

MEMORIA

Índice

1. PRÁCTICA 10 (p.2-4)
2. PRÁCTICA 11 (p.5-6)
3. Ejercicios optativos (p.7-17)
 - Optativos Práctica 10 (p.7-10)
 - Optativos Práctica 11 (p.11-17)

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.

The screenshot shows a MIPS assembly code editor on the left and a 'Keyboard and Display MMIO Simulator' window on the right. The assembly code implements a program that reads characters from the keyboard, converts them to uppercase, and prints them. The keyboard simulator window shows the text 'HOLAMUNDO' on the display and 'hola mundo' on the keyboard input.

```
1 .text
2 main:
3     lui $t0, 0xffff # Direc. del registro de control del teclado
4     li $t1, 0 #Inicia un contador de espera
5     jal getc #Leer un registro por teclado
6     jal caps
7     move $a0, $v0
8     jal putc #Mostrar un registro leído
9     j echo
10    j main
11 end:
12    li $v0, 10
13    syscall
14
15 getc:
16    lw $t2, ($t0)
17
18 #SINCRONIZACIÓN:
19 andi $t2, $t2, 1 #Extrae el bit de ready
20 addiu $t1, $t1, 1 #Incrementa el contador (cuenta las iteraciones)
21 beqz $t2, getc # Si cero no hay carácter continuamos esperando
22
23 #TRANSFERENCIA:
24 lw $v0, 4($t0) #Lee registro de datos del teclado Código de tecla guardado en $v0
25 jr $ra
26
27 putc:
28 lw $t1, 8($t0) #registre control
29 andi $t1, $t1, 0x0001 #bit de ready SINCRONIZACIÓN
30 beq $t1, $0, putc
31 sw $a0, 12($t0) # TRANSFERENCIA
32 jr $ra
33
34 echo:
35 la $s1, '\n'
36 beq $a0, $s1, end
37 j main
38
39 caps:
40 #32 de diferencia
41 add $v0, $v0, -32
42 jr $ra
```

Keyboard and Display MMIO Simulator, Version 1.4

DISPLAY: Store to Transmitter Data 0xffff000c, cursor 10, area 95 x 0

HOLAMUNDO

Font ☒ DAD Fixed transmitter delay, select using slider Delay length: 5 instruction executions

KEYBOARD: Characters typed here are stored to Receiver Data 0xffff0004

hola mundo

Tool Control

Disconnect from MIPS Reset Help Close

Mars Messages Run I/O

Go: running Entregable5.asm

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 creamos.

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 registro 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)

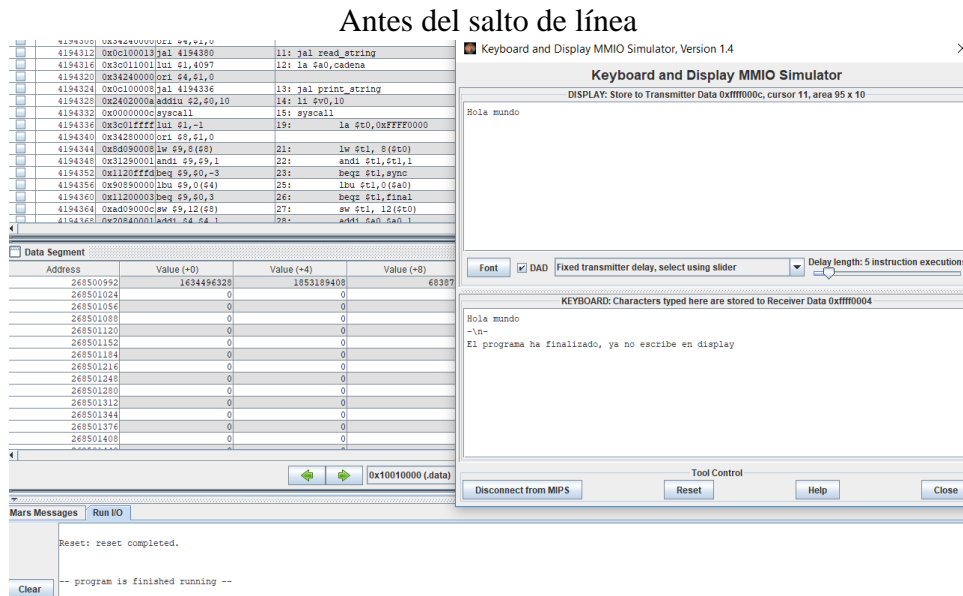
    move $a0, $v0
    sb $a0, 0($t5)
    addi $t4, $t4, 1
    bne $a0, 10, loop # El salto de linea equivale al 10
    #Ha introducido \n, acabamos el bucle
    jr $ra
```

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.

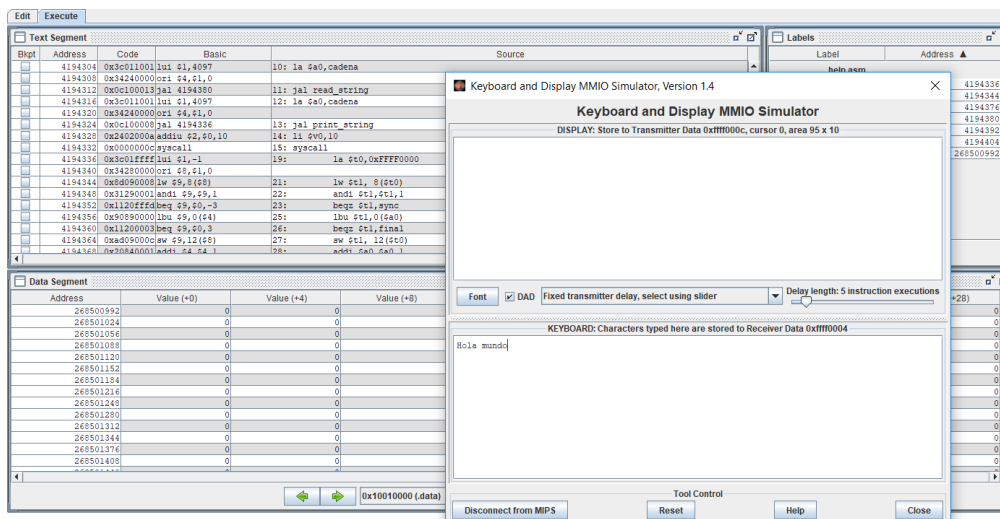
Una vez se ha introducido un registro, usamos la instrucción **sb** para guardar en \$t5 el valor de \$a0 (El valor que había en \$v0, el cual era el registro introducido por teclado)

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:



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.

```
1 .data
2 registros: .word 0,0,0,0 # Espacio para guardar 4 registros
3 mis1: .ascii "\nExcepción dirige error en la dirección:"
4 mis2: .ascii "\nExcepción desbordamiento ocurrida en la dirección:"
5 mis3: .ascii "\nEn cualquier caso continuamos el programa...\n"
6 mis4: .ascii "\nExcepción de dirección no alineada"
7 .text 0x80000180 # Dirección de comienzo de la rutina
8 # Salvar los registros a utilizar
9 la $k1, registros
10 sw $at, 0($k1) # Es importante guardar el registro $at
11 sw $v0, 4($k1)
12 sw $a0, 8($k1)
13 mfc0 $a0, $13 # $a0 <= registro Cause
14 andi $a0, $a0, 0xC # extraemos en $a0 el código de excepción
15 #Detectamos sólo dos excepciones
16 li $a0, 0x0030 # código Desbordamiento
17 li $t1, 0x0014 # código error de dirección store
18 li $t2, 0x00000054 #No alineada
19 andi $t2, $t2, 100
20 beq $a0, $t0, Desborde
21 hne $a0, $t1, salida
22 hne $a0, $t2, Noalineada
23 la $a0, mis1
24 li $v0, 4
25 syscall
26 mfc0 $a0, $14 # $a0 <= EPC, donde ha ocurrido la excepción
27 li $v0, 34
28 syscall # Escribimos EPC en hexadecimal
29
30 Desborde:
31 la $a0, mis2
32 li $v0, 4
33 syscall
34 mfc0 $a0, $14 # $a0 <= EPC, donde ha ocurrido la excepción
35 li $v0, 34
36 syscall # Escribimos EPC en hexadecimal
37 salida:
38 la $a0, mis3
39 li $v0, 4
40 syscall
41 Noalineada:
42 la $a0, mis4
43 li $v0, 4
44 syscall
45 #Restauramos los registros
46 la $k1, registros
47 lw $at, 0($k1)
48 lw $v0, 4($k1)
49 lw $a0, 8($k1)
50 mtc0 $a0, $13
51 #Cómo se trata de excepciones se actualiza el registro EPC
52 mtc0 $k0, $14 # $k0 <= EPC
53 addiu $k0, $k0, 4 # Incremento de $k0 en 4
54 mtc0 $k0, $14 # Ahora EPC apunta a la siguiente instrucción
55
56 ret # Vuelve al programa de usuario
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
```

Mars Messages Run I/O

Assemble: assembling C:\Users\IronDark\I\I\Desktop\Entregar\13 My\Entregable2.asm
Assemble: operation completed successfully.
Go: running Entregable2.asm
Go: execution terminated by null instruction.

Clear

```
25 syscall
26 mfc0 $a0, $14 # $a0 <= EPC, donde ha ocurrido la excepción
27 li $v0, 34
28 syscall # Escribimos EPC en hexadecimal
29
30 Desborde:
31 la $a0, mis2
32 li $v0, 4
33 syscall
34 mfc0 $a0, $14 # $a0 <= EPC, donde ha ocurrido la excepción
35 li $v0, 34
36 syscall # Escribimos EPC en hexadecimal
37 salida:
38 la $a0, mis3
39 li $v0, 4
40 syscall
41 Noalineada:
42 la $a0, mis4
43 li $v0, 4
44 syscall
45 #Restauramos los registros
46 la $k1, registros
47 lw $at, 0($k1)
48 lw $v0, 4($k1)
49 lw $a0, 8($k1)
50 mtc0 $a0, $13
51 #Cómo se trata de excepciones se actualiza el registro EPC
52 mtc0 $k0, $14 # $k0 <= EPC
53 addiu $k0, $k0, 4 # Incremento de $k0 en 4
54 mtc0 $k0, $14 # Ahora EPC apunta a la siguiente instrucción
55
56 ret # Vuelve al programa de usuario
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
```

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 uso de la instrucción bne para comprobar qué error tenemos que tratar.

```
1 .kdata
2 registros: .word 0,0,0,0 # Espacio para guardar 4 registros
3 mis1:.asciiz "\nExcepción dirige errónea ocurrida en la dirección:"
4 mis2:.asciiz "\nExcepción desbordamiento ocurrida en la dirección: "
5 mis3:.asciiz "\nEn cualquier caso continuamos el programa...\n"
6 mis4: .asciiz "\nExcepción de dirección no alineada"
7
8 .ktext 0x80000180 # Dirección de comienzo de la rutina
9 # Salvar los registros a utilizar
10 la $k1, registros
11 sw $at, 0($k1) # Es importante guardar el registro $at
12 sw $v0, 4($k1)
13 sw $a0, 8($k1)
14 mfc0 $a0, $13 # $a0 <= registro Cause
15 andi $a0, $a0, 0x3C # extraemos en $a0 el código de excepción
16 #Detectamos sólo dos excpciones
17 li $s0, 0x0030 # código Desbordamiento
18 li $s1, 0x0014 # código error de dirección store
19 li $s2, 0x00000024 #No alineada
20 beq $a0, $s0, Desbordo
21 bne $a0, $s1, salida
22 bne $a0, $s2, NoAlineada
23 la $a0, mis1
24 li $v0, 4
25 syscall
26 mfc0 $a0, $14 # $a0 <= EPC, donde ha ocurrido la excepción
27 li $v0, 34
28 syscall # Escribimos EPC en hexadecimal
29 Desbordo:
30 la $a0, mis2
31 li $v0, 4
32 syscall
33 mfc0 $a0, $14 # $a0 <= EPC, donde ha ocurrido la excepción
34 li $v0, 34
35 syscall # Escribimos EPC en hexadecimal
36 salida:
37 la $a0, mis3
38 li $v0, 4
39 syscall
40 NoAlineada:
41 la $a0, mis4
42 li $v0, 4
```

line: 7 Column: 1 ☒ Show Line Numbers

Mars Messages Run I/O

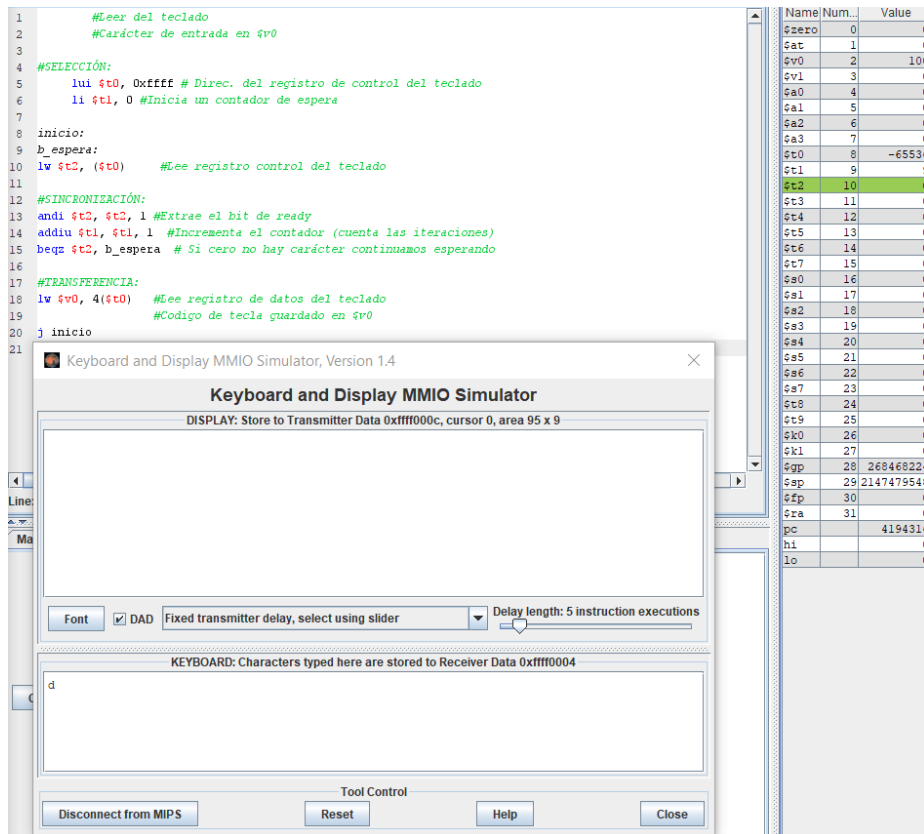
Excepción desbordamiento ocurrida en la dirección: 0x0040000c
En cualquier caso continuamos el programa...

Clear

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.

The screenshot displays the MIPS assembly code in the main editor and a register window on the right. The assembly code is as follows:

```
1 #Leer del teclado
2 #Carácter de entrada en $v0
3 #SELECCIÓN:
4 lui $t0, 0xffff # Direc. del registro de control del teclado
5 li $t1, 0 #Inicia un contador de espera
6 inicio: #bucle inf
7 b_espera:
8 lw $t2, ($t0) #Lee registro control del teclado
9 #SINCRONIZACIÓN:
10 andi $t2, $t2, 1 #Extrae el bit de ready
11 addiu $t1, $t1, 1 #Incrementa el contador
12 #(cuenta las iteraciones)
13 beqz $t2, b_espera # Si cero no hay carácter
14 #continuamos esperando
15 #TRANSFERENCIA:
16 lw $v0, 4($t0) #Lee registro de datos del teclado
17 #Codigo de tecla guardado en $v0
18 move $a0, $v0
19 #Escribir en la consola
20 #Carácter de salida en $a0
21 lui $t0, 0xffff #ffff0000; SELECCIÓN
22 b_espera:
23 lw $t1, 8($t0) #registre control
24 andi $t1, $t1, 0x0001 #bit de ready SINCRONIZACIÓN
25 beq $t1, $0, b_espera
26 sw $a0, 12($t0) # TRANSFERENCIA
27
28 j inicio
```

The register window on the right shows the following values:

Name	Num.	value
\$zero	0	0
\$at	1	0
\$v0	2	111
\$v1	3	0
\$a0	4	111
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	-65536
\$t1	9	41
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	0
\$s1	17	0
\$s2	18	0
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$t8	24	0
\$t9	25	0
\$k0	26	0
\$k1	27	0
gp	28	268468224
sp	29	2147479548
fp	30	0
ra	31	0
c		4194312
i		0
o		0

The Keyboard and Display MMIO Simulator window is open, showing the DISPLAY: Store to Transmitter Data 0xffff000c, cursor 17, area 95 x 5. The DISPLAY area contains the text "Prueba" and "Hola Mundo". The KEYBOARD area contains the text "Prueba" and "Hola Mundo". The DAD checkbox is checked, and the Delay length is set to 5 instruction executions. The Tool Control buttons are Disconnect from MIPS, Reset, Help, and Close.

¿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.


```

1 .text
2 main:
3     lui $t0, 0xffff # Direc. del registro de control del teclado
4     li $t1, 0 #Inicia un contador de espera
5     jal getc #Leer un registro por teclado
6     move $a0, $v0
7     jal putc #Mostrar un registro leído
8
9     j main
10 end:
11     li $v0, 10
12     syscall
13
14 getc:
15     lw $t2, 4($t0)
16
17 #SINCRONIZACIÓN:
18     andi $t2, $t2, 1 #Extrae el bit de ready
19     addiu $t1, $t1, 1 #Incrementa el contador (cuenta las iteraciones)
20     beqz $t2, getc # Si cero no hay carácter continuamos esperando
21
22 #TRANSFERENCIA:
23     lw $v0, 4($t0) #Lee registro de datos del teclado Código de tecla guardado en $v0
24     jr $ra
25
26 putc:
27     lw $t1, 8($t0) #registre control
28     andi $t1, $t1, 0x0001 #bit de ready SINCRONIZACIÓN
29     beq $t1, $0, putc
30     sw $a0, 12($t0) # TRANSFERENCIA
31     jr $ra

```

Name	Num...	value
\$zero	0	0
\$at	1	0
\$v0	2	10
\$v1	3	0
\$a0	4	10
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	-65536
\$t1	9	281
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	0
\$s1	17	0
\$s2	18	0
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$s8	24	0
\$s9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$30	30	0
\$31	31	4194316
		4194344
		0
		0

Keyboard and Display MMIO Simulator, Version 1.4

DISPLAY: Store to Transmitter Data 0xffff000c, cursor 11, area 95 x 3

Hola Mundo

Font ☒ DAD Fixed transmitter delay, select using slider Delay length: 5 instruction executions

KEYBOARD: Characters typed here are stored to Receiver Data 0xffff0004

Hola Mundo

Tool Control

Disconnect from MIPS Reset Help Close

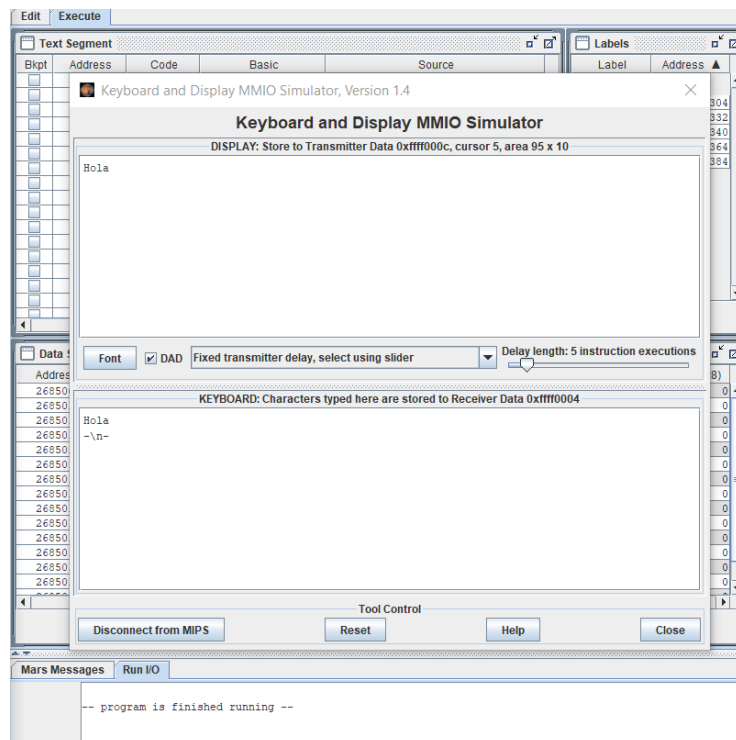
Probad su funcionamiento y comprobad cómo varían los contenidos de los registros y la memoria MMIO.

- ➔ 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.

```
1 .text
2 main:
3     lui $t0, 0xffff # Direc. del registro de control del teclado
4     li $t1, 0 #Inicia un contador de espera
5     jal getc #Lee un registro por teclado
6     move $a0, $v0
7     jal putc #Mostrar un registro leído
8     j echo
9     j main
10 end:
11     li $v0, 10
12     syscall
13
14 getc:
15     lw $t2, ($t0)
16
17 #SINCRONIZACIÓN:
18     andi $t2, $t2, 1 #Extrae el bit de ready
19     addiu $t1, $t1, 1 #Incrementa el contador (cuenta las iteraciones)
20     beqz $t2, getc # Si cero no hay carácter continuamos esperando
21
22 #TRANSFERENCIA:
23     lw $v0, 4($t0) #Lee registro de datos del teclado Código de tecla guardado en $v0
24     jr $ra
25
26 putc:
27     lw $t1, 8($t0) #registre control
28     andi $t1, $t1, 0x0001 #bit de ready SINCRONIZACIÓN
29     beq $t1, $0, putc
30     sw $a0, 12($t0) # TRANSFERENCIA
31     jr $ra
32
33 echo:
34     la $s1, '\n'
35     beq $a0, $s1, end
36     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

```
1 # Excepción por desbordamiento aritmético
2 li $t0, 0x7FFFFFFF
3 addiu $t1, $t0, 1 #Se ignora el desbordamiento
4
5 addi $t2, $t0, 1 #Detecta el desbordamiento
6
```

Line: 6 Column: 1 ☒ Show Line Numbers

Mars Messages Run I/O

Assemble: assembling C:\Users\IronDarkRL\Desktop\Entregar\13 My\Ejl.asm
Assemble: operation completed successfully.
Go: running Ej1.asm
Error in C:\Users\IronDarkRL\Desktop\Entregar\13 My\Ejl.asm line 5: Runtime exception at 0x0040000c: ar
Go: execution terminated with errors.

Clear

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.

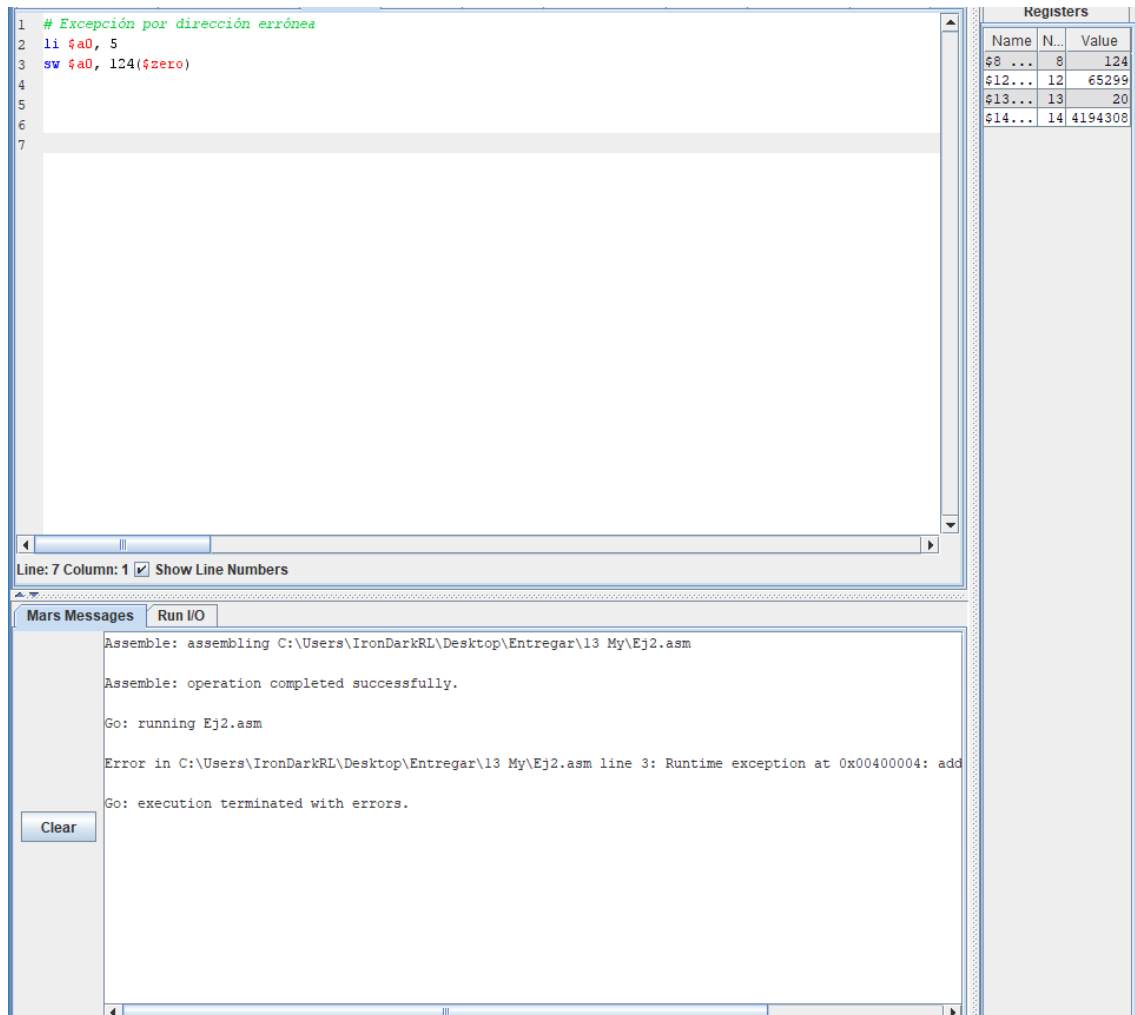
➔ En Coproc 0 ha cambiado \$12 de ff11 a ff13(No se permite generar excepciones por el usuario)

¿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

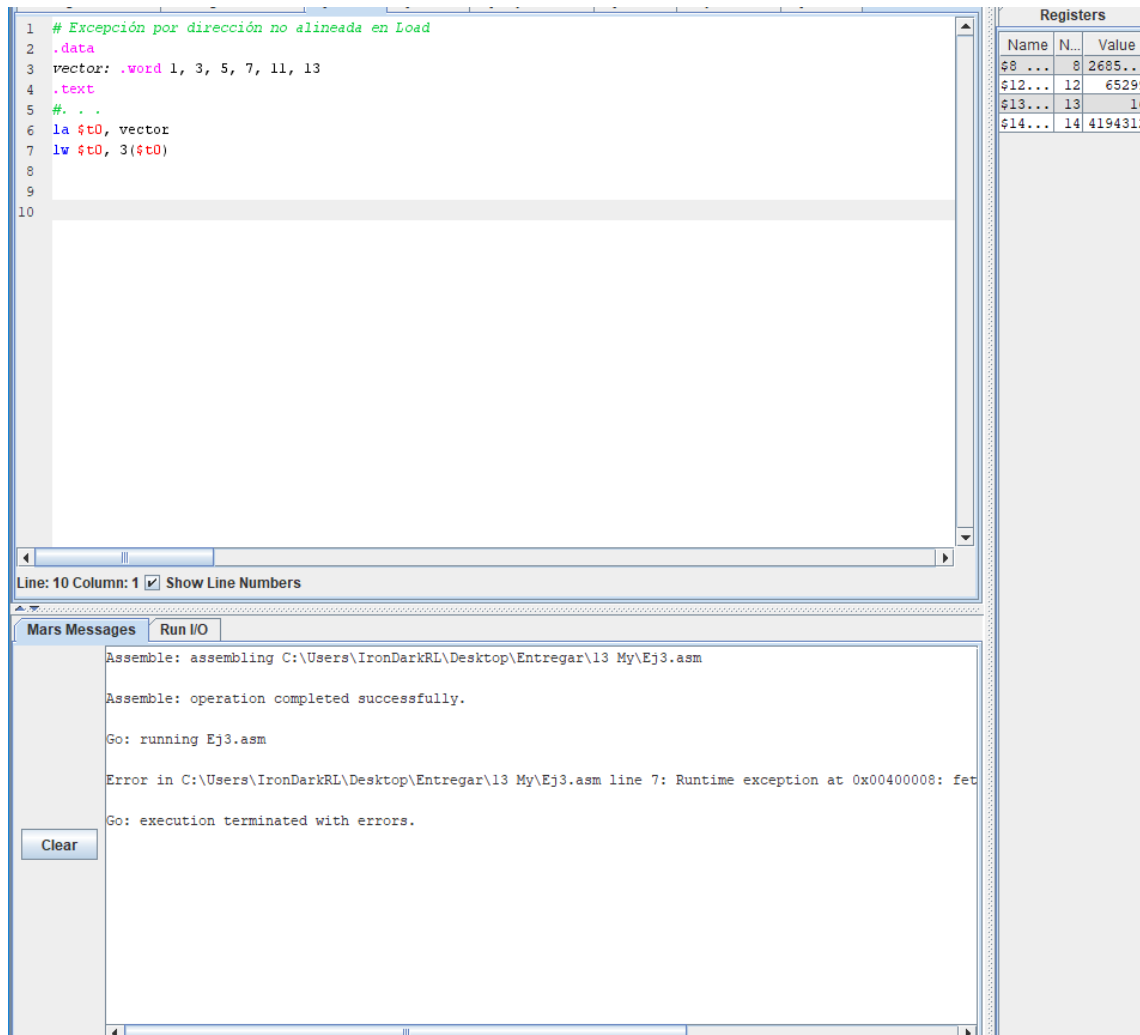


¿Cuáles son los valores de los registros del coprocesador 0 antes y después de producirse la excepción?

- ➔ 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

<i>Cause</i>		<i>Fuente de la excepción</i>
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 Excepción 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 registros \$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.

The image displays two side-by-side screenshots of the Mars MIPS simulator, illustrating the execution of an exception handling routine.

Left Screenshot (Assembly Code):

```
1 #####
2 ## ÁREA DE DATOS DE LA RUTINA DE TRATAMIENTO ##
3 #####
4
5 .data
6 registros: .word 0,0,0,0 # Espacio para guardar 4 registros
7 mis1:.asciiz"Excepción dirigida errónea ocurrida en la dirección:"
8 mis2:.asciiz"Excepción de desbordamiento ocurrida en la dirección:"
9 mis3:.asciiz"En cualquier caso continuamos el programa..."
10
11 #####
12 ## EMPieza CódIGO DE LA RUTINA DE TRATAMIENTO De EXCEPCIONES##
13 #####
14 .text 0x00000180 # Dirección de comienzo de la rutina
15 # Salvar los registros a utilizar
16 la $t1, registros
17 sw $t0, 0($t1) # Es importante guardar el registro $t0
18 sw $t0, 4($t1)
19 sw $t0, 8($t1)
20 mfc0 $a0, $t0 # $a0 <- registro Cause
21 andi $a0, $a0, 0x3C # extraemos en $a0 el código de excepción
22 #Detectamos sólo dos excepciones
23 li $t0, 0x0030 # código Desbordamiento
24 li $t1, 0x00014 # código error de dirección store
25 beq $a0, $t0, Desbordo
26 hne $a0, $t1, salida
27 la $t0, mis1
28 li $v0, 4
29 syscall
30 mfc0 $a0, $t1 # $a0 <- EPC, donde ha ocurrido la excepción
31 li $v0, 34
32 syscall # Escribimos EPC en hexadecimal
33 Desbordo:
34 la $t0, mis2
35 li $v0, 4
36 syscall
```

