ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

ORGANIZACIÓN DE COMPUTADORES

GRADO EN INGENIERÍA DE COMPUTADORES

ACTIVIDADES PRÁCTICAS

ACTIVIDAD 2.2:
ESCRITURA EN UN DISPOSITIVO DE SALIDA EN MIPS32

CURSO 2019-20

Luis Rincón Córcoles
luis.rincon@urjc.es

Prólogo

En este cuaderno de prácticas se proponen varias actividades sobre escritura de un dispositivo de salida en sendos visualizadores de 7 segmentos incorporados en la herramienta **Digital Lab Sim** de **MARS**. Además, se ofrece una breve descripción de dicha herramienta.

Se realizarán dos versiones del ejercicio:

- Una primera versión, en la que un programa escribe directamente sobre los visualizadores.
- Una segunda versión, en la que la escritura sobre los visualizadores se realiza mediante una llamada a sistema operativo con la instrucción **syscall**.

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**.

2.2.1. LECTURA PREVIA

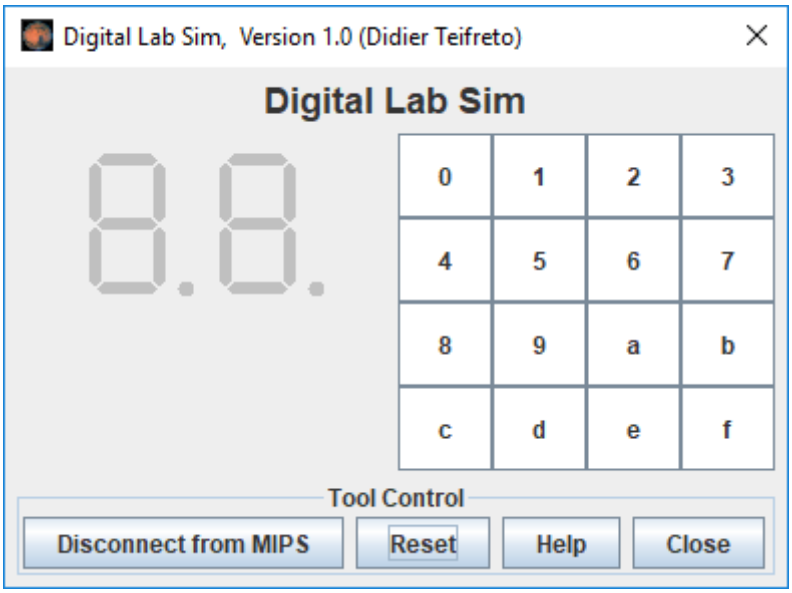
LABORATORIO DE SIMULACIÓN DIGITAL DE MARS

El simulador de laboratorio digital de **MARS** es una herramienta que simula varios elementos hardware sencillos:

- Dos visualizadores de 7 segmentos.
- Un teclado matricial hexadecimal.
- Un contador que puede actuar como un temporizador.

Esta herramienta es una contribución a **MARS** realizada por Didier Teifreto, de la Université de Franche-Comté (Francia). Dentro de MARS, es accesible a través del menú **Tools** seleccionando la opción **Digital Lab Sim**.

El aspecto de la herramienta es el siguiente:



En MIPS, el espacio de direcciones de memoria y entrada/salida es único, por lo cual las direcciones correspondientes a los controladores del laboratorio digital se mezclan con las de memoria, y son las siguientes:

DIRECCIÓN	FUNCIÓN	ELEMENTO ASOCIADO
0xFFFF0010	Registro de datos	Visualizador de 7 segmentos derecho
0xFFFF0011	Registro de datos	Visualizador de 7 segmentos izquierdo
0xFFFF0012	Registro de control	Teclado hexadecimal
0xFFFF0013	Registro de control	Contador
0xFFFF0014	Registro de datos	Teclado hexadecimal

A continuación describiremos cada uno de estos elementos.

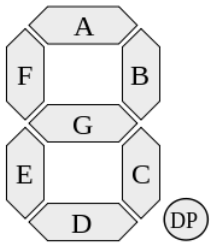
Visualizadores de 7 segmentos

Los visualizadores de 7 segmentos son sendos dispositivos de salida que no pueden generar interrupciones.

El visualizador que vemos a nuestra derecha se encuentra asociado a la dirección 0xFFFF0010, mientras que el que vemos a nuestra izquierda está asociado a la dirección 0xFFFF0011.

