etsinf

Escola Tècnica
Superior d'Enginyeria
Informàtica

ESTRUCTURA DE COMPUTADORES (GII)

Ejercicios del tema 8

Mecanismos de sincronización de la Entrada/Salida

Profesores:

Ana PONT
Antonio ROBLES
José M. VALIENTE
José FLICH
Xavi MOLERO
Jorge REAL
Álvaro DOMENECH
Milagros MARTÍNEZ
Julio PONS

Curso 2017 - 2018

Tema 8

Mecanismos de sincronización de la E/S

8.1. Sincronización por consulta de estado

EJERCICIO 1. La figura 8.1 muestra las conexiones de la interfaz de un periférico que se conecta a una versión reducida del MIPS R2000 con sólo 16 bits de dirección y 8 de datos. Los registros de estado y control son de 8 bits. El circuito de selección consiste en una única puerta tipo NAND de 15 entradas a la que se conectan los bits de dirección desde A_1 hasta A_{15}.

Escriba un programa que espere, por consulta de estado, a que el bit de menor peso del registro de estado sea igual a 1, para entonces escribir un 0 en el registro de control.

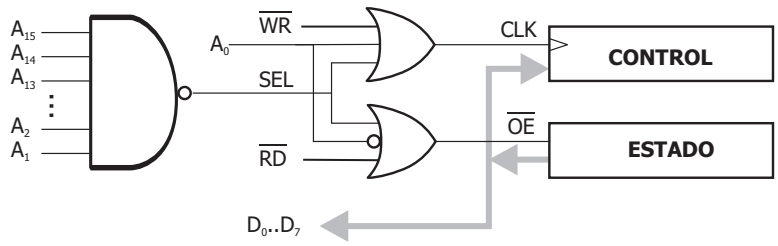

Figura 8.1: Conexiones de la interfaz del periférico (ejercicio 1).

SOLUCIÓN:

Según el circuito de selección (la puerta NAND), la dirección base de este dispositivo es la que tiene los bits de dirección des de A_1 hasta A_{15} a 1 y un cero en A_0. Se trata pues de la dirección $0xFFFE$.

Cuando A_0 vale 0 se habilita la lectura del registro de control, entonces su dirección es $0xFFFE$. Por su parte, el registro de estado se puede escribir sólo si $A_0 = 1$, con lo que su dirección es $0xFFFF$, lo que equivale a la dirección base más 1.

```
            .data
dir_base:   .word 0xFFFE

            .text
            la $t0, dir_base
espera:     lb $t1, 1($t0)
            andi $t1, $t1, 0x01
            beq $zero, $t1, espera
            sb $zero, 0($t0)
```

■

EJERCICIO 2. Un MIPS R2000 tiene conectados dos periféricos sencillos: un pulsador y una bombilla (vea la Figura 8.2). El estado del pulsador puede ser observado en el bit P que ocupa la posición 0 de un registro de 8 bits que se puede leer en la dirección 0xFFFF0000. Mientras el pulsador está presionado $P = 1$ y cuando está libre $P = 0$. La bombilla puede ser controlada mediante el bit L que ocupa la posición 1 de otro registro ubicado en la dirección 0xFFFF1000. Cuando $L = 1$ la bombilla se enciende y cuando $L = 0$ se apaga.

Figura 8.2: Los periféricos del ejercicio 2 y sus interfaces.

1. Explique qué hace este programa:

```
        la $t0,0xFFFF0000
        la $t1,0xFFFF1000
        sb $zero,0($t1)
b1:     lb $t2,0($t0)
        andi $t2,$t2,1
        beqz $t2,b1
        li $t2,2
        sb $t2,0($t1)
b2:     lb $t2,0($t0)
        andi $t2,$t2,1
        bnez $t2,b2
        sb $zero,0($t1)
        j b1
```

2. Escriba un programa que permita invertir el estado de la bombilla al presionar y liberar el pulsador. En este caso, invertir quiere decir que si está encendida se ha de apagar y si está apagada se ha de encender. Un esquema del programa podría ser este:

   apagar bombilla
   repetir
   esperar pulsador presionado
   esperar pulsador libre
   invertir el estado de la bombilla
   por siempre

SOLUCIÓN:

1. El programa enciende la bombilla mientras el pulsador está presionado y la deja apagada mientras el pulsador está libre. Veámoslo con más detalle:

   Primeramente, el programa inicializa el sistema, preparando las direcciones base de los dos periféricos y apagando la bombilla:

`la $t0,0xFFFF0000`	$t0 = dirección base del botón
`la $t1,0xFFFF1000`	$t1 = dirección base de la bombilla
`sb $zero,0($t1)`	apaga la bombilla

   Después, el programa se queda en un bucle de espera del que sólo saldrá cuando se presione el pulsador.

```
b1:   lb $t2,0($t0)      lee el registro con el estado del botón
      andi $t2,$t2,1     aisla el bit P
      beqz $t2,b1        si P == 0 vuelve a leer el estado
```

Al salir del bucle de espera el programa emite la orden de encender la bombilla. Primero prepara la orden en el registro $t2 y después la escribe en la interfaz de la bombilla.

```
      li $t2,2           $t2 = orden de encender la bombilla
      sb $t2,0($t1)      escribe la orden en el registro de la bombilla
```

A continuación, espera que se libere el pulsador para apagar la bombilla. Note que no hace falta preparar la orden de apagar en el registro $t2 porque el registro $zero ya la tiene.

```
b2:   lb $t2,0($t0)      lee el registro con el estado del botón
      andi $t2,$t2,1     aisla el bit P
      bnez $t2,b2        si P ≠ 0 vuelve a leer el estado
      sb $zero,0($t1)    escribe la orden de apagar en el registro de la bombilla
```

Y vuelve al principio:

```
      j b1
```

2. Traduciendo el esquema de programa al ensamblador, tenemos:

   Inicialización: escogemos $t2 para construir la orden que hay que dar a la bombilla. Su valor inicial será 0, y lo escribimos en la interfaz. Así, el programa comienza apagando la luz.

```
la $t0,0xFFFF0000
la $t1,0xFFFF1000
li $t2,0
sb $t2,0($t1)
```

   Luego espera a que se presione el pulsador:

```
b1:   lb $t3,0($t0)
      andi $t3,$t3,1
      beqz $t3,b1
```

   y luego espera que se libere:

```
b2:   lb $t3,0($t0)
      andi $t3,$t3,1
      bnez $t3,b2
```

   Finalmente, construye la orden. Si $t2 vale 0, es decir, si la luz estaba apagada, ha de encenderla haciendo $t2 = 2. Igualmente, si $t2 vale 2, es decir, si la luz estaba encendida, ha de apagarla haciendo $t2 = 0. Después de escribir la orden, el programa vuelve a comenzar.

```
xori $t2,$t2,2
sb $t2,0($t1)
j b1
```

   Otra solución supondría invertir el estado de la bombilla en el momento en que se presione el botón. El esquema de programa sería este:

   apagar bombilla
   repetir
       esperar pulsador presionado

invertir el estado de la bombilla
esperar pulsador libre
por siempre

En ensamblador:

```
        la $t0,0xFFFF0000
        la $t1,0xFFFF1000
        li $t2,0
        sb $t2,0($t1)
b1:     lb $t3,0($t0)
        andi $t3,$t3,1
        beqz $t3,b1
        xori $t2,$t2,2
        sb $t2,0($t1)
b2:     lb $t3,0($t0)
        andi $t3,$t3,1
        bnez $t3,b2
        j b1
```

■

EJERCICIO 3. Considere un periférico TEMP_SENSE, que consiste en un sensor de temperatura ambiente y cuya interfaz tiene tres registros: registro de control, registro de estado y registro de datos, que se muestran en la figura 8.3 y cuya descripción se incluye más adelante. Esta interfaz está conectada a un sistema basado en MIPS R2000, siendo su dirección base $DB = 0x0700FFF0$. La interfaz también dispone de una conexión a línea de interrupción $\overline{Int2}$ del MIPS.

El funcionamiento del sensor es el siguiente: Cuando cambia la temperatura al menos un grado la almacena en el registro de datos, y activa los bits RDY y ERROR. Si el bit IE está activo solicita interrupción al procesador. La interfaz se considera atendida e inactiva el bit RDY cuando se escribe el bit CL del registro de control.

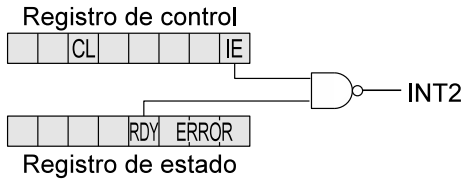

Figura 8.3: Registros de control y estado (ejercicio 3).

REGISTRO DE CONTROL (Dir: DB - 8 bits - escritura):

IE (bit 0): '1' Interrupciones habilitadas, '0' inhabilitadas.

CL (bit 5): '1' Bit Clear: borra el bit RDY. Cuando las interrupciones están activas tiene como efecto que la línea de interrupción se desactiva.

REGISTRO DE ESTADO (Dir: DB - 8 bits - lectura):

RDY (bit 3): '1' Interfaz preparada (hay nueva temperatura), '0' No preparada (no hay nueva temperatura). Además, si está preparado y el bit IE del registro de control está activo, se emite petición de interrupción $\overline{Int_2}$.

ERROR (bit 2, 1 y 0): ’000’ Cuando no hay error en la medida, valor entre ’001’ y ’111’ cuando ha ocurrido un error en la medida, tipificando el error de acuerdo a unas tablas internas.

REGISTRO DE DATOS (Dir: DB+4 - 8 bits - lectura):

TEMP (bits 0..7): Valor de temperatura registrado por el sensor (0 - 255 grados)

Se pretende escribir una rutina de atención (*leer_temp*) a esta interfaz que realice la sincronización mediante la técnica de consulta del estado. La rutina de atención debe leer el contenido de los bits de ERROR del registro de estado y, si son cero, leer el registro de datos y almacenar su contenido en la variable del sistema TEMP. Si los bits de ERROR son distintos de cero, la rutina debe almacenarlos en la variable del sistema ERROR, y no actualizar TEMP. No olvide que para que el bit RDY cambie a cero y pueda la interfaz mostrar otra medida, se deberá escribir un ’1’ en el bit CL.

Las variables del sistema asociadas, asi como las primeras instrucciones del programa se muestran a continuación:

```
          .kdata
TEMP:     .byte 0
ERROR:    .byte 0

           .ktext
__start:   mfc0 $t0, $12
            andi $t0, $t0, 0xFBFF
            mtc0 $t0, $12
            li $t0, 0x0700FFF0
            sb $zero, 0($t0)
            .....
            jal leer_temp    # Llama a la subrutina 'leer_temp'
            ....
            ....
            .end
```

1. Indique cuál es el propósito de las primeras líneas del programa.
2. Escriba el código de la subrutina *leer_temp*.

SOLUCIÓN:

1. Las primeras instrucciones del programa se usan para enmascarar la línea $\overline{Int_2}$ en el MIPS R2000, así como para poner el bit IE=0 (Interrupt Enable) en la interfaz del sensor. Ambas acciones inhabilitan la interrupción.
2. Código de rutina de atención por consulta del estado:

```
leer_temp: li $t0, 0x0700FFF0     # Puntero a BASE
bucle:     lb $t1, 0($t0)         # Acceso a ESTADO
           andi $t2, $t1, 0x08    # Filtro bit RDY (bit 3)
           beq $t2, $zero, bucle

           andi $t1, $t1, 0x07    # Filtro bits ERROR (bits 2,1,0)
           bne $t1, $zero, error
```

```
                lb $t1, 4($t0)          # Acceso a DATOS
                sb $t1, TEMP            # Almaceno en TEMP
                j fin

        error:  sb $t1, ERROR           # Almaceno ERROR
        fin:    li $t1, 0x20            # bit CL = 1, IE = 0
                sb $t1, 0($t0)
                jr $ra                  # Retorno de subrutina
```

■

8.2. Sincronización por interrupciones

EJERCICIO 4. Considere el periférico TEMP_SENSE del ejercicio 3. Se pretende ahora gestionar dicho periférico mediante el mecanismo de sincronización por interrupción. Parte del código de inicialización del sistema de excepciones se muestra a continuación:

```
          .kdata
TEMP:     .byte 0
ERROR:    .byte 0

           .ktext
__start:   mfc0 $t0, $12
            andi $t0, $t0, 0xFBFF
            mtc0 $t0, $12
            li $t0, 0x0700FFF0
            sb $zero, 0($t0)
            ....
            ....
            .end
```

1. Modifique el código de inicialización para habilitar la atención a la interrupción $\overline{Int_2}$, tanto en el procesador como en el periférico.
2. Escriba el código de la rutina de servicio de la interrupción $\overline{Int_2}$.

SOLUCIÓN:

1. Las primeras instrucciones del código de inicialización del sistema de excepciones deben desenmascarar la interrupción $\overline{Int_2}$ en el MIPS R2000, así como para poner el bit IE=1 (Interrupt Enable) en la interfaz del sensor.

```
            .ktext
__start:    mfc0 $t0, $12
             ori $t0, $t0, 0x0401 #  desenmascara el bit 10, asociado a INT2
             mtc0 $t0, $12        #  y habilita interrupciones del procesador
             li $t0, 0x0700FFF0
             li $t1, 0x01         # Habilita interrupciones en la interfaz
             sb $t1, 0($t0)
             ....
             ....
             .end
```

2. Código de rutina de servicio de la interrupción $\overline{Int_2}$:

```
int2:     li $t0, 0x0700FFF0     # Puntero a BASE
          lb $t1, 0($t0)         # Acceso a ESTADO
          andi $t1, $t1, 0x07    # Filtro bits ERROR (bits 2,1,0)
          bne $t1, $zero, error
          lb $t1, 4($t0)         # Acceso a DATOS
          sb $t1, TEMP           # Almaceno en TEMP
          j fin
error:    sb $t1, ERROR          # Almaceno ERROR
fin:      li $t1, 0x21           # bit CL = 1, IE = 1
          sb $t1, 0($t0)         # cancela la interrupción
          b retexec              # Retorno de excepción
```

■

EJERCICIO 5. Considere de nuevo el periférico *TEMP_SENSOR* del ejercicio 3. Suponga ahora que se conecta dos de estos sensores a un sistema MIPS. La dirección base del sensor_0 es 0xFFFFFF00 y la del sensor_1 es 0xFFFFFF10. Ambos sensores se conectan a la misma línea de interrupción $\overline{Int_2}$. Esta línea se activará cuando alguno o ambos sensores activen la interrupción. La rutina de servicio de dicha interrupción deberá averiguar cuál se ellos ha interrumpido y almacenar la temperatura actual de dicho sensor en las variables del sistema Temperatura_0 o Temperatura_1. Se asume que el sensor_0 es el más prioritario.

```
              .kdata
Temperatura_0:  .byte 0
Temperatura_1:  .byte 0
```

1. Escriba el código de inicio del sistema de excepciones que debe habilitar la interrupción $\overline{Int_2}$, dejando el resto de interrupciones igual que estaban.
2. Escriba el código de servicio de la interrupción $\overline{Int_2}$.
   (Nota: Se pueden usar los registros $t0, $t1 y $t2).

SOLUCIÓN:

1. Código inicialización del sistema de interrupciones:

```
          .text
          .globl __start

__start:
                                  ## Código de inicialización
           ......
           la $t0,0xFFFFFF00      # $t0 = dirección base sensor_0
           li $t1, 0x01
           sb $t1, 0($t0)         # IE = 1
           la $t0,0xFFFFFF10      # $t0 = dirección base sensor_1
           li $t1, 0x01
           sb $t1, 0($t0)         # IE = 1
           mfc0 $t0, $12
           ori $t0, $t0, 0x0403   # desenmarcarar int_2, modo USER
           mtc0 $t0, $12          # e Interrupt Enable = 1
           .....
```

2. El código de la rutina de servicio de la interrupción 2 comprueba cuál de los dos sensores ha interrumpido, comenzando por el sensor 0 que es el más prioritario.

```
int_2:      la $t0,0xFFFFFF00       # $t0 = dirección base sensor_0
            lb $t1, 0($t0)          # registro de estado
            andi $t1, $t1, 0x08     # bit R ready
            beq $t1, $0, leer_s_1
            lb $t1, 4($t0)
            sb $t1, temperatura_0   # Guarda temp del sensor_0
            li $t1, 0x81
            sb $t1, 0($t0)          # Clear int_2
            j retexc
leer_s_1:   la $t0,0xFFFFFF10       # $t0 = dirección base sensor_1
            lb $t1, 4($t0)
            sb $t1, temperatura_1   # Guarda temp del sensor_1
            li $t1, 0x81
            sb $t1, 0($t0)          # Clear int_2
            j retexc
```

■

8.3. Interrupciones y funciones del sistema

EJERCICIO 6. Un computador con procesador MIPS R2000 tiene conectados dos periféricos *A* y *B* con las siguientes interfaces:

- El periférico *A* sólo contiene dos botones *A1* y *A2* independientes. Su interfaz consta de los dos registros de 8 bits, *Estado* y *Órdenes*, que se muestran en la figura 8.4. El registro de estado sólo admite lectura y el de órdenes sólo escritura. Los dos registros tienen la misma dirección `0xFFFFCC00`. En todo momento, los bits *A1* y *A2* del registro de estado indican a 1 si el botón correspondiente se encuentra pulsado y a 0 el caso contrario. El bit *R* pasa a valer 1 cuando alguno de los bits *A1* o *A2* cambia de 0 a 1 y vuelve a valer 0 cuando se escribe 1 en el bit *OK* del registro de órdenes. Finalmente, si se escribe 1 en el bit *IE*, el adaptador del periférico activa la línea de interrupción $\overline{Int_3}$ cuando $R = 1$.

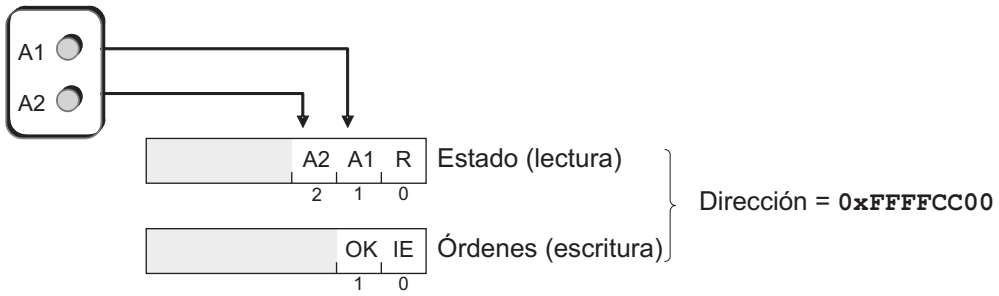

Figura 8.4: Interfaz del periférico *A* (ejercicio 6).

- El periférico *B* es una lámpara controlada mediante la interfaz de la figura 8.5. El único bit practicable de la interfaz (*ON*) puede ser leído y escrito mediante el registro de estado y órdenes ubicato en `0xFFFFDD00`. Si se escribe $ON = 1$, la lámpara se enciende, en caso contrario se apaga. La lectura de este registro proporciona el último valor que se ha escrito.

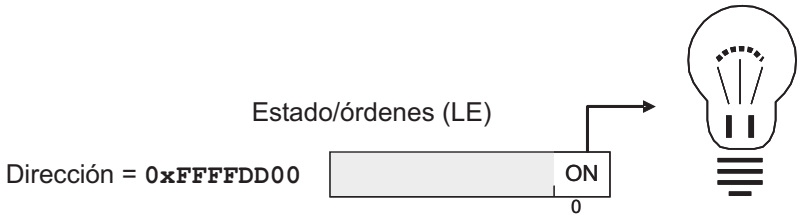

Figura 8.5: Interfaz del periférico *B* (ejercicio 6).

1. Observa el programa siguiente y describe el comportamiento del sistema:

```
            la $t0,0xFFFFCC00
            la $t2,0xFFFFDD00

bucle1:     lb $t1,0($t0)
            andi $t1,$t1,4
            beq $t1,$zero,bucle1

            li $t1,1
```

```
        sb $t1,0($t2)

buble2: lb $t1,0($t0)
        andi $t1,$t1,4
        bne $t1,$zero,bucle2

        sb $zero,0($t2)

        j bucle1
```

2. Escribe un programa que, por consulta de estado del bit *R*, controle los periféricos para que el sistema tenga el comportamiento siguiente:

   Al pulsar el botón *A1*, la lámpara se ha de encender (esté como esté previamente) y ha de mantenerse encendida hasta que, al pulsar el botón *A2*, la lámpara se apague. Pulsar los dos botones al mismo tiempo no ha de tener ningún efecto.

3. Considera ahora que las interrupciones están habilitadas en la interfaz de *A*. Escribe el tratamiento que, dentro del manejador de excepciones, ha de aplicarse a la interrupción $\overline{Int_3}$. El comportamiento ha de ser el descrito en el apartado 2. Puedes utilizar los registros *$t0* a *$t3*.

4. Se ha incluido en el manejador las funciones 3297 y 3298 que pueden utilizar los programas de usuario por medio de llamada al sistema. La función 3297 permite a los programas encender y apagar la lámpara *B*. La función 3298 permite a los programas consultar el estado de la lámpara. Su especificación es la siguiente:

Función	Índice ($v0)	Parámetros entrada	Parámetros de salida
Encender_lamp	3297	$a0 = 1:encendido 0:apagado	—
Estado_lamp	3298	—	$v0 = 1:encendido 0:apagado

   Escribe un programa de usuario que llame a estas funciones para invertir el estado de la lámpara: si está apagada hay que encenderla; si está encendida hay que apagarla.

SOLUCIÓN:

1. Comportamiento del sistema: El computador, por consulta de estado, hace que s encienda la luz *B* mientras se pulsa el botón correspondiente al bit *A2*.

2. Ejercicio de consulta de estado:

   Una versión con dos bucles de sincronización.

```
        la $t0,0xFFFFCC00
        la $t3,0xFFFFDD00

        # la luz está apagada
bucle1: lb $t1,0($t0)   # espera R = 1
        andi $t2,$t1,1
        beq $t2,$zero,bucle1

        li $t2,2        # hace OK = 1
        sb $t2,0($t0)

        andi $t2,$t1,4  # si A2 = 1 continúa esperando
```

```
        bne $t2,$zero,bucle1

encen:  li $t2,1        # hace ON = 1
        sb $t2,0($t3)

        # la luz está encendida
bucle2: lb $t1,0($t0)   # espera R = 1
        andi $t2,$t1,1
        beq $t2,$zero,bucle2

        li $t2,2        # hace OK = 1
        sb $t2,0($t0)

        andi $t2,$t1,2  # si A1 = 1 continúa esperando
        beq $t2,$zero,bucle2

apaga:  sb $zero,0($t3) # hace ON = 0
j bucle
```

Otra versión con sólo un bucle de sincronización.

```
        la $t0,0xFFFFCC00
        la $t3,0xFFFFDD00

bucle:  lb $t1,0($t0)   # espera R = 1
        andi $t2,$t1,1
        beq $t2,$zero,bucle

        li $t2,2        # hace OK = 1
        sb $t2,0($t0)

        li $t2,5        # analiza A2 y A1
        beq $t2,$t1,apaga
        li $t2,3
        beq $t2,$1,enciende
        j bucle

enciende:
        li $t2,1        # hace ON = 1
        sb $t2,0($t3)
j bucle

apaga:  sb $zero,0($t3) # hace ON = 0
j bucle
```

3. El tratamiento de las interrupciones se puede hacer fácilmente a partir de la versión de consulta de estado que se sincroniza con un bucle de espera. Sólo habrá que eliminar el bucle y sustituir los saltos dirigidos a la etiqueta `bucle` por saltos a la etiqueta `retexc` de final del manejador

```
int3:   la $t0,0xFFFFCC00
        la $t3,0xFFFFDD00

        lb $t1, 0($t0)  # lee registro estado
        li $t2,2        # hace OK = 1
        sb $t2,0($t0)
```

```
        li $t2,5        # analiza A2 i A1
        beq $t2,$t1,apaga
        li $t2,3
        beq $t2,$t1,enciende
        j retexc

enciende:
        li $t2,1        # hace ON = 1
        sb $t2,0($t3)
j retexc

apaga:  sb $zero,0($t3) # hace ON = 0
j retexc
```

4. Llamadas al sistema para invertir el estado de la luz:

```
# llama a la función que consulta el estado
  li $v0,3298
  syscall
# ahora tiene en $v0 el estado de la luz
# calcula el parámetro para invertir el estado
  xori $a0,$v0,1
# llama a la función que fija nuevo estado
  li $v0,3297
  syscall
```

■

EJERCICIO 7. Un sistema empotrado controla la temperatura de una planta comercial. Este sistema está compuesto por un computador basado en un procesador MIPS R2000 conectado a una interfaz que consta de un termómetro, un calefactor (bomba de calor) y un refrigerador (bomba de frío). El funcionamiento que se desea conseguir es mantener la temperatura en torno a un valor que se programará inicialmente. Los registros que muestra la interfaz son cuatro (todos de 8 bits):

CONTROL_ESTADO (escritura/lectura, dirección base `0xF9000010`)

R: (bit 0) se activa a 1 cuando la temperatura cambia un grado.

CL: (bit 6) a 1 tiene el efecto de poner R a 0 (cancelando la interrupción si la hay).

IE: (bit 7) a 1 habilita la interrupción. Si $IE = 1$, la interrupción $\overline{Int_0}$ se emitirá cuando $R = 1$.

CONTROL_MOTOR (sólo escritura, dirección base + 4)

CA: (bit 0) a 1 activa el calefactor (bomba de calor), a 0 lo apaga.

FI: (bit 7) a 1 activa el refrigerador (bomba de frío), a 0 lo apaga.

TEMPERATURA (lectura/escritura, dirección base + 8) Contiene el valor de la temperatura en grados centígrados, codificada como entero con signo (complemento a dos).

1. Escriba el código de inicio del manejador de excepciones en el MIPS R2000 que debe habilitar las interrupciones generales, poner el procesador en modo usuario, habilitar la línea de interrupción $\overline{Int_0}$ dejando el resto de interrupciones igual que estaban, y acceder a la interfaz para habilitar las interrupciones. El contenido del registro de estado está descrito en la figura A.1.

2. Escriba el código de la rutina de tratamiento de la línea de interrupción $\overline{Int_0}$. Su cometido es leer el registro de temperatura (valor con signo) y almacenarlo en la variable del sistema `temperatura` que se ha definido como se muestra a continuación. No olvide cancelar la petición de interrupción. Se pueden emplear los registros temporales `$t0` y `$t1`.

```
                .kdata
temperatura:    .byte 0
```

3. Escriba el tratamiento de las funciones del sistema que tienen el siguiente perfil:

Función	Índice (`$v0`)	Parámetros entrada	Parámetros salida
`leer_temp`	33	-	`$a0` = temperatura
`activar_calor`	34	-	
`activar_frío`	35	-	
`parar`	36	-	

   Las funciones anteriores tienen el siguiente comportamiento:

   - `leer_temp` carga en registro `$a0` el contenido de la variable del sistema `temperatura`.
   - `activar_calor` activa el calefactor de la interfaz (bomba de calor) y apaga el de frío.
   - `activar_frío` activa el refrigerador de la interfaz (bomba de frío) y apaga el de calor.
   - `parar` apaga el calefactor y el refrigerador; si ya estaban apagados no tiene ningún efecto sobre la interfaz.

4. Codifique una rutina que, utilizando las llamadas al sistema anteriores, reproduzca las acciones indicadas por el siguiente pseudocódigo:

```
si (temp > temp_máx) entonces "activar frío"
si no si (temp < temp_mín) entonces "activar calor"
      si no  "parar frío y calor"
      fsi
fsi
```

   La variable *temp* indica la temperatura medida por la interfaz, se deberá acceder a su valor mediante la llamada al sistema apropiada. Las temperaturas de referencia `temp_máx` y `temp_mín` son variables definidas de la siguiente manera:

```
            .data
temp_max:   .byte 25
temp_min:   .byte 17
```

SOLUCIÓN:

1. Inicialización general de interrupciones.

```
inicializa:     la $t0, 0xF9000010      # Base de la interfaz
                li $t1, 0x80            # bit IE (bit 7) a "1"
                sb $t1, 0($t0)          # activa interrupciones en la interfaz
```

```
                mfc0 $t0, $12          # lectura del registro estado coprocesador 0
                ori $t0, $t0, 0x103    # activar int0  + int generales +  modo usuario
                mtc0 $t0, $12          # escritura del registro estado
```

2. Rutina de tratamiento de la interrupción $\overline{Int_0}$.

```
int0:           la $t0, 0xF9000010     # Base de la interfaz
                li $t1, 0xC0           # bit IE (bit 7) a "1", y CL (bit 6) a "1"
                sb $t1, 0($t0)         # cancelar interrupción, manteniendo activación

                lb $t1, 8($t0)         # leer temperatura
                sb $t1, temperatura    # actualizar variable del sistema
                j retexc
```

3. Llamadas al sistema,

```
leer_temp:      la $t0, temperatura   # variable del sistema
                sb $a0, 0($t0)        # $a0 = temperatura
                j retexc

activar_calor:  la $t0, 0xF9000010   # Base de la interfaz
                li $t1, 1            #  bit CA = "1"
                sb $t1, 8($t0)       # activa calor, parar frio
                j retexc

activar_frio:   la $t0, 0xF9000010   # Base de la interfaz
                li $t1, 0x80         #  bit FI = "1"
                sb $t1, 8($t0)       # activa frio, parar calor
                j retexc

parar:          la $t0, 0xF9000010   # Base de la interfaz
                sb $0, 8($t0)        #  bit FI = "0" y CA="0"
                j retexc
```

4. Rutina que implementa la comprobación de la temperatura entre márgenes.

```
                .data
temp_max:  .byte 25   # temperatura referencia maxima
temp_min:  .byte 17   # temperatura referencia minima

                .text
rutina   :      li $v0, 33
                syscall               #  $a0 = temperatura

                lb $t0, temp_max      #  Margen superior
                bgt $a0, $t0, frio    #  Si temperatura > margen: frio

                lb $t0, temp_min      #  Margen inferior
                blt $a0, $t0, calor   #  Si temperatura < margen: calor

                li $v0, 36            #  Dentro margenes: parar
                j fin

frio:           li $v0, 35            #  activa el calor, para el frio
                j fin
```

```
calor:      li $v0, 34          #  activa el frio, para el calor
            j fin

fin:        syscall             #  realiza la función apropiada
            jr $ra
```

■

EJERCICIO 8. Tienes que diseñar el sistema de acceso al aparcamiento la universidad. Cada acceso incluye tres dispositivos: un punto de inserción de tarjetas magnéticas, una barrera motorizada y un sensor de paso de coches. Las tarjetas magnéticas son de tres categorías: de estudiante, de plantilla y de VIP. Los tres dispositivos van conectados a un computador con MIPS R2000 mediante las interfaces que se describen a continuación (figura 8.6):

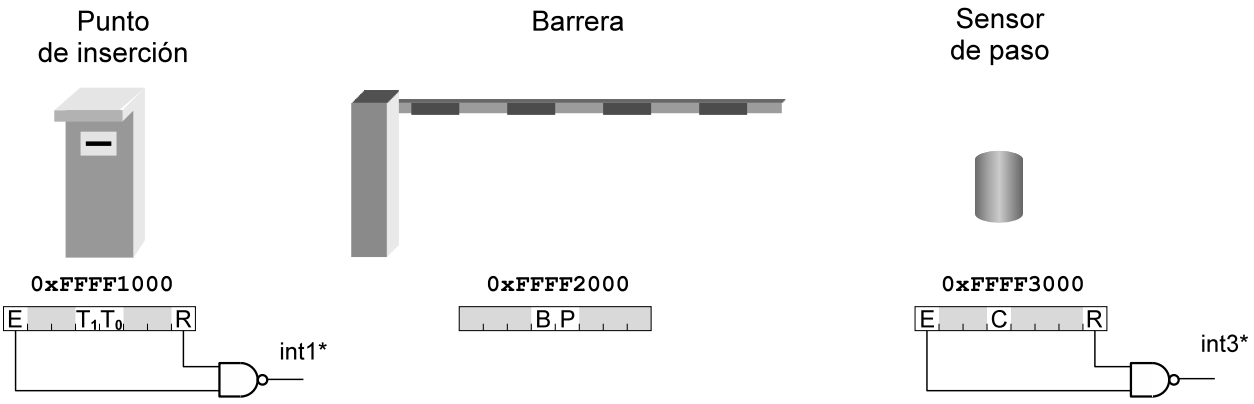

Figura 8.6: Dispositivos para la entrada del aparcamiento (ejercicio 8).

Punto de inserción: Contiene un único registro que combina estado y órdenes en la dirección 0xFFFF1000. Los bits relevantes que contiene son:

	Campo	Bits	Acceso	Función
R	Preparado	0	Lectura	Se hace R=1 al insertar una tarjeta
T	Tipo	4-3	Lectura	Tipo de tarjeta : 00 (estudiantes), 01 (plantilla), 10 (VIP)
E	Habilitación	7	Escritura	E = 1 habilita las interrupciones

Cuando se inserta una tarjeta, R pasa a valer 1 y el par de bits T indica el tipo de tarjeta. Además, si E = 1 se produce la interrupción $\overline{Int_1}$. R cambia a 0 al leer o escribir el registro.

Barrera: Contiene un registro de órdenes en la dirección 0xFFFF2000.

	Campo	Bits	Acceso	Función
P	Abrir	3	Escritura	P=1 sube la barrera
B	Cerrar	4	Escritura	B=1 baja la barrera

Para levantar la barrera hay que escribir un 1 en P y para bajarla hay que escribir un 1 en B. Escribir 1 en los dos bits no produce ningún efecto. Igualmente, escribir 0 en los dos bits tampoco produce ningún efecto. Las órdenes inician el movimento de levantar o bajar; la barrera se detiene al final del recorrido y así queda hasta que se da la orden contraria.

Sensor de paso: Contiene un único registro que combina estado y órdenes en la dirección 0xFFFF3000.

	Campo	Bits	Acceso	Función
R	Preparado	0	Lectura	Se hace R=1 a pasar un coche
C	Cancelación	4	Escritura	Escribir C = 1 hace R = 0
E	Habilitación	7	Escritura	E = 1 habilita las interrupciones

El bit R del sensor de paso cambia a 1 cuando pasa el coche. Además, si E = 1 se produce la interrupción $\overline{Int_3}$. R vuelve a 0 cuando se escribe un 1 en el bit C.

El manejador incluye este tratamiento para la interrupción del punto de inserción:

```
int1:   la $t0,0xFFFF1000
        lb $t1,0x0($t0) # cancelo interrupción
        la $t0,0xFFFF2000
        li $t1,0x08
        sb $t1,0($t0)
        j retexc # salta al final del manejador
```

1. Explica lo que hace este fragmento de programa que se ejecuta en modo supervisor.

```
la $t0,0xFFFF1000
sb $zero,0($t0)
la $t0,0xFFFF2000
li $t1,0x10
sb $t1,0($t0)
la $t0,0XFFFF3000
sb $zero,0($t0)
```

2. El vicerrector de Fomento de la universidad quiere que sólo se levante la barrera cuando la tarjeta sea del tipo VIP. Modifica el tratamiento de la interrupción $\overline{Int_1}$ para seguir estas directrices.

3. Escribe el tratamiento de la interrupción $\overline{Int_3}$ para que se baje la barrera al pasar un coche. Además, el tratamiento ha de contar los coches que entran incrementando la variable definida en el segmento de memoria `.ktext`:

```
coches: .word 0
```

4. Escribe el tratamiento de dos funciones de sistema con la especificación siguiente:

Servicio	Código	Parámetros de entrada	Parámetros de salida
`clear_counter`	$v0 = 900	—	—
`read_counter`	$v0 = 901	—	$v0 = número de coches

Estas funciones dan acceso a la variable `coches` definida en el apartado 3. La función *clear_counter* inicia la variable a 0 y la función *read_counter* devuelve su valor.

SOLUCIÓN:

1. El programa cancela las interrupciones y baja la barrera.

2. Tratamiento de $\overline{Int_1}$ para que solamente entren VIP:

```
int1:   la $t0,0xFFFF1000
        lb $t1,0($t0)
        andi $t1,$t1,0x18   # aisla bits de tipo de tarjeta
        li $t0,0x10         # t0 = tipo VIP
        bne $t0,$t1,retexc  # si tipo tarjeta != VIP, salta a final
        la $t0,0xFFFF2000
        li $t1,0x08
        sb $t1,0($t0)
        j retexc # salta al final del manejador
```

3. Tratamiento de $\overline{Int_3}$:

```
int3:   la $t0,0xFFFF3000
        li $t1,0x90
        sb $t1,0x0($t0)
        la $t0,0xFFFF2000
        li $t1,0x10
        sb $t1,0($t0)
        lw $t1,cotxes
        addi $t1,$t1,1
        sw $t1,coches
        j retexc # salta al final del manejador
```

4. Funciones 900 y 901:

```
fun900: sw $zero,coches
        j retexc # salta al final del manejador

fun901: lw $v0,coches
        j retexc # salta al final del manejador
```

■

EJERCICIO 9. Considere el periférico *KeyPad*, que consiste en un teclado numérico con teclas del 0 al 9 y un visualizador de 7 segmentos, como se muestra en la figura 8.7.

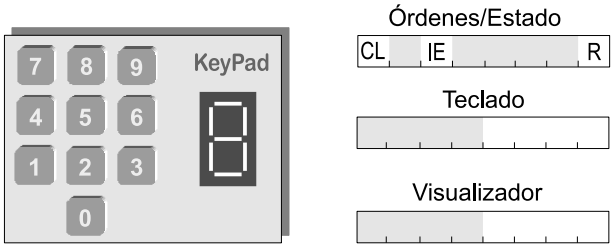

Figura 8.7: Aspecto de *KeyPad* y de su interfaz (ejercicio 9).

Este dispositivo tiene la interfaz siguiente:

REGISTRO DE ÓRDENES Y ESTADO (8 bits, lectura / escritura, Dirección base + 0)

R: (Ready), bit de solo escritura. A 1 indica que se ha pulsado una tecla y que el registro de datos del teclado contiene un valor nuevo.

IE: (Interrupt Enable), bit de lectura / escritura. Hacer $IE = 1$ habilita las interrupciones. Mientras $IE = 1$ y $R = 1$, el periférico activa la línea de petición de interrupción.

CL: (Clear), bit de solo escritura. Hacer $CL = 1$ fuerza el bit $R = 0$.

REGISTRO DE DATOS DEL TECLADO (8 bits, lectura, Dirección base + 4)

bits 3...0: Código de tecla (del 0 al 9).

REGISTRO DE DATOS DEL VISUALIZADOR (8 bits, escritura, Dirección Base + 4)

bits 3...0: Si aquí se escribe un valor del 0 al 9, el visualizador muestra el dígito correspondiente. Un valor -1 (*0xF*) enciende solamente el segmento central. El resto de valores no están definidos.

Si el bit *IE* está a 1, el dispositivo activa una línea de petición de interrupción cuando se pulsa una tecla de *KeyPad* y la desactiva si el procesador escribe un 1 en el bit *CL* del registro de órdenes. El visualizador de segmentos está siempre preparado y, por lo tanto, no tiene ninguna relación con las interrupciones. El controlador se conecta a un computador basado en el MIPS R2000. La dirección base escogida es `0xFFFFFF00`. La petición de interrupción está conectada a la línea $\overline{Int_2}$ del procesador.

1. Complete el diseño del circuito de selección de la interfaz del *KeyPad* que se muestra en la figura 8.8. La parte que falta (selección de registro) tiene cuatro salidas: $OEst_{out}$, $OEst_{clk}$, Tec_{out} y Vis_{clk}. La señal de entrada *BE** que aparece en la figura es la línea de habilitación (a nivel bajo) del byte menos significativo de las palabras.

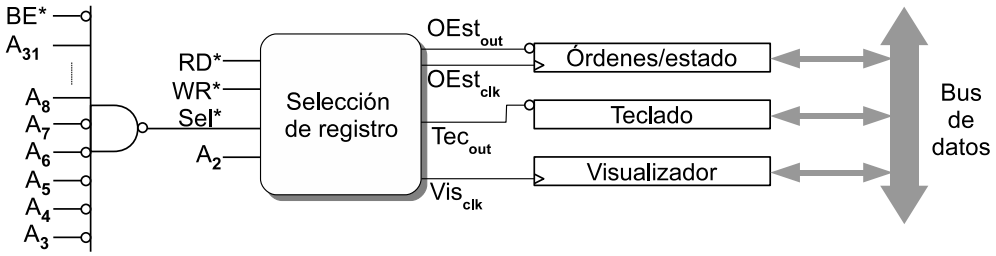

Figura 8.8: Circuito de selección de *KeyPad* (ejercicio 9).

2. Considere que, inicialmente, el teclado tiene las interrupciones inhibidas. ¿Qué hace el programa siguiente cuando se ejecuta en modo supervisor? Añada los comentarios apropiados a las instrucciones y las explicaciones necesarias.

```
        la t0,0xFFFFFF00
bucle:  lb $t1,0($t0)
        andi $t1,$t1,1
        beqz $t1,bucle
        li $t1,0x80
        sb $t1,0($t0)
        lb $t1,4($t0)
        andi $t1,t1,0xF
        li $t2,5
        bne $t1,$t2,bucle
        li $t1,0xA0
        sb $t1,0($t0)
```

3. Escriba el código de tratamiento del *KeyPad* que responda a una petición de interrupción hardware. Este tratamiento ha de almacenar el código de la tecla pulsada en la variable del manejador *KP_Key* (de tipo *byte*), pero no ha de operar sobre el visualizador. Explique su código con los comentarios apropiados.

4. Escriba el código de inicialización del *KeyPad* para que el tratamiento del apartado 3 sea correcto. Este código ha de inicializar la variable *KP_Key* a -1, encender el segmento central del visualizador, habilitar la interrupción en la interfaz del periférico y preparar la máscara apropiada en el registro de estado del MIPS R2000. Explique su código con los comentarios apropiados.

5. Escriba el código de un servicio del sistema con el perfil siguiente:

Servicio	Código	Parámetros de entrada	Parámetros de salida
`read_KP`	$v0 = 11	—	$v0 = último número tecleado

Este servicio ha de tener dos efectos: ha de entregar al programa el valor almacenado en la variable *KP_Key* por el tratamiento del apartado 3 y ha de mostrarlo en el visualizador de *KeyPad*. Explique su código con los comentarios apropiados.

6. A continuación está el tratamiento de la función 513 y un tratamiento de la interrupción $\overline{Int_2}$ distinto del descrito en el apartado 3:

```
fun513: sb $a0,KP_Key
        jal suspende_este_proceso
        j retexc

int2:   la $t0,0xFFFFFF00
        li $t1,0xA0
        sb $t1,0($t0)
        lb $t1,4($t0)
        lb $t2,KP_Key
        bne $t1,$t2,retexc
        jal activa_proc_en_espera
        j retexc
```

Explique brevemente qué hace el servicio 513 y escriba su especificación con el estilo con que se ha definido el servicio 11 en el apartado 5.

SOLUCIÓN:

1. Diseño del circuito de selección de los registros: En la figura 8.9 (izquierda) aparece la solución convencional con un decodificador de una entrada. También se puede diseñar el circuito directamente con puertas. En función de la dirección de cada registro y del tipo de acceso (lectura o escritura) con el que han de operar, podemos determinar qué valores de las cuatro variables de entrada activan cada uno de las cuatro señales de operación:

Sel*	A_2	RD*	WR*	Eixida activa
0	0	0	X	$OEst_{out}$
0	0	X	0	$OEst_{clk}$
0	1	0	X	Tec_{out}
0	1	X	0	Vis_{clk}

Teniendo en cuenta que las señales de salida son activas por nivel bajo, se pueden obtener las expresiones algebraicas de cada una (por exemplo, $OEst_{out} = Sel^* + A_2 + RD^*$ i $Vis_{clk} = Sel^* + \overline{A_2} + WR^*$). Vea el circuito resultante en la figura 8.9 (derecha).

2. El código espera por consulta de estado a que se pulse la tecla "5". Entonces, habilita las interrupciones en la interfaz de *KeyPad*. Con detalle:

```
        la $t0,0xFFFFFF00
bucle:  lb $t1,0($t0)     # Bucle de consulta de estatdo
        andi $t1,$t1,1    #   espera hasta teclado preparado
        beqz $t1,bucle
        li $t1,0x80       # Reinicia el teclado
        sb $t1,0($t0)
```

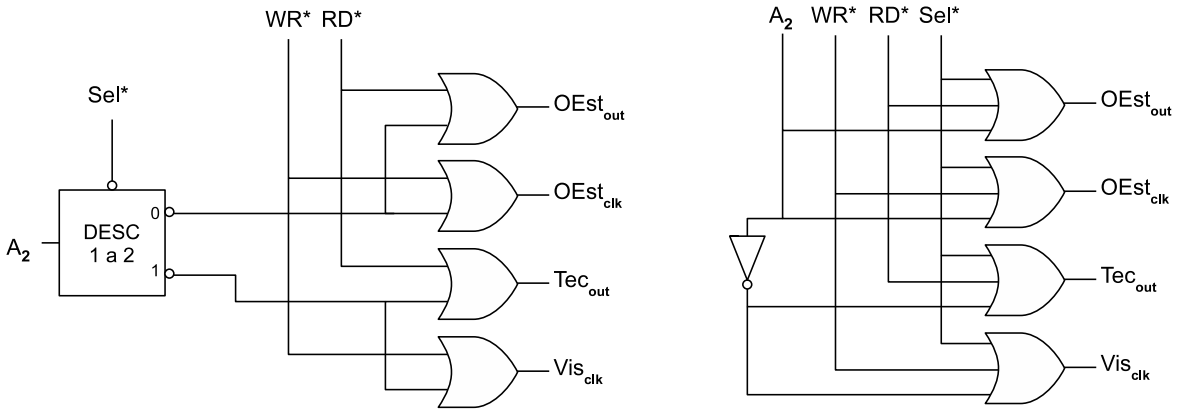

Figura 8.9: Dos versiones del circuito de selección de los registros de la interfaz, (ejercicio 9, apartado 1).

```
        lb $t1,4($t0)     # Lee código de tecla
        andi $t1,t1,0xF
        li $t2,5          # Compara código con "5"
        bne $t1,$t2,bucle # Si código != "5", vuelve a comenzar
        li $t1,0xA0       # Si código == "5", reinicia teclado y
        sb $t1,0($t0)     #     habilita las interrupciones
```

