¿Cuáles son los valores de los registros del coprocesador 0 antes y después de producirse la excepción?

➔ ANTES: $8= 0 ; $12= FF11; $13= 0; $14= 0
➔ DESPUES: $8= 10010003 ; $12= FF13; $13= 14 ; $14= 400004

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

➔ Dirección no alineada en el Load

Supón que el contenido del registro Cause ($13) tiene los siguientes valores

después de haberse producido una excepción. Rellena la tabla indicando cual ha sido la causa que ha provocado la excepción en cada caso:

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

Cause		*Fuente de la excepción*
0x00000000	→	Interrupción (hardware)
0x00000020	→	Excepción syscall
0x00000024	→	Excepción de punto de ruptura
0x00000028	→	Excepción instrucción reservada
0x00000030	→	Excepción por desbordamiento

Código más abajo

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

➔ El programa realiza un tratamiento de excepciones, es decir, cuando da error muestra un mensaje editado por el que ha hecho el programa.

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

➔ La secuencia en la que se copia en $s0 y $s1, los códigos de desbordamiento y dirección errónea y las instrucciones beq y bne que comparan el resultado obtenido para mostrar el mensaje deseado. (líneas 23-26)

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

➔ Muestra el mensaje Excepció desbordamiento ocurrida en la dirección y seguidamente imprime la dirección de la instrucción que ha generado ese error

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

➔ Las instrucciones que se encargan de salvar los registros que se utilizan (líneas 16-21)

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

➔ Entonces el programa no funcionaría correctamente

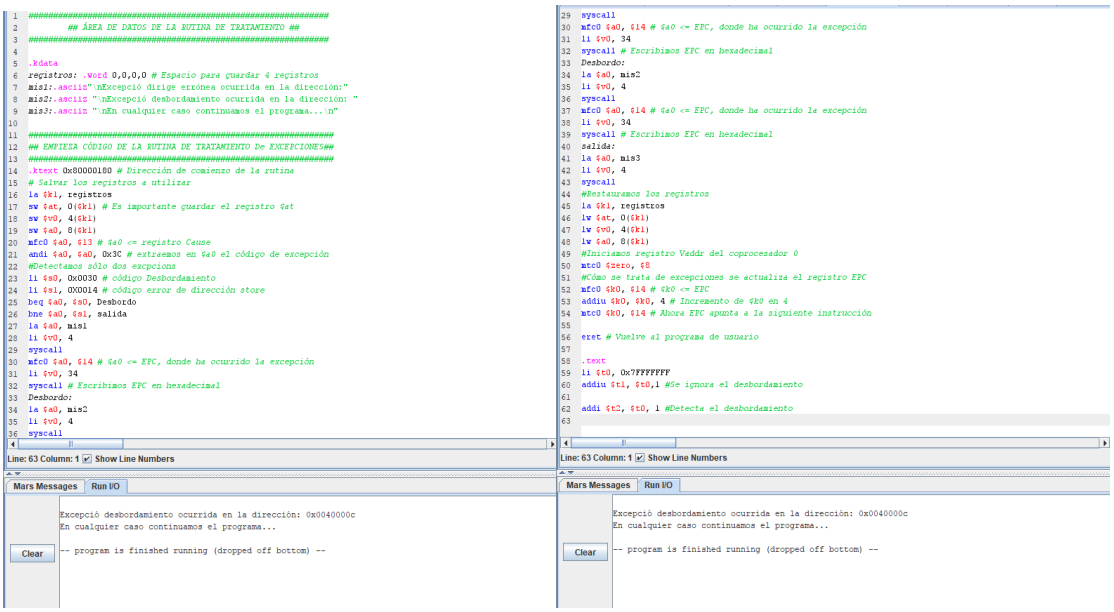
¿En qué casos se han utilizado los registres $k0 y $k1?

➔ Para actualizar el registro EPC y para restaurar/guardar los registros

¿Por qué otra instrucción podrías sustituir la instrucción eret? ¿Cómo quedaría?

➔ jr $k0

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



```
# Reserva de espacio para guardar registros en kdata
.kdata
contexto: .word 0,0,0,0 # espacio para alojar cuatro registros
.ktext 0x80000180 # Dirección de comienzo de la rutina
# Guardar registros a utilizar en la rutina.
la $k1, contexto
sw $at, 0($k1) # Guardamos $at
sw $t0, 4($k1)
sw $v0, 8($k1)
sw $a0, 12($k1)
#Comprobación de si se trata de una interrupción
mfc0 $k0, $13 # Registro Cause
srl $a0, $k0, 2 # Extraemos campo del código
andi $a0, $a0, 0x1f
bne $a0, $zero, acabamos # Sólo procesamos aquí E/S
#Tratamiento de la interrupción
li $t0, 0xffff0000
lb $a0, 4($t0) # Lee carácter del teclado
# Por ejemplo:
# Escribe en la consola del MARS el carácter leído
li $v0, 11
syscall
# Antes de acabar se podría dejar todo iniciado:
acabamos: mtc0 $0, $13 # Iniciar registro Cause
mfc0 $k0, $12 # Leer registre Status
andi $k0, 0xfffd # Iniciar bit de excepción
ori $k0, 0x11 # Habilitar interrupciones
mtc0 $k0, $12 # reescribir registre Startus
# Restaurar registros
lw $at, 0($k1) # Recupero $at
lw $t0, 4($k1)
lw $v0, 8($k1)
lw $a0, 12($k1)
# Devolver en el programa de usuario
eret
```

```
acabamos: mtc0 $0, $13 # Iniciar registro Cause
mfc0 $k0, $12 # Leer registre Status
andi $k0, 0xfffd # Iniciar bit de excepción
ori $k0, 0x11 # Habilitar interrupciones
mtc0 $k0, $12 # reescribir registre Startus
# Restaurar registros
lw $at, 0($k1) # Recupero $at
lw $t0, 4($k1)
lw $v0, 8($k1)
lw $a0, 12($k1)
# Devolver en el programa de usuario
eret

.data
        mensaje: .asciiz "Pulsa teclas: \n"
.text
lui $t0,0xffff # Dirige del registro de control
lw $t1,0($t0) # Registre de control del receptor
ori $t1,$t1,0x0002 # Habilitar interrupciones del teclado
sw $t1,0($t0) # Actualizamos registro de control

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

la $a0, mensaje
li $v0, 4
syscall

inicio:
j inicio
```

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

➔ Proviene de mfc0 $k0, $13 y usa un bne para comprobar si se trata o no de una excepción

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

➔ mfc0 $k0, $13 # Registro Cause

srl $a0, $k0, 2

andi $a0, $a0, 0x1f

bne $a0, $zero, acabamos

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

➔ En el tratamiento de excepciones va aumentando EPC y podemos ver donde se ha producido el error

¿Podrían incluirse los dos tratamientos en una misma rutina?

➔ Sí teniendo en cuenta que el epc no se incrementa

Ej. De ejecución: