

Práctica 12: Sincronización por consulta de estado

Objetivos

Afianzar los conocimientos en sincronización con periféricos mediante consulta del estado. Para conseguir este objetivo se han integrado en el simulador dos periféricos: el teclado y la consola.

Material

Esta práctica utiliza **PCSpim-ES** y los archivos con código fuente *espera.asm* y *eco.asm*. Todo este material está disponible en la carpeta correspondiente de PoliformaT.

PCSpim-ES es una versión del **simulador PCSpim** al que se le han añadido dos periféricos. Se usará también esta versión del simulador en la práctica siguiente. Confirme que dispone de la versión adaptada consultando el cuadro *Help>AboutPCSpim* de la aplicación una vez abierta, que debe incluir el texto “PCSpim Version 1.0 - adaptación para las asignaturas ETC2 y EC, etc...”. La figura 1 muestra la configuración apropiada en el cuadro de configuración del simulador (*simulator>settings*). Observe que hay un botón nuevo, *Syscall Exception*, que ha de dejar sin marcar de momento.

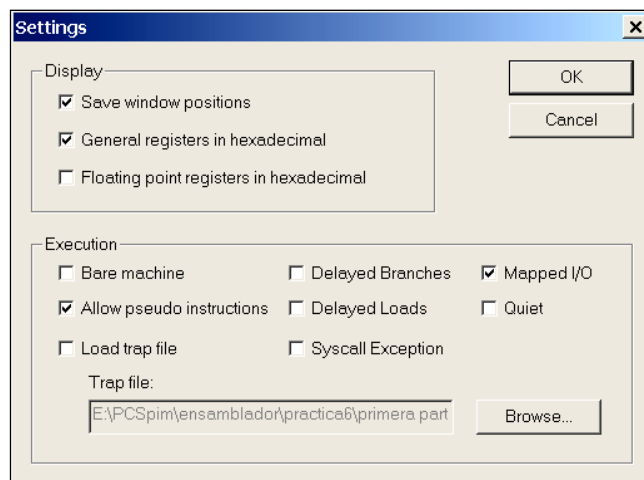


Figura 1. Configuración del simulador necesaria para la práctica

La E/S en PCSpim

Normalmente, los programas de usuario no acceden a las interfaces de los periféricos mediante las instrucciones de acceso a la memoria. Además, los sistemas operativos usuales prohíben a los programas corrientes este tipo de acceso por muchas razones. En su lugar, los programas han de utilizar las funciones de entrada/salida del sistema que realizan el trabajo de manera segura y eficiente. Estas funciones sí que acceden a las interfaces mediante las instrucciones de memoria.

El entorno de PCSpim es semejante a un sistema operativo, pero con diferencias importantes:

- Las funciones de E/S predefinidas en PCSpim son simples y no están pensadas para su uso por procesos concurrentes, véase el Apéndice Llamadas gestionadas por PCSPIM.

- Los programas no están obligados a usar las funciones y pueden acceder a la interfaz (Figura 2) de los periféricos disponibles sin ninguna restricción

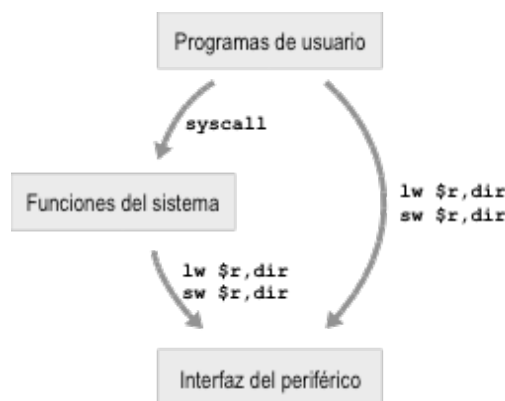


Figura 2. Acceso a los periféricos bajo PCSpim

Los periféricos disponibles (Figura 3) son dos: teclado y consola. El teclado y la consola son los habituales del PCSpim que llevan asociadas funciones conocidas, como `read_string` o `print_char`. Cada uno de los periféricos tiene un adaptador simulado que crea una interfaz con la que se tiene que trabajar en esta práctica.



Figura 3. Esquema completo del computador simulado por PCSpim. Además del procesador y la memoria están el monitor y teclado.

1. La interfaz del teclado

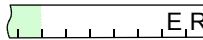
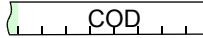
Cuando PCSpim está activo, el teclado es un periférico de entrada cuya operación elemental es la lectura de un carácter. Esa operación de entrada se lleva a cabo cuando se pulsa una tecla y la interfaz suministra al programa el código del carácter tecleado. La interfaz del teclado está compuesta por los dos registros que describe la Figura 4. Ambos registros están accesibles en el espacio de direcciones del MIPS a partir de la dirección base $DB = 0xFFFF0000$ y tienen 32 bits, aunque los bits útiles se encuentran en la parte menos significativa del contenido.

Observe que para leer de estos registros se pueden utilizar las instrucciones `lw`, `lhu`, `lh`, `lbu` o `lb` porque los bits operativos se encuentran en el byte menos significativo de la palabra. La utilización del bit E se aplicará en la práctica siguiente, en esta práctica se **deberá mantener el bit E = 0**.

El bit de preparado del teclado

El bit R indica que el teclado está preparado. Cada vez que se pulsa una tecla, este bit toma el valor 1. Leyendo este bit, los programas pueden saber cuándo hay que tratar al periférico.

Interfaz de teclado (DB = 0xFFFF0000)

Nombre	Dirección	Acceso	Estructura
Estado/órdenes	DB	L/E	
Datos	DB+4	L	

Registro de órdenes y estado (Lectura/escritura. Dirección = Base).

- R (bit 0, sólo lectura). Indicador de dispositivo preparado: R = 1 cada vez que se pulsa una tecla. Para hacer R = 0 es necesario realizar un acceso de lectura en el registro de datos.
- E: (bit 1, lectura/escritura). Habilitación de la interrupción (mientras E = 1, el valor R = 1 activa la línea de interrupción del dispositivo).

Registro de datos (Sólo lectura. Dirección = Base + 4).

- COD (bits 7...0). Código ASCII de la tecla pulsada. Leer de este registro provoca que R = 0.

Figura 4. Descripción de la interfaz del teclado. En esta práctica se deberá mantener el bit E = 0

Actividad número 1. Comprensión de la consulta del bucle de espera.

► Con un editor de textos, abre el programa *espera.asm* y observa su código. Identifica las tres partes principales: la escritura de una cadena T1 en la consola mediante una función del sistema, el proceso de espera que se estudiará a continuación y la escritura de T2 en la consola.

La Figura 5 muestra la estructura del programa, que contiene un bucle que espera a que el bit de preparado esté a 1. Observa los siguientes detalles sobre la espera:

- Se carga en \$t0 la dirección base de la interfaz 0xFFFF0000.
 - Se lee el registro de estado de la interfaz (desplazamiento 0 respecto la dirección base).
 - Se aísla el bit R mediante una operación lógica (and) y la máscara adecuada.
 - Sólo se sale del bucle cuando R=1.
- Carga el programa *espera.asm* con el simulador y ejecútalo con la orden *Simulator>Go* (F5). Notarás que inicialmente aparece el texto T1, y que cuando se pulsa una tecla aparece el texto T2.
- Vuelve a ejecutar el programa sin cerrar el simulador (por ejemplo, seleccionando *Simulator>Reload* en el menú de PCSpim). Verás que ya no hay espera: la razón es que el bit de preparado de la interfaz conserva el valor R = 1 desde la ejecución anterior.

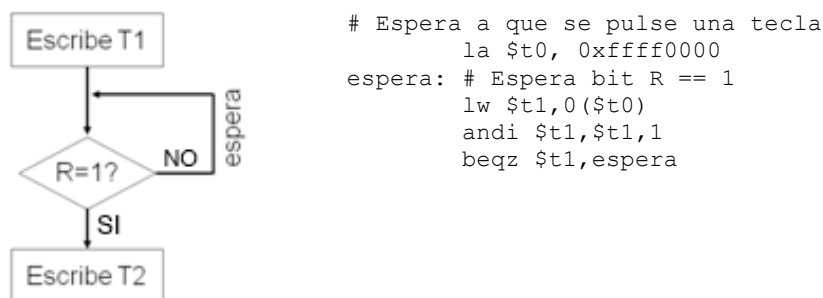


Figura 5. A la izquierda, estructura del programa *espera.asm*. A la derecha, un detalle del proceso de espera.

► Cuestión 1.

Si cambiara la interfaz del teclado de manera que el bit *R* pasara a ocupar la posición 5 de la palabra de control/estado, ¿qué instrucción debería cambiarse y cómo?

Actividad número 2. La cancelación del periférico.

La cancelación es el mecanismo que devuelve a *R* el valor 0 y lo deja dispuesto para una nueva espera. En esta interfaz, la cancelación se realiza mediante la lectura del registro de datos.

► **Cuestión 2.** Modifica *espera.asm* añadiendo una instrucción que lea del registro de datos y deje el contenido en *\$t2*, como indica la Figura 6. Observa que la dirección de este registro se expresa en la Figura 4 como “DB+4”.

► Prueba a ejecutar varias veces seguidas el nuevo programa modificado, sin cerrar el simulador. Notarás que el programa siempre espera a que se pulse una tecla, porque el bit de preparado queda con el valor *R* = 0 al final de la ejecución.

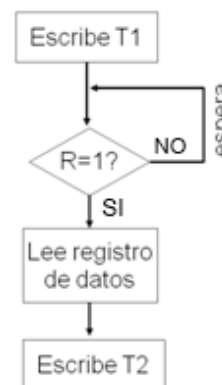


Figura 6. Estructura de *espera.asm* modificado.

2. Sincronización por consulta de estado

Cuando un programa espera a que un periférico esté preparado leyendo repetidamente su registro de estado, se habla de sincronización por consulta de estado. En general, el esquema de trabajo con el periférico es el que muestra la figura 7. El tratamiento depende del dispositivo y de la operación que se realice con él, pero en todo caso ha de cancelar el bit de preparado.

► Escribe un programa (en un nuevo archivo *ascii.asm*) que lea del teclado las teclas que se pulsán y escriba su código ASCII en la consola. El programa ha de terminar cuando se pulse una tecla elegida (como un retorno de carro, el carácter punto, etc.). El tratamiento será:

- Leer el registro de datos de la interfaz del teclado
- Escribir en pantalla el valor leído mediante la función del sistema `print_int`

El esquema completo del programa sería:

Repetir

*esperar a que el teclado esté preparado (bit *R* = 1)*

Tratamiento:

leer el registro de datos del teclado

escribir en pantalla el valor leído

Hasta que carácter = carácter elegido.



Figura 7. Flujo habitual del tratamiento de un periférico mediante consulta del estado.

Para escribir el programa se pueden utilizar bastantes líneas de código de *espera.asm*, por lo tanto se recomienda copiar *espera.asm* en un archivo *ascii.asm* e ir introduciendo los cambios sobre él.

► **Cuestión 3.** Copia aquí las líneas de código que realizan la sincronización y la lectura del registro de datos.

3. La interfaz de consola

La Figura 8 muestra la interfaz de la consola de PCSpim. La operación elemental de salida de datos por la consola consiste en imprimir un carácter. El carácter se imprime cuando el programa escribe el código del carácter en el registro de datos de la interfaz. Respecto de la interfaz del teclado, se encontrarán las siguientes diferencias:

- La dirección base ha cambiado. Ahora es 0xFFFF0008.
- El registro de datos es de escritura.

Interfaz de consola (DB = 0xFFFF0008)

Nombre	Dirección	Acceso	Estructura
Estado/órdenes	DB	L/E	<div style="border: 1px solid black; padding: 2px; display: inline-block; width: 100px; text-align: right;">E,R</div>
Datos	DB+4	E	<div style="border: 1px solid black; padding: 2px; display: inline-block; width: 100px; text-align: right;">COD</div>

Registro de órdenes y estado (Lectura/escritura. Dirección = Base).

- R (bit 0, sólo lectura). Indicador de dispositivo preparado: R = 1 cuando la consola está disponible. R = 0 cuando se escribe en el registro de datos.
- E: (bit 1, lectura/escritura). Habilidad de la interrupción (mientras E = 1, el valor R = 1 activa la línea de interrupción del dispositivo).

Registro de datos (Sólo escritura. Dirección = Base + 4).

- COD (bits 7...0). Código ASCII de del carácter que se ha de escribir en la consola. Escribir en este registro provoca que R = 0 mientras se procesa la operación.

Figura 8. Interfaz de la consola de PCSpim. En esta práctica se deberá mantener el bit E = 0.

El bit de preparado de la consola

La consola requiere cierto tiempo para realizar la operación, a lo largo del cual el bit R de preparado vale 0. Solo cuando la operación termina, el bit R vuelve a valer 1. Por tanto, antes de escribir un carácter es necesario esperar a que R = 1.

Actividad número 3. Funciones básicas con el teclado y la consola.

Para el trabajo con ambos periféricos se suelen implementar dos funciones:

- `void putchar(char c);` escribe un carácter en la consola
- `char getchar();` toma un carácter del teclado

Ambas funciones existen con el mismo nombre en la biblioteca de entrada/salida estándar `<stdio.h>` de los lenguajes de programación C y C++, y tienen semejantes en Java como `TextIO.put(char c)` y `TextIO.getAnyChar()`. El entorno PCSpim las ofrece también como `read_char` y `print_char` (funciones 12 y 11).

► Abre el programa *eco.asm* con un editor y observa su estructura. A partir de la etiqueta `__start` encontrarás el programa principal, que llama a la función *putchar* para escribir en la consola los caracteres 'P12' y el final de línea. A continuación de la etiqueta bucle hay una iteración que lee caracteres del teclado llamando a *getchar* y escribe el eco en la consola llamando a *putchar*. Fíjate en que el programa acaba cuando lee la tecla de escape (ASCII 27).

► Completa el código de las rutinas *getchar* y *putchar* a continuación de las etiquetas correspondientes. **Hazlo siguiendo el convenio de programación habitual para las funciones y sin utilizar las funciones de PCSpim.** Es decir:

- *getchar* ha de sincronizarse con el teclado mediante consulta de estado, tomar el carácter de su registro de datos y depositarlo en `$v0`.
- *putchar* ha de sincronizarse con la consola mediante consulta de estado y escribir en su registro de datos el contenido de `$a0`.

► **Cuestión 4.** Escribe el código de *getchar* y *putchar*.

--	--

► Ejecuta *eco.asm* (con *Simulator>Run* o pulsando [F5]) y detenlo mediante *Simulator>Break* cuando esté esperando a que se pulse una tecla, después de mostrar en la consola el texto "P12". Consulte el valor del PC e identifica la instrucción a la que señala.

Instrucción:	Valor de PC (hex):
--------------	--------------------

► **Cuestión 5.** ¿Puedes explicar por qué el programa se ha detenido en ese punto?

Apéndice. Llamadas gestionadas por PCSPIM

Service	System call code	Arguments	Result
print_int	1	\$a0 = integer	
print_float	2	\$f12 = float	
print_double	3	\$f12 = double	
print_string	4	\$a0 = string	
read_int	5		integer (in \$v0)
read_float	6		float (in \$f0)
read_double	7		double (in \$f0)
read_string	8	\$a0 = buffer, \$a1 = length	
sbrk	9	\$a0 = amount	address (in \$v0)
exit	10		
print_char	11	\$a0 = char	
read_char	12		char (in \$a0)
open	13	\$a0 = filename (string), \$a1 = flags, \$a2 = mode	file descriptor (in \$a0)
read	14	\$a0 = file descriptor, \$a1 = buffer, \$a2 = length	num chars read (in \$a0)
write	15	\$a0 = file descriptor, \$a1 = buffer, \$a2 = length	num chars written (in \$a0)
close	16	\$a0 = file descriptor	
exit2	17	\$a0 = result	