Los visualizadores se estructuran en segmentos, de modo que cada uno de los segmentos está controlado por un único bit:

- El segmento A está asociado al bit 0 (el menos significativo).
- El segmento B está asociado al bit 1.
- El segmento C está asociado al bit 2.
- El segmento D está asociado al bit 3.
- El segmento E está asociado al bit 4.
- El segmento F está asociado al bit 5.
- El segmento G está asociado al bit 6.
- El punto decimal (DP) está asociado al bit 7 (el más significativo).



Así, si queremos encender el segmento A del visualizador de la derecha, lo haremos mediante la secuencia

```
la    $t0,0Xffff0010    # Dirección base de los visualizadores
li    $t1,0x01
sb    $t1,0($t0)
```

Si queremos encender todos los segmentos del visualizador de la izquierda, incluyendo el punto decimal, lo haremos mediante la secuencia

```
la    $t0,0Xffff0010    # Dirección base de los visualizadores
li    $t1,0xff
sb    $t1,1($t0)
```

Si queremos escribir el número 7 en el visualizador de la derecha, tendremos que encender los segmentos A, B y C, y lo haremos con

```
la    $t0,0Xffff0010    # Dirección base de los visualizadores
li    $t1,0x07          # Binario: 00000111
sb    $t1,0($t0)
```

Contador

El contador es un elemento extremadamente sencillo que, cuando está activado, genera una interrupción cada cierto tiempo. Es decir, actúa como un temporizador, pero no es programable, ya que el lapso de tiempo entre interrupciones consecutivas viene dado por la ejecución de un número predefinido de instrucciones.

El contador tiene asociado un único *byte* en el mapa de direcciones, que se encuentra en la posición 0xFFFF0013, y corresponde con su registro de control. Si en dicha posición escribimos un 0, el contador queda desactivado y no generará interrupciones. En cambio, si en ella escribimos un 1, el contador generará una interrupción cada 30 instrucciones ejecutadas (este número no es configurable). Cuando se genere la interrupción, se pondrá a 1 el bit 10 del registro de causa (**Cause**) del coprocesador 0.

Teclado matricial hexadecimal

El último elemento del laboratorio digital simula un teclado hexadecimal matricial. Este teclado tiene dos bytes asociados en el mapa de direcciones, que corresponden con un registro de datos y otro de control, ambos de 8 bits de ancho:

REGISTRO DE CONTROL DEL TECLADO MATRICIAL HEXADECIMAL

	7	6	5	4	3	2	1	0
Dirección: 0xFFFF0012	IE				FILA CONSULTADA			

Si el bit IE del registro de control vale 1, las interrupciones generadas por el teclado estarán habilitadas. Cuando se pulse una tecla se generará una interrupción, y se activará el bit 11 del registro **Cause**.

REGISTRO DE DATOS DEL TECLADO MATRICIAL HEXADECIMAL

	7	6	5	4	3	2	1	0
Dirección: 0xFFFF0014	COLUMNA PULSADA				FILA PULSADA			

Para saber qué tecla se ha pulsado, es preciso buscar fila por fila activando en el registro de control un único bit que identifica la fila chequeada (bit 0 para la primera, bit 1 para la segunda, bit 2 para la tercera y bit 3 para la cuarta). Al escribir sobre el registro de control para chequear una fila, se obtiene la siguiente respuesta en el registro de datos:

- Si la tecla pulsada no estaba en dicha fila, el registro de datos tendrá todos sus bits a 0.

- Si la tecla sí estaba en esa fila, en los cuatro bits menos significativos del registro de datos se escribirá el código de la fila (el mismo que se escribió en el registro de control), y en sus cuatro bits más significativos se activará un único bit que identificará la columna en la que se encuentra la tecla pulsada (bit 4 para la primera, bit 5 para la segunda, bit 6 para la tercera y bit 7 para la cuarta), de acuerdo con lo siguiente:

Tecla	0	1	2	3	4	5	6	7
Código	0x11	0x21	0x41	0x81	0x12	0x22	0x42	0x82

Tecla	8	9	A	B	C	D	E	F
Código	0x14	0x24	0x44	0x84	0x18	0x28	0x48	0x88

El siguiente código chequea, de forma rudimentaria, el teclado matricial fila por fila para ver qué tecla se encuentra pulsada (el código de la tecla pulsada queda almacenado en **$a0**):

```
# Comprobar fila 1
            li    $v0,0x01
            sb    $v0,0xffff0012
            lb    $a0,0xffff0014
            bne   $a0,$zero,key_found
# Comprobar fila 2
            li    $v0,0x02
            sb    $v0,0xffff0012
            lb    $a0,0xffff0014
            bne   $a0,$zero,key_found
# Comprobar fila 3
            li    $v0,0x04
            sb    $v0,0xffff0012
            lb    $a0,0xffff0014
            bne   $a0,$zero,key_found
# Comprobar fila 4
            li    $v0,0x08
            sb    $v0,0xffff0012
            lb    $a0,0xffff0014
            bne   $a0,$zero,key_found
```

CUESTIONARIO 2.2.1

En preparación

2.2.2. ESCRITURA DIRECTA EN UN DISPOSITIVO DE SALIDA

INTRODUCCIÓN

En este ejercicio se escribirá un fragmento de código que escribirá dígitos hexadecimales en los visualizadores de 7 segmentos incorporados en la herramienta **Digital Lab Sim** de **MARS**.

Se parte del fichero **caso07.asm**, que pide al usuario que teclee un dato numérico positivo entre 0 y 255, para a continuación escribir su representación hexadecimal en los visualizadores de 7 segmentos. El programa imprimirá cada dígito en uno de los dos visualizadores de 7 segmentos, y después finalizará su ejecución.

El programa en ensamblador primero pide por teclado un dato numérico entre 0 y 255 y lo pone en **$v0**. Si el dato introducido está fuera de este rango, el programa lo descarta y solicita la introducción de un nuevo dato.

A continuación, se pone en **$a0** un dato que contiene los cuatro bits menos significativos de **$v0**, y se invoca a la subrutina **cod7seg**, que convierte este número a su representación codificada de acuerdo con los diodos de un visualizador de 7 segmentos. La subrutina devuelve el código en el byte menos significativo de **$v0**, y este dato deberá ser escrito en el visualizador derecho para que sus segmentos se iluminen adecuadamente.

Tras ello, se pone en **$a0** un dato que contiene los cuatro siguientes bits de **$v0**, y se procede como antes, pero ahora escribiendo el código devuelto en $v0 por **cod7seg** en el visualizador derecho. Después el programa termina.

El código de **caso07.asm** está incompleto. El alumno lo completará para que realice la tarea encomendada, incluyendo el código faltante en los huecos reservados para ello, de acuerdo en cada caso con los comentarios existentes junto a los mismos.

ANÁLISIS DEL CÓDIGO FUENTE DEL PROGRAMA

Subrutina de conversión a código de 7 segmentos

Para no repetir código, la conversión de un dato hexadecimal en el código que ilumina los diodos correspondientes en el visualizador hexadecimal se realiza mediante una subrutina, llamada **cod7seg**, que recibe su dato de entrada en **$a0**, y devuelve su resultado en **$v0**. La conversión se realiza mediante una tabla, llamada **cod7seg_tabla**.

En primer lugar se enmascara el dato de entrada, para convertir únicamente el dígito representado por sus últimos cuatro bits. El dato resultante queda en **$t0**. A continuación se carga **$t1** con la dirección base de la tabla. Y finalmente se suman **$t1** con **$t0** para que **$t1** quede apuntando al

código de 7 segmentos correspondiente al dato introducido. No es preciso escalar el contenido de **$t0**, ya que cada elemento de la tabla ocupa un byte.

A continuación se presenta el código completo de la subrutina:

```
cod7seg:
            andi  $t0,$a0,0x000F
            la    $t1,cod7seg_tabla
            add   $t1,$t1,$t0
            lbu   $v0,0($t1)
            jr    $ra
            .data
cod7seg_tabla:
            .byte 0x3f, 0x06, 0x5b, 0x4f
            .byte 0x66, 0x6d, 0x7d, 0x07
            .byte 0x7f, 0x67, 0x77, 0x7c
            .byte 0x39, 0x5e, 0x79, 0x71
```

Pedir dato por teclado

Para poder realizar esta tarea, se crea una cadena de caracteres que contiene el mensaje escrito en la consola para solicitar el dato:

```
            .data
str_pedir:  .asciiz     "Introduzca un dato numérico (0 <= dato <= 255): "
```

La solicitud de introducción del dato se realiza mediante un bucle que comprueba que el dato introducido se encuentra dentro del rango especificado:

```
            .text
main:
# Pedir el dato de entrada
pedir_dato:
            li    $v0,4       # Escribir tira de caracteres
            la    $a0,str_pedir
            syscall
            li    $v0,5       # Leer entero
            syscall
            blt   $v0,$zero,pedir_dato
            bgt   $v0,255,pedir_dato
```

Mediante este código, en el byte menos significativo de **$v0** queda escrito un dato que puede ser representado mediante dos cifras hexadecimales, cada una de las cuales se escribirá en uno de los visualizadores de 7 segmentos.

Escribir sobre los visualizadores

En primer lugar, el dato se copia en el registro **$s0**.

```
# Copiar dato en $s0
            move  $s0,$v0
```

A continuación, se enmascara el dato de modo que se ponen a 0 todos sus bits excepto los cuatro menos significativos, que contienen la parte menos significativa del dato tecleado, y el resultado se pone en **$a0**:

```
# Seleccionar los 4 bits menos significativos del dato
```

```
            andi  $a0,$s0,0x0f
```

Lo siguiente es convertir dicho dato a una representación adecuada para escribir sobre el visualizador de 7 segmentos. Para ello se invoca la subrutina **cod7seg**, que toma como entrada un número entre 0 y 15, y devuelve en **$v0** un dato de un byte en el que cada bit representa uno de los segmentos del visualizador, de modo que un cierto bit a 1 indica que dicho segmento debe estar encendido, y un cierto bit a 0 indica que dicho segmento debe estar apagado.

```
# Convertir la cifra hexadecimal a la codificación de 7 segmentos
            jal   cod7seg
```

A continuación se escribirá el dígito convenientemente codificado en el visualizador derecho. La operación exacta deberá ser aportada por el alumno, ya que en este punto el fichero está incompleto. El alumno escribirá el código necesario justo detrás de la etiqueta **escribir_derecho**.

Para escribir en el visualizador izquierdo se procede de forma muy similar. Pero en este caso la máscara aplicada es diferente, ya que hay que dejar intactos los bits 7-4 del dato. Para obtener la codificación en 7 segmentos mediante llamada a **cod7seg**, habrá que desplazar el dato enmascarado 4 lugares hacia la derecha:

```
# Seleccionar los 4 bits más significativos del dato y ponerlos a la derecha
            andi  $a0,$s0,0xf0
            srl   $a0,$a0,4
# Convertir la cifra hexadecimal a la codificación de 7 segmentos
            jal   cod7seg
```

Lo siguiente será escribir el dígito codificado en 7 segmentos en el visualizador izquierdo. Nuevamente, esta operación deberá ser añadida por el alumno, detrás de la etiqueta **escribir_izquierdo**, ya que en este punto el fichero también está incompleto.

REALIZACIÓN DEL CUESTIONARIO 2.2.2

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**.

Arrancar la herramienta **Digital Lab Sim**, que está en el menú **Tools**. Una vez arrancada, conectarla pulsando **Connect to MIPS**. Recordar que, después de cada prueba, para reiniciar la herramienta será preciso pulsar el botón **Reset** en la parte inferior de la misma.

A continuación, cargar el fichero **caso07.asm** con la opción **File -> Open**.

Llegados a este punto, es necesario editar el fichero **caso07.asm** para rellenar las dos operaciones que faltan, y que corresponden con la escritura en ambos visualizadores.

1. **¿Cuál de las siguientes operaciones sirve para escribir un dato codificado en 7 segmentos y contenido en $v0 en el visualizador derecho?**
   a) lb $v0,0xffff0010
   b) sb $v0,0xffff0010
   c) lb $v0,0xffff0011
   d) sb $v0,0xffff0011
   e) Ninguna de las restantes respuestas es cierta.

Escribir la operación correcta justo tras la etiqueta **escribir_derecho**.

2. **¿Cuál de las siguientes operaciones sirve para escribir un dato codificado en 7 segmentos y contenido en $v0 en el visualizador izquierdo?**
   a) lb $v0,0xffff0010
   b) sb $v0,0xffff0010
   c) lb $v0,0xffff0011
   d) sb $v0,0xffff0011
   e) Ninguna de las restantes respuestas es cierta.

Escribir la operación indicada justo tras la etiqueta **escribir_izquierdo**.

Guardar el fichero con el nombre **caso07_solucion.asm**. A continuación, ensamblarlo mediante la opción **Run -> Assemble**.

Poner un punto de ruptura en la instrucción cuya dirección coincide con la etiqueta **procesar_derecho**, y ejecutar el programa con **Run -> Go**. Cuando se nos pida introducir un dato numérico por teclado, escribir **100** y pulsar **INTRO**. La ejecución se detendrá en el punto de ruptura.

3. **¿Cuál es el contenido del byte menos significativo del registro $v0, en hexadecimal? (escribir exclusivamente sus dos últimos dígitos, precedidos del prefijo 0x)**

Ejecutar únicamente la instrucción cuya dirección coincide con la etiqueta **procesar_derecho**, que copia el contenido de **$v0** en **$s0**.

4. **¿Por qué motivo se ha copiado $v0 en $s0 mediante la instrucción de cuya dirección coincide con la etiqueta procesar_derecho?**
   a) Para poder conservar el dato introducido por teclado después de llamar con posterioridad a la subrutina **cod7seg**.
   b) Es una operación innecesaria.
   c) Porque no se puede hacer una operación **andi** usando el registro **$v0**.
   d) Ninguna de las restantes respuestas es cierta.

Ahora, analizar y ejecutar la instrucción **andi $a0,$s0,0x0f**.

5. **¿Qué hace la instrucción andi $a0,$s0,0x0f?**

a) Poner a 0 todos los bits de **$s0** excepto los cuatro menos significativos.
b) Copiar **$s0** en **$a0**.
c) Copiar en **$a0** los cuatro bits menos significativos de **$s0** y poner a 0 los restantes bits de **$a0**.
d) Ninguna de las restantes respuestas es cierta.

6. **¿Para qué se realiza esta operación?**
   a) Para eliminar la parte sobrante del dato que se ha tecleado y que no puede representarse en los visualizadores.
   b) Para pasar el dígito que queremos escribir en el visualizador derecho como argumento a la subrutina **cod7seg**, codificado con cuatro bits.
   c) Para convertir el dato tecleado a su representación hexadecimal.
   d) Ninguna de las restantes respuestas es cierta.

Poner un punto de ruptura en la instrucción asociada a la etiqueta **escribir_derecho**. Ahora, ejecutar la subrutina **cod7seg** completa. La ejecución se detendrá tras retornar de la misma.

7. **¿Qué contenido tiene $v0 tras retornar de la subrutina? (escribir exclusivamente sus dos últimos dígitos, precedidos del prefijo 0x)**
8. **¿Qué representa en este punto el contenido de $v0?**
   a) El dato que hemos leído por teclado, codificado en hexadecimal con dos cifras.
   b) El dígito que hay que escribir en el visualizador derecho, codificado en hexadecimal con dos cifras.
   c) El dígito hexadecimal que hay que escribir en el visualizador derecho, codificado según el código de 7 segmentos con un byte.
   d) Ninguna de las restantes respuestas es cierta.

Poner un punto de ruptura en la instrucción asociada a la etiqueta **procesar_izquierdo**. Ahora, ejecutar el programa, que se detendrá en dicha instrucción.

9. **¿Qué cifra hexadecimal se ha escrito en el visualizador derecho? (escribirla utilizando exclusivamente un único carácter, sin ningún prefijo)**

Ahora, analizar y ejecutar la instrucción **andi $a0,$s0,0xf0**.

10. **¿Qué hace la instrucción andi $a0,$s0,0xf0?**
    a) Poner a 0 todos los bits de **$s0** excepto los que ocupan las posiciones 7 a 4.
    b) Copiar **$s0** en **$a0**.
    c) Copiar en **$a0** los bits 7 a 4 de **$s0** y poner a 0 los restantes bits de **$a0**.
    d) Ninguna de las restantes respuestas es cierta.
11. **¿Para qué se realiza esta operación?**

a) Para eliminar la parte sobrante del dato que se ha tecleado y que no puede representarse en los visualizadores.
b) Para aislar los cuatro bits que conforman el dígito que queremos escribir en el visualizador izquierdo, y poder pasarlos como argumento a la subrutina **cod7seg**.
c) Para convertir el dato tecleado a su representación hexadecimal.
d) Ninguna de las restantes respuestas es cierta.

Ahora, analizar y ejecutar la instrucción **srl $a0,$a0,4**.

12. **¿Qué hace la instrucción srl $a0,$a0,4?**
a) Desplazar el contenido del registro **$a0** cuatro posiciones a la derecha según un desplazamiento aritmético.
b) Desplazar el contenido del registro **$a0** cuatro posiciones a la derecha según un desplazamiento lógico.
c) Realizar una rotación a la izquierda del registro **$a0** de cuatro posiciones.
d) Ninguna de las restantes respuestas es cierta.

13. **¿Para qué se realiza esta operación?**
a) Para poner el dato que queremos escribir en el visualizador izquierdo en la mitad izquierda del registro **$a0**.
b) Para poder pasar el número que queremos escribir en el visualizador izquierdo como parámetro a la subrutina **cod7seg**.
c) Para convertir el dato tecleado en su parte izquierda a hexadecimal.
d) Ninguna de las restantes respuestas es cierta.

Poner un punto de ruptura en la instrucción asociada a la etiqueta **escribir_izquierdo**. Ahora, ejecutar la subrutina **cod7seg** completa. La ejecución se detendrá tras retornar de la misma.

14. **¿Qué contenido tiene $v0 tras retornar de la subrutina? (escribir exclusivamente sus dos últimos dígitos, precedidos del prefijo 0x)**

15. **¿Qué representa en este punto el contenido de $v0?**
a) El dato que hemos leído por teclado, codificado en hexadecimal con dos cifras.
b) El dígito que hay que escribir en el visualizador izquierdo, codificado en hexadecimal con dos cifras.
c) El dígito hexadecimal que hay que escribir en el visualizador, codificado según el código de 7 segmentos con un byte.
d) Ninguna de las restantes respuestas es cierta.