Right Screenshot (Assembly Code):

```
29 syscall
30 mfc0 $a0, $t1 # $a0 <- EPC, donde ha ocurrido la excepción
31 li $v0, 34
32 syscall # Escribimos EPC en hexadecimal
33 Desbordo:
34 la $t0, mis2
35 li $v0, 4
36 syscall
37 mfc0 $a0, $t1 # $a0 <- EPC, donde ha ocurrido la excepción
38 li $v0, 34
39 syscall # Escribimos EPC en hexadecimal
40 salida:
41 la $t0, mis3
42 li $v0, 4
43 syscall
44 #Restauramos los registros
45 la $t1, registros
46 lw $t0, 0($t1)
47 lw $t0, 4($t1)
48 lw $t0, 8($t1)
49 #Iniciamos registro Vaddr del coprocesador 0
50 mtc0 $zero, $t0
51 #Cómo se trata de excepciones se actualiza el registro EPC
52 mtc0 $a0, $t0 # $t0 <- EPC
53 addiu $t0, $t0, 4 # Incremento de $t0 en 4
54 mtc0 $t0, $t0 # Ahora EPC apunta a la siguiente instrucción
55
56 eret # Vuelve al programa de usuario
57
58 .text
59 li $t0, 0x7FFFFFFF
60 addiu $t1, $t0, 1 #Se ignora el desbordamiento
61
62 addi $t2, $t0, 1 #Detecta el desbordamiento
63
```

Execution Results (Mars Messages):

Left Panel:

```
Line: 63 Column: 1 [x] Show Line Numbers
Mars Messages Run IO
Excepción de desbordamiento ocurrida en la dirección: 0x0040000c
En cualquier caso continuamos el programa...
-- program is finished running (dropped off bottom) --
```

Right Panel:

```
Line: 63 Column: 1 [x] Show Line Numbers
Mars Messages Run IO
Excepción de desbordamiento ocurrida en la dirección: 0x0040000c
En cualquier caso continuamos el programa...
-- program is finished running (dropped off bottom) --
```

```
1  # Reserva de espacio para guardar registros en kdata
2  .kdata
3  contexto: .word 0,0,0,0 # espacio para alojar cuatro registros
4  .ktext: 0x80000180 # Dirección de comienzo de la rutina
5  # Guardar registros a utilizar en la rutina.
6  la $k1, contexto
7  sw $at, 0($k1) # Guardamos $at
8  sw $t0, 4($k1)
9  sw $v0, 8($k1)
10 sw $a0, 12($k1)
11 #Comprobación de si se trata de una interrupción
12 mfc0 $k0, $13 # Registro Cause
13 srl $a0, $k0, 2 # Extraemos campo del código
14 andi $a0, $a0, 0x1f
15 bne $a0, $zero, acabamos # Sólo procesamos aquí E/S
16 #Tratamiento de la interrupción
17 li $t0, 0xffff0000
18 lb $a0, 4($t0) # Lee carácter del teclado
19 # Por ejemplo:
20 # Escribe en la consola del MARS el carácter leído
21 li $v0, 11
22 syscall
23 # Antes de acabar se podría dejar todo iniciado:
24 acabamos: mtc0 $0, $13 # Iniciar registro Cause
25 mfc0 $k0, $12 # Leer registre Status
26 andi $k0, 0xffffd # Iniciar bit de excepción
27 ori $k0, 0x11 # Habilitar interrupciones
28 mtc0 $k0, $12 # reescribir registre Status
29 # Restaurar registros
30 lw $at, 0($k1) # Recupero $at
31 lw $t0, 4($k1)
32 lw $v0, 8($k1)
33 lw $a0, 12($k1)
34 # Devolver en el programa de usuario
35 eret
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
24 acabamos: mtc0 $0, $13 # Iniciar registro Cause
25 mfc0 $k0, $12 # Leer registre Status
26 andi $k0, 0xffffd # Iniciar bit de excepción
27 ori $k0, 0x11 # Habilitar interrupciones
28 mtc0 $k0, $12 # reescribir registre Status
29 # Restaurar registros
30 lw $at, 0($k1) # Recupero $at
31 lw $t0, 4($k1)
32 lw $v0, 8($k1)
33 lw $a0, 12($k1)
34 # Devolver en el programa de usuario
35 eret
36
37 .data
38     mensaje: .asciiz "Pulsa teclas: \n"
39 .text
40 lui $t0, 0xffff # Dirige del registro de control
41 lw $t1, 0($t0) # Registre de control del receptor
42 ori $t1, $t1, 0x0002 # Habilitar interrupciones del teclado
43 sw $t1, 0($t0) # Actualizamos registro de control
44
45 mfc0 $a0, $12 # leer registre Status
46 ori $a0, 0xffff # Habilitar todas las interrupciones
47 mtc0 $a0, $12 # reescribir el registro status
48
49 la $a0, mensaje
50 li $v0, 4
51 syscall
52
53 inicio:
54 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:

