# Práctica 10
# Entrada/salida 1

## Objetivos

- Conocer las funciones de la unidad de entrada y salida.
- Entender el mecanismo de entrada/salida mapada en memoria
- Entender el método de sincronización por consulta de estado o polling .

## Material

Simulador MARS y un código fuente de partida

## Teoría

### 1. Selección, sincronización y transferencia

El sistema de entrada y salida es básico en un computador para poder comunicar el procesador con todas el dispositivos periféricos conectados a él, tanto de entrada (ratón, teclado..), de salida (impresora, pantalla...) o de entrada-salida (disco). Tratar con los periféricos es una tarea mucho más difícil que, por ejemplo, acceder a la memoria. La memoria tiene un comportamiento totalmente predecible, siempre está dispuesta para ser leída o escrita. Los dispositivos de entrada y salida, en cambio, no lo son nada, son asíncronos y totalmente impredecibles. Pensad, por ejemplo, en el comportamiento de un usuario y el teclado.

Los dispositivos de entrada y salida son increíblemente variados en cuanto al comportamiento (hay de entrada, de salida, de almacenamiento), en cuanto al tipo (puede ser una máquina, un humano) y a la velocidad a la que se puede transferir la información entre el dispositivo y el procesador.
La necesidad de conectar adecuadamente toda esta diversidad de dispositivos al procesador implica que tiene que haber unos mecanismos que controlan y gestionan la comunicación y transferencia de información con ellos. De esto se encarga la unidad de entrada y salida del computador.
Las tres funciones principales de un sistema de entrada y salida (sistema de E/S) son la selección del dispositivo, la sincronización y la transferencia de datos entre el procesador y los dispositivos externos.
Hay que seleccionar el dispositivo ya que hay varios periféricos conectados al computador con funciones y velocidades distintas. Como la velocidad del procesador es muy superior a la de los periféricos hay que sincronizarlos antes de realizar el intercambio de datos. Y finalmente hay que realizar la transferencia de información

entre el computador y el periférico. En la figura 1 podéis observar un ejemplo de conexión del sistema de E/S.

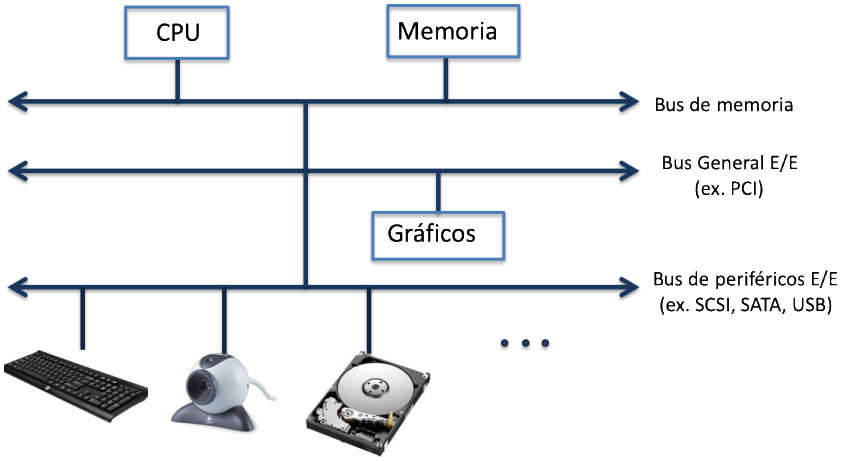


*Figura 1. Ejemplo de sistema de entrada y salida*

El procesador MIPS se comunica con los dispositivos de E/S utilizando una técnica llamada **E/S mapeada en memoria** (*memory mapped i/o).* Con esta técnica una porción del espacio de direccionamiento de la memoria se reserva para dedicarla a la comunicación con los dispositivos de E/S. El procesador cuando quiere atender a los dispositivos externos utilizará las instrucciones de acceso a la memoria `lw` y sw.
Otros procesadores utilizan instrucciones especiales para comunicarse con los periféricos, en este caso se denomina Entrada/Salida aislada.