Poner un punto de ruptura en la instrucción asociada a la etiqueta **fin**. Ahora, ejecutar el programa, que se detendrá en dicha instrucción.

16. **¿Qué cifra hexadecimal se ha escrito en el visualizador derecho? (escribirla utilizando exclusivamente un único carácter, sin ningún prefijo)**

Ejecutar el resto del programa. Pulsar el botón **Clear** de la zona de mensajes y de la consola, y cerrar el fichero **caso07-solucion.asm** mediante la opción **File -> Close**.

2.2.3. ESCRITURA EN DISPOSITIVO DE SALIDA MEDIANTE SERVICIO DEL SISTEMA OPERATIVO

INTRODUCCIÓN

En cualquier sistema operativo, las direcciones de los controladores de los dispositivos de E/S pertenecen al espacio del sistema operativo. Por tanto, los programas de usuario no deberían escribir directamente en los controladores de los periféricos de salida, que es lo que se ha hecho en el apartado previo. En el presente apartado, este problema se resolverá mediante la implementación de un servicio de sistema operativo que se encargará de realizar la escritura. Los programas que quieran escribir en los visualizadores de 7 segmentos utilizarán dicho servicio, y será el manejador de excepciones quien, a través de una rutina de tratamiento, realizará la escritura.

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

DEFINICIÓN DEL SERVICIO DE ESCRITURA EN LOS VISUALIZADORES DE 7 SEGMENTOS

Los datos del servicio implementado son los siguientes:

- **Nombre del servicio**: write_d7seg.
- **Código del servicio** (en **$v0**): 1001.
- **Argumentos**:
  - **$a0**: dato que se quiere escribir en el visualizador, con la codificación de los segmentos HGFEDCBA propia de estos dispositivos. El bit H corresponde con el punto decimal, mientras que los demás bits corresponden con los segmentos del visualizador.
  - **$a1**: identificador del visualizador (0: derecho; 1: izquierdo).
- **Valor de retorno**: ninguno.

Para invocar este servicio, el programa principal:

- Escribirá en **$v0** el número del servicio (1001).

- Escribirá en **$a0** el dato que se pretende escribir en el visualizador, codificado con 7 segmentos.
- Escribirá en **$a1** el identificador del visualizador.
- Ejecutará la instrucción **syscall**.

El fichero **caso08.asm** contiene un programa que escribe en los visualizadores de 7 segmentos mediante la instrucción **syscall**.

La rutina de tratamiento de llamada a sistema, codificada en espacio del *kernel*, se encargará de leer el código del servicio y sus argumentos, tras lo que realizará el servicio pedido. Si hubiera algún valor de retorno, la rutina de tratamiento de **syscall** lo pondría en el registro **$v0**.

Por tanto, en este ejercicio el alumno deberá modificar la rutina de tratamiento de la excepción **syscall** para que mire el valor de **$v0**, y si es igual a 1001, salte a la rutina del servicio **write_d7seg**, que escribirá sobre el visualizador correspondiente el dato pedido, previa consulta de los contenidos de **$a0** y **$a1**. Si **$v0** no contiene el valor 1001, la rutina de tratamiento de la excepción **syscall** escribirá un mensaje por pantalla como hasta ahora.

DESCRIPCIÓN DEL MANEJADOR

La estructura del manejador propuesto coincide con la del manejador con rutinas específicas. Como novedades tenemos las siguientes:

- En la sección de datos se incluyen varias constantes que sirven para dar nombres a las direcciones de los dispositivos de entrada/salida simulados en **Digital Lab Sim**.
- También en la sección de datos se incluye una definición de constante para darle nombre al servicio de escritura en los visualizadores.
- En la rutina de tratamiento de la excepción por **syscall**, hay que identificar el servicio solicitado y saltar a una rutina de servicio que realice la operación pedida.

```
# MANEJADOR DE EXCEPCIONES: VARIABLES Y DEFINICIONES
            .kdata
# Zona de salvaguarda de registros
# Mensajes
# Tabla de punteros a los mensajes de excepción
# Tabla de vectores de excepción
# Direcciones de E/S de los dispositivos:
            .eqv  D7SEG_RIGHT_ADDR,0xFFFF0010   # Display de 7 segmentos derecho
            .eqv  D7SEG_LEFT_ADDR,0xFFFF0011    # Display de 7 segmentos izquierdo
            .eqv  HEXKEY_CTRL_ADDR,0xFFFF0012   # Teclado hexadecimal: control
            .eqv  COUNTER_ADDR,0xFFFF0013       # Contador
            .eqv  HEXKEY_DATA_ADDR,0xFFFF0014   # Teclado hexadecimal: datos
# Códigos de los servicios de llamada a sistema (syscall)
            .eqv  WRITE_D7SEG,1001

# 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
```

Rutina de tratamiento de la excepción syscall

Esta rutina identifica el servicio solicitado, y llama a una rutina aún más específica de servicio del mismo, que después retorna a la rutina de **syscall**.

Ante una llamada a servicio correctamente implementado, la rutina de tratamiento de **syscall** deberá modificar el valor de **EPC** sumándole 4. Así, el manejador, al terminar, restaurará sobre el **PC** el valor de la instrucción siguiente a **syscall**, con lo que continuará la ejecución del programa que invocó el servicio.

Si el servicio invocado no está implementado, se producirá un error fatal. En tal caso, 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.

```
SYSCALL_RTE:
# Chequear el registro $v0 para identificar el servicio y saltar a rutina
##### EL ALUMNO SUSTITUIRÁ ESTOS COMENTARIOS POR CÓDIGO ENSAMBLADOR
##### QUE SALTE A LA RUTINA DE SERVICIO CORRESPONDIENTE
# Si el servicio no está implementado, presentar mensajes y abortar
            j     GEN_RTE
# Punto de retorno para los servicios syscall implementados
continue_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
```

Rutina de servicio para escribir en los visualizadores

Esta rutina comprueba el valor del parámetro **$a1**, y en función del mismo escribe el código contenido en **$a0** sobre el visualizador correspondiente. Después, se retornará a la parte general del manejador mediante la instrucción **j continue_syscall**. Si el contenido de **$a1** no es ni 1 (visualizador izquierdo) ni 0 (visualizador derecho), entonces se saltará a la parte general del manejador con **j continue_syscall** sin escribir nada en los visualizadores.

```
WRITE_D7SEG_RTE:
##### EL ALUMNO ESCRIBIRÁ AQUÍ EL CÓDIGO ENSAMBLADOR
##### QUE IMPLEMENTE LA RUTINA DE SERVICIO
# Si $a1=0, escribir sobre el visualizador derecho y saltar a continue_syscall
```

```
# Si $a1=1, escribir sobre el visualizador izquierdo y saltar a continue_syscall
# Si $a1 no es ni 0 ni 1, saltar directamente a continue_syscall sin escribir en
el visualizador
            j     continue_syscall
```

Esta rutina debe ser completada por el alumno.

REALIZACIÓN DEL CUESTIONARIO 2.2.3

Abrir **MARS**, y cargar el fichero **excepciones_syscall_1001.asm** con la opción **File -> Open**, para editarlo rellenando el código faltante. Es preciso poner atención en las nuevas constantes definidas en la sección de datos, y también en el nuevo código (con comentarios y huecos) para las rutinas de tratamiento de **syscall** y servicio de escritura en los visualizadores.

1. **En la rutina de tratamiento de syscall, ¿cuál de las siguientes operaciones sirve para chequear el número de servicio y saltar en su caso a la rutina de servicio correspondiente?**
   a) `j continue_syscall`
   b) `beq $v0, WRITE_D7SEG,WRITE_D7SEG_RTE`
   c) `bne $v0, WRITE_D7SEG,WRITE_D7SEG_RTE`
   d) Ninguna de las restantes respuestas es cierta.
2. **En la rutina del servicio de escritura sobre los visualizadores, ¿para qué puede servir la operación beq $a1,$zero,etiqueta1?**
   a) Para saltar a **etiqueta1** si hay que escribir sobre el visualizador izquierdo.
   b) Para saltar a **etiqueta1** si hay que escribir sobre el visualizador derecho.
   c) Para saltar a **etiqueta1** cuando el identificador del visualizador proporcionado en la petición de servicio es erróneo.
   d) Ninguna de las restantes respuestas es cierta.
3. **En la rutina del servicio de escritura sobre los visualizadores, ¿para qué puede servir la operación j continue_syscall?**
   a) Para saltar a **continue_syscall** si hay que escribir sobre el visualizador izquierdo.
   b) Para saltar a **continue_syscall** si hay que escribir sobre el visualizador derecho.
   c) Para saltar a **continue_syscall** cuando el identificador del visualizador proporcionado en la petición de servicio es erróneo.
   d) Ninguna de las restantes respuestas es cierta.
4. **En la rutina del servicio de escritura sobre los visualizadores, ¿para qué puede servir la operación beq $a1,1,etiqueta2?**
   a) Para saltar a **etiqueta2** si hay que escribir sobre el visualizador izquierdo.
   b) Para saltar a **etiqueta2** si hay que escribir sobre el visualizador derecho.

c) Para saltar a **etiqueta2** cuando el identificador del visualizador proporcionado en la petición de servicio es erróneo.
d) Ninguna de las restantes respuestas es cierta.