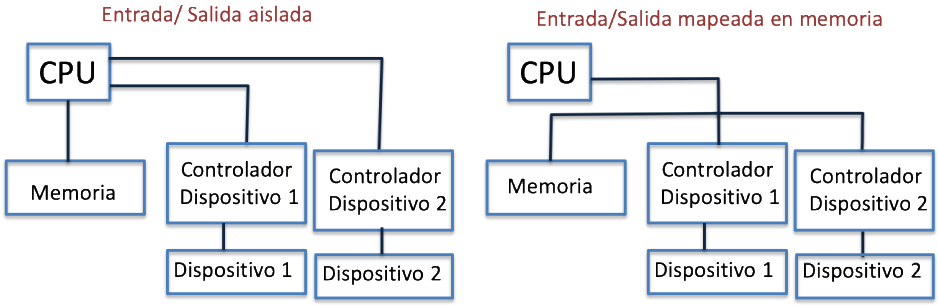


*Figura 2. Entrada/salida aislada versus Entrada/Salida mapeada en memoria*

Dos técnicas que estudiaremos para sincronizar los dispositivos de E/S con el procesador serán la sincronización por consulta de estado (o polling) y la sincronización por interrupciones.

En la sincronización por consulta de estado el procesador se encarga de controlar el estado del dispositivo periférico para hacer la transferencia de información. Esta tarea la hará comprobando dentro de un bucle el registro de control del periférico correspondiente. Cuando está preparado, el procesador inicia la transferencia. Normalmente esto provoca una pérdida de tiempo en los recursos del procesador si continuamente se ha de estar consultando si los dispositivos externos están preparados.
En el caso de la sincronización por interrupciones, básicamente se envía una señal externa cuando el dispositivo está preparado para la transferencia. Esto provoca que el

programa se interrumpa temporalmente y se ejecute una rutina de servicio de la interrupción. Esta rutina se encarga de comunicarse con el periférico, cuando finaliza su tarea se devuelve el control en el programa interrumpido.

## 2. Entrada/salida mapeada en memoria

Describiremos a continuación como el MIPS se comunica con los dispositivos E/S mediante E/S mapeada en memoria. Como hemos comentado antes hay un espacio de las direcciones de memoria que están dedicadas exclusivamente a la comunicación con los periféricos, el procesador ve a los periféricos como si de una memoria se tratara pero donde las direcciones físicas se corresponden realmente con los registros de los dispositivos de E/S. Dicho de otro modo, los dispositivos de E/S tienen internamente unos registros de comunicación, cada uno de ellos está representado por una posición de memoria. El procesador se comunica con ellos escribiendo y leyendo en las posiciones de memoria asociadas a esos registros.
Se debe tener con cuenta que el procesador está ejecutándose al mismo tiempo que los dispositivos de entrada y salida, por lo que ha de haber algún mecanismo para las dos cosas se puedan hacer sin problemas.
Cada registro de E/S tiene que tener unas direcciones de memoria propias asignadas para que el procesador direccione el registro del dispositivo específico. El MIPS reserva un espacio de la memoria comprendido entre la dirección 0xffff0000 y la 0xffff0010 para los dispositivos de E/S mapeada en la memoria conocida como MMIO (*Memory-mapped I/O)*. En la figura 3 se muestra la distribución de la memoria del MIPS ya comentada en la práctica 6. Esa es la configuración por defecto de la memoria del MIPS. El simulador MARS permite cambiar la configuración con el menú `Settings > Memory Configurations`.

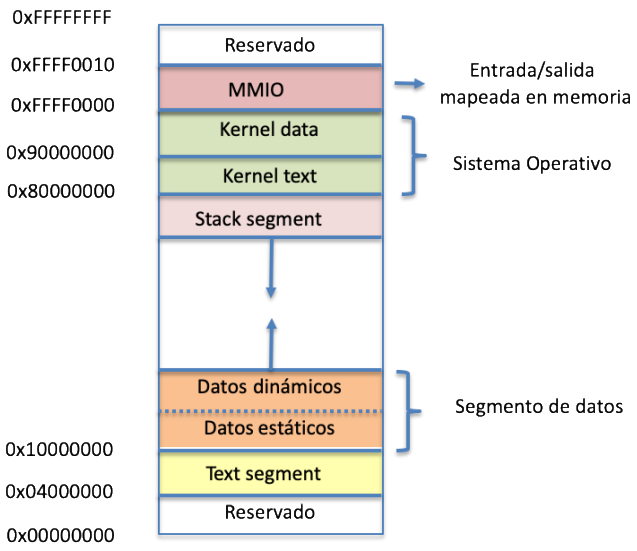


*Figura 3. Distribución de la memoria del MIPS*

Utilizando E/S mapeada en memoria no se tiene necesidad de añadir instrucciones adicionales para seleccionar los dispositivos. Con las instrucciones *lw* y *sw* con la dirección efectiva 0xffff0000 o mayor se accederá a los registros de los dispositivos de E/S.

## 3. MARS y los dispositivos externos

Un dispositivo genérico tiene un *registro de datos* y un *registro de control* para gestionar la comunicación entre el periférico y el procesador. El registro de control contiene el bit de *preparado* (*ready*) que indica si el dispositivo está preparado para la transmisión de la información. El dispositivo tiene la capacidad de modificar el *bit de control* y el procesador sólo lo podrá leer. El funcionamiento será el siguiente: el procesador comprueba si el dispositivo está preparado observando el bit de ready del registro de control. Si está preparado, la transferencia se hace leyendo o escribiendo en el registro de datos del dispositivo.

El MARS simula un único dispositivo de entrada/salida, un terminal mapeado en memoria donde el usuario puede leer y escribir caracteres.
El dispositivo consta de dos unidades separadas, un receptor y un transmisor. El *receptor* lee un carácter del teclado, y el *transmisor* escribe un carácter en la pantalla. Las dos unidades son completamente independientes, por lo tanto si tecleamos un carácter en el teclado no aparece automáticamente en la consola. Un programa es el que tiene que leer el carácter del receptor y escribirlo en el transmisor para que aparezca en la consola. Tanto el transmisor como el receptor constan de un registro de control y uno de datos de tamaño *palabra* mapeados en memoria. En particular los registros se encuentran en las posiciones de memoria 0xFFFF0000 hasta la 0xFFFF000C como se muestra en la figura 4.

Cuando el MIPS accede a una de estas direcciones para leer o escribir, el procesador realmente está accediendo para leer o escribir en el registro seleccionado de uno de los controladores de los dispositivos de entrada/salida.
Como se observa en la figura 4, sólo se utilizan dos bits de los registros de control. El bit 0 es el bit de *ready* que indica que el dispositivo está preparado cuando su valor es 1.

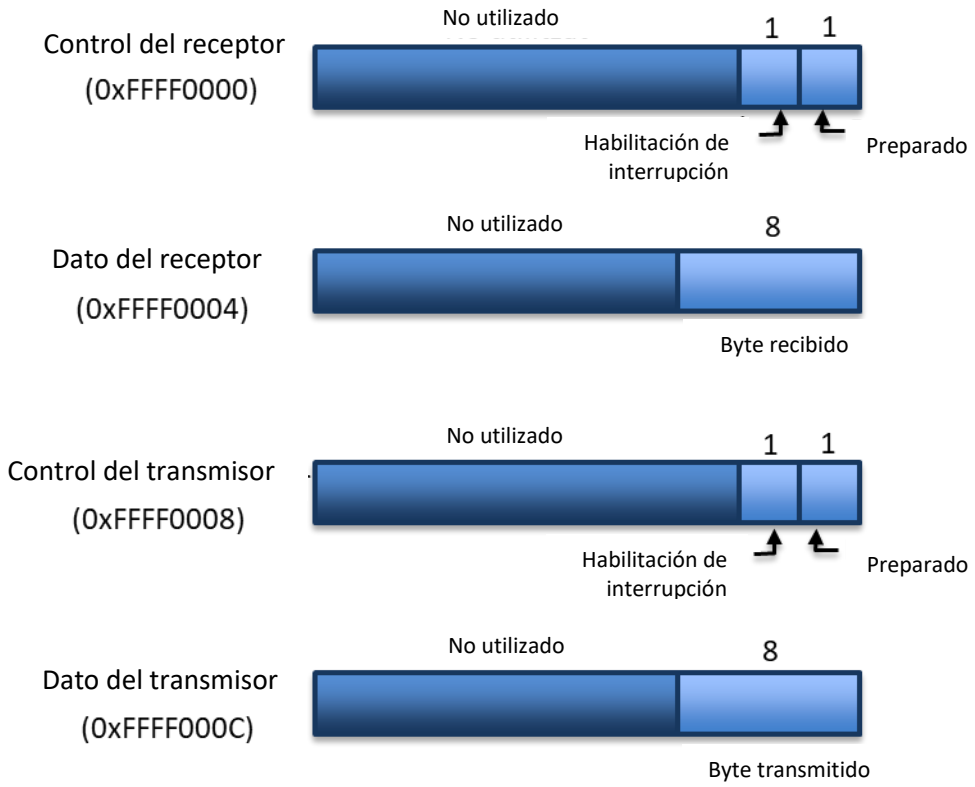


*Figura 4. Registros de control y datos del receptor y transmisor mapeados en memoria.*

Los programas se ejecutan normalmente en modo *usuario* pero para acceder a los registros de los dispositivos se requiere que se ejecutan en modo *kernel*. De este modo el sistema operativo se salvaguarda de que el usuario provoque algún tipo de error. Al utilizar el simulador MARS será como si nuestros programas formaron parte del *kernel* y se nos permitierá acceder a la MMIO directamente.

# Desarrollo de la práctica

## 1. MARS y su interfaz

El simulador del MARS nos proporciona la herramienta (`Tools > Keyboard and Display MMIO Simulator`) que simula el teclado y la pantalla como se ve en la figura 6. Para utilizarla con el simulador hay que pulsar la opción `Connect to MIPS` para conectarla. Esto activará la herramienta y permitirá comunicar el programa con el teclado y la pantalla carácter a carácter. En un caso, real el código que se comunica con los dispositivos externos a este nivel se conoce como *driver* del dispositivo. Mediante el simulador se nos permitirá comunicarnos directamente con ellos.

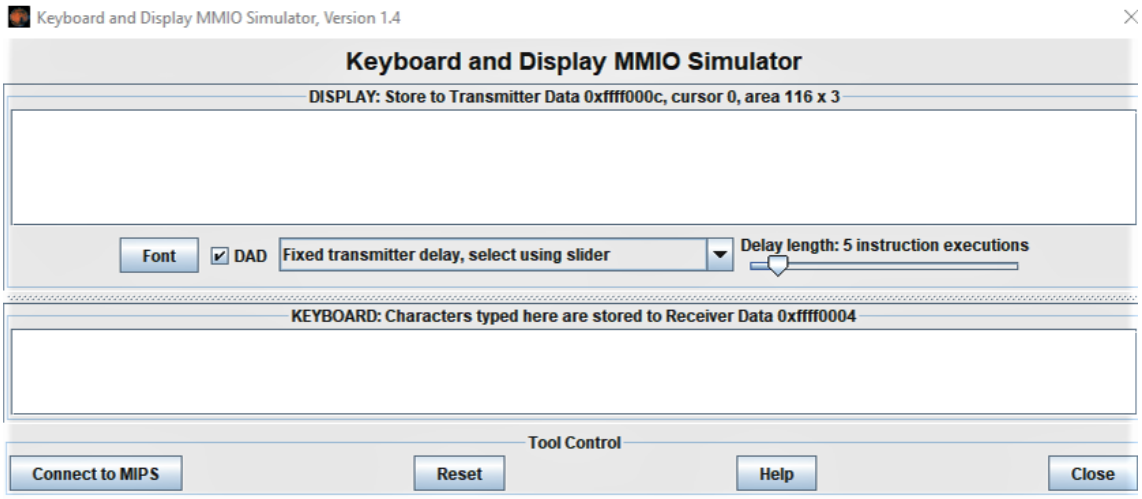


*Figura 6. Terminales teclado y pantalla mapeados en memoria.*

## 2. Funcionamiento del teclado.

Los dos registros asociados con el teclado son el registro de control de recepción mapeados en la dirección 0xffff0000 y el registro de datos mapeados en la 0xffff0004. Ambos registros son de 32 bits aunque sólo son útiles los bits menos significativos de los registros como se puede ver en la figura 4.

El proceso de comunicación con el teclado por consulta de estado o polling es el siguiente: El procesador lee el registro de control dentro de un bucle esperando que el dispositivo ponga a 1 el bit de *ready*, para indicar que está preparado para hacer la transferencia.

En el momento que se pulsa una tecla, el bit de *ready* se pone a 1 y se almacena el carácter (su código ASCII) en el byte de menor peso del registro de datos. Esta operación desconecta el bit de *ready* temporalmente hasta que la operación de entrada haya finalizado. El procesador detecta el cambio en el bit de *ready* y lee el carácter del registro de datos. Una vez leído el carácter el bit de *ready* vuelve al valor 0.

A continuación se muestra un código MIPS que muestra un ejemplo de acceso al control del teclado y al registro de datos en la memoria mapeada utilizando polling.

```
          #Leer del teclado
          #Carácter de entrada en $v0

  #SELECCIÓN:
   lui $t0, 0xffff # Direc. del registro de control del teclado
   li $t1, 0     #Inicia un contador de espera
b_espera:
          lw $t2, ($t0)     #Lee registro control del teclado

          #SINCRONIZACIÓN:
          andi $t2, $t2, 1     #Extrae el bit de ready
          addiu $t1, $t1, 1    #Incrementa el contador
                               #(cuenta las iteraciones)
          beqz $t2, b_espera   # Si cero no hay carácter
                               #continuamos esperando
          #TRANSFERENCIA:
          lw $v0, 4($t0)   #Lee registro de datos del teclado
                           #Codigo de tecla guardado en $v0
```