5. **En la rutina del servicio de escritura sobre los visualizadores, ¿para qué puede servir la operación li $k0,0xffff0010?**
a) Para que **$k0** contenga la dirección del puerto correspondiente al registro de control del contador.
b) Para que **$k0** contenga la dirección del puerto correspondiente al registro de control del teclado hexadecimal.
c) Para que **$k0** contenga la dirección del puerto correspondiente al registro de datos del teclado hexadecimal.
d) Para que **$k0** contenga la dirección del puerto correspondiente al visualizador derecho.
e) Para que **$k0** contenga la dirección del puerto correspondiente al visualizador izquierdo.
f) Ninguna de las restantes respuestas es cierta.

6. **En la rutina del servicio de escritura sobre los visualizadores, ¿a qué constante equivale la dirección 0xffff0010?**
a) A COUNTER_ADDR.
b) A D7SEG_LEFT_ADDR.
c) A D7SEG_RIGHT_ADDR.
d) A HEXKEY_CTRL_ADDR.
e) A HEXKEY_DATA_ADDR.
f) Ninguna de las restantes respuestas es cierta.

7. **En la rutina del servicio de escritura sobre los visualizadores, ¿para qué puede servir la operación li $k0,0xffff0011?**
a) Para que **$k0** contenga la dirección del puerto correspondiente al registro de control del contador.
b) Para que **$k0** contenga la dirección del puerto correspondiente al registro de control del teclado hexadecimal.
c) Para que **$k0** contenga la dirección del puerto correspondiente al registro de datos del teclado hexadecimal.
d) Para que **$k0** contenga la dirección del puerto correspondiente al visualizador derecho.
e) Para que **$k0** contenga la dirección del puerto correspondiente al visualizador izquierdo.

8. **En la rutina del servicio de escritura sobre los visualizadores, ¿a qué constante equivale la dirección 0xffff0011?**
a) A HEXKEY_CTRL_ADDR.
b) A HEXKEY_DATA_ADDR.
c) A D7SEG_LEFT_ADDR.
d) A D7SEG_RIGHT_ADDR.
e) A COUNTER_ADDR.
f) Ninguna de las restantes respuestas es cierta.

9. **En la rutina del servicio de escritura sobre los visualizadores, si utilizamos las operaciones li $k0,0xffff0011 y li $k0,0xffff0011, ¿con qué operación podríamos escribir en el visualizador requerido?**
   a) sw $k0,0($a0)
   b) sw $a0,0($k0)
   c) sh $k0,0($a0)
   d) sh $a0,0($k0)
   e) sb $k0,0($a0)
   f) sb $a0,0($k0)
   g) Ninguna de las restantes respuestas es cierta.
10. **En la rutina de tratamiento de syscall, ¿qué sucedería dejásemos el EPC intacto sin sumarle 4?**
   a) Se produciría una excepción, porque el manejador retornaría a una dirección errónea.
   b) Nada, al salir del manejador continuaría la ejecución del programa en curso normalmente.
   c) Al retornar del manejador, volvería a ejecutarse la instrucción syscall.
   d) Ninguna de las restantes respuestas es cierta.

Es preciso rellenar el código que falta en la rutina de **syscall** y en la rutina de servicio, y grabar el fichero resultante con el nombre **excepciones_syscall_1001-solucion.asm**. Después, entrar en **Exception Handler ...** dentro del menú **Settings**, activar la opción **Include this exception handler file in all assemble operations** y usar los botones **Browse** y **OK** para cargar el manejador del fichero recién grabado. Cerrar los ficheros abiertos con **File -> Close All**.

A continuación, cargar el fichero **caso08.asm** con la opción **File -> Open**, y ensamblarlo con la opción **Run -> Assemble**. Después, ejecutar el programa **mediante la opción Run -> Go**. Probar el funcionamiento del programa y ver si se escriben en los visualizadores los dígitos correctos. Si no, abrir el fichero **excepciones_syscall_1001-solucion.asm** y editarlo de nuevo. No olvidar grabarlo y cerrarlo antes de realizar pruebas adicionales.

Pulsar el botón **Clear** de la zona de mensajes y de la consola, cerrar todos los ficheros abiertos mediante la opción **File -> Close All**, y cerrar **MARS**.