El código consulta el registro de control del teclado hasta que indica que hay un carácter (bit de *ready* a 1), entonces el programa lo lee. La frecuencia a la que se pulsan los caracteres del teclado es muy lenta comparada con la velocidad a la que el procesador puede ejecutar instrucciones. Típicamente millones de instrucciones se ejecutan hasta que la tecla se pulsa. El registro $t1 mantiene una traza del número de iteraciones en el bucle de espera

- Ensambla el programa pero antes de ejecutarlo pulsa en la opción `connect to MIPS` del simulador del teclado y pantalla.
- 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.
- Ejecuta de nuevo el programa pero ahora disminuye su velocidad de ejecución, por ejemplo a 15 instrucciones por segundo.
- Haz pruebas observando el segmento MMIO de la memoria.
- Observa el contador para comprobar la diferencia de velocidad del programa y el usuario.
- 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.
- Si algo va mal prueba con Reset para devolver el dispositivo a las condiciones de inicio.

## 3. Funcionamiento de la consola.

Los dos registros asociados con la consola son el registro de control de transmisión mapeado en la dirección 0xffff0008 y el registro de datos mapeado en la 0xffff000c. Al igual que con el teclado, sólo son útiles los bits menos significativos de los registros como se puede ver en la figura 4.

El proceso de comunicación con la consola por *polling* se puede hacer de manera similar al del teclado. El procesador lee el registro de control dentro de un bucle hasta que el bit de *ready* se ponga a 1, indicando que la consola está preparada para la transmisión. Entonces el procesador escribe el carácter a transmitir en el registro de datos y una vez escrito, el controlador de la consola pone el bit de *ready* a 0. Una vez mostrado el carácter en la consola se pone el bit de *ready* a 1 para indicar que está preparado para recibir un nuevo carácter. A continuación se muestra el fragmento de código que realiza el proceso de polling en la consola:

```
            #Escribir en la consola
            #Carácter de salida en $a0

            lui $t0,0xffff      #ffff0000; SELECCIÓN
b_espera:
            lw $t1,8($t0)       #registre control
            andi $t1,$t1,0x0001 #bit de ready SINCRONIZACIÓN
            beq $t1,$0,b_espera
            sw $a0,12($t0)      # TRANSFERENCIA
```

- Comprueba las similitudes de este código con el de leer del teclado.

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

Los computadores reales requieren tiempos para enviar caracteres a la consola o a un terminal. El MARS simula este retraso midiendo instrucciones ejecutadas en lugar de ciclos de reloj. Por ejemplo, una vez empieza el transmisor a escribir un carácter, el bit de *ready* permanecerá en 0 durante unos instantes. Esto significa que el transmisor no estará preparado hasta que no se hayan ejecutado un número fijado de instrucciones. Si se parara la máquina se observaría que el bit de *ready* permanecería a 0 y ya no cambiaría. Pero si la máquina continuara funcionando normalmente el bit de *ready* cambiaría de nuevo a 1 al cabo de unas cuántas instrucciones. Esta es la razón de desactivar la casilla DAD en el ejercicio anterior.

## 4. Practiquemos la entrada y salida

### 1. getc y putc.

Dado el siguiente fragmento de código:

```
.texto
 main:
            jal getc
            move $a0, $v0
            jal putc

 end:
            li $v0,10
            syscall
```

- Completadlo con las funciones *getc* (leer un carácter del teclado) y putc (escribir un carácter en teclado) vía *polling*.
- Probad su funcionamiento y comprobad cómo varían los contenidos de los registros y la memoria MMIO.

### 2. Programa *echo*

- Iterad el código anterior hasta que el carácter introducido sea un salto de línea (‘/’).
- Para probar el programa podéis activar previamente la casilla DAD, la cual controla el retraso en número de instrucciones en las que aparecerá el carácter en la consola desde que se escribe en el registro de datos.

## 5. Ejercicios a entregar:

- 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.

- 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.

```
.data

cadena:     .space 32

            .eqv ControlTeclado 0
            .eqv BufferTeclado 4
            .eqv ControlDisplay 8
            .eqv BufferDisplay 12

.text

            la $a0,cadena
            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
```

## Resumen

- El MIPS se comunica con los periféricos mediante E/S mapeada en memoria.
- Los dispositivos externos tienen un registro de control y uno de datos asociados a direcciones de memoria. El procesador se comunica con los periféricos leyendo y escribiendo los registros.
- La sincronización por consulta de estado está basada en la lectura del bit de *ready* dentro de un bucle a la espera que el dispositivo esté preparado para hacer la transferencia de información.