3. Tratamiento de la interrupción de la línea int_2^*:

```
int2:   la $t0,0xFFFFFF00
        lb $t1,4($t0)    # lee código de tecla
        sb $t1,KP_Key    # actualiza variable
        li $t1,0xA0
        sb $t1,0($t0)    # conserva IE=1 y cancela interrupción
        j retexc
```

4. Inicialización del sistema completo:

```
    # inicializa KeyPad:
        la $t0,0xFFFFFF00
        li $t1,0x20      # hace IE = 0
        sb $t1,0($t0)    # en la interfaz:
    # inicializa variables
        li $t1,-1        # asigna KP_Key = -1
        sb $t1,KP_Key
    # inicializa el coprocesador de interrupciones
        li $t1,0x0403    # habilita int2 y IEc
        mtc0 $t1,$12
```

5. Código del servicio 11 del sistema:

```
fun11:  la $t0,0xFFFFFF00
        lb $v0,KP_Key    # $v0 = último código de tecla
        sb $v0,4($t0)    # hace copia en el visualizador
        j retexc
```

6. La función 513 permite a los programas esperar hasta que alguien pulse la tecla especificada como argumento. Cuando un programa de usuario llama a la función 513, el tratamiento hace dos cosas: almacena el argumento (*$a0*) en la variable *KP_Key* y suspende el proceso. La

interrupción de *KeyPad* lo reactivará si el código de la tecla coincide con la variable. Cmb detalle:

```
int2:   la $t0,0xFFFFFF00
        li $t1,0xA0       # Cancela interrupción
        sb $t1,0($t0)
        lb $t1,4($t0)     # Lee tecla
        lb $t2,KP_Key     # Lee variable
        bne $t1,$t2,retexc # si no son iguales, acaba
     ## Si tecla == KP_Key, reactiva el proceso en espera:
        jal activa_proc_en_espera
        j retexc
```

La especificación es la siguiente:

Servicio	Código	Parámetros de entrada	Parámetros de salida
`wait_KP`	$v0 = 513	—	$a0 = tecla que hay que esperar

■

EJERCICIO 10. Considere un punto de venta donde los clientes piden turno cuando llegan y los dependientes atienden por orden. El sistema irá controlado por un computador con tres periféricos (véase la figura 8.10):

- Un **expendedor de turno** formado por un botón y una impresora que escribe un número en un tiquet. La interfaz comprende tres registros de 1 byte:

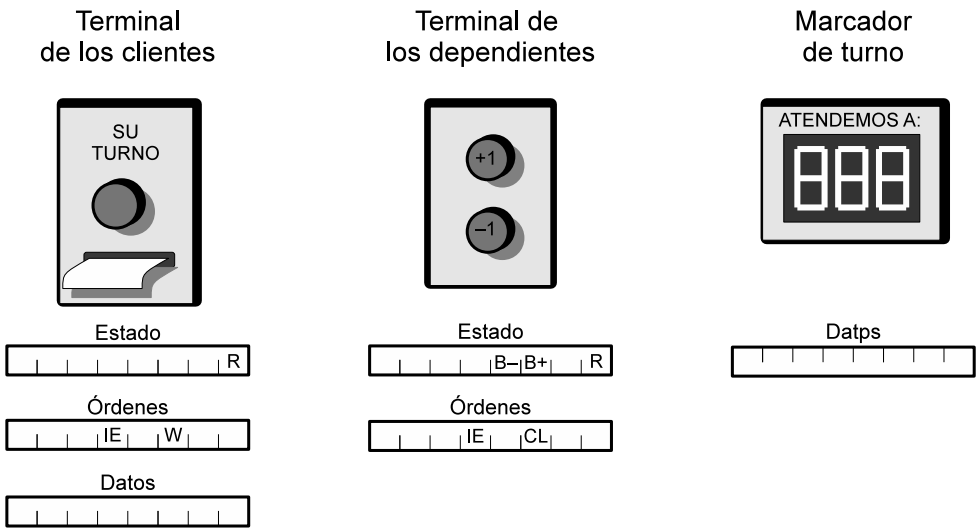

Figura 8.10: Periféricos del punto de venta (ejercicio 10).

Estado (sólo lectura, dirección base + 0)

R: (bit 0) se activa cada vez que un cliente pulsa el botón y la impresora está preparada.

Órdenes (sólo escritura, dirección base + 4)

IE: (bit 4) a 1 habilita la interrupción. Si $IE = 1$, la interrupción se emitirá cuando $R = 1$.

W: (bit 2) tiene dos efectos: hace $R = 0$ (cancelando la interrupción si hay) y activa la impresora.

Datos (sólo escritura, dirección base + 8): el valor numérico (entre 0 y 255) que se imprime al dar la orden W.

- Una **botonera** para los dependientes con dos botones rotulados "+1" y "–1".

  Estado (sólo lectura, dirección base + 0):

  R: (bit 0) vale 1 cuando alguien pulsa un botón cualquiera.

  B+: (bit 2) vale 1 mientras el botón rotulado "+1" está presionado.

  B–: (bit 3) vale 1 mientras el botón rotulado "–1" está presionado.

  Ordres (sólo escritura, dirección base + 4):

  IE: (bit 4) a 1 habilita la interrupción. Si $IE = 1$, la interrupción se emitirá cuando $R = 1$.

  CL: (bit 2) fuerza el bit $R = 0$ (cancelando la interrupción si está activa).

- Un **marcador** que está siempre preparado. Su interfaz sólo contiene un registro de datos de 8 bits donde se escribe el valor numérico que ha de aparecer en el visualizador de tres dígitos.

El hardware ignora el valor de los bits no descritos en las interfaces anteriores. Los tres periféricos están conectados a un computador con MIPS R2000. En la tabla siguiente tiene los detalles:

Periférico	Dirección base	Línea de interrupción
Expendedor para los clientes	`0xFFFF0010`	$\overline{Int_1}$
Botonera de los dependientes	`0xFFFF8040`	$\overline{Int_4}$
Marcador	`0xFFFFB000`	—

El manejador de excepciones del sistema contiene dos variables relevantes: `clientes` indica cuántos clientes han pedido turno y `servicio` a qué número se está atendiendo.

1. Escriba un programa de prueba del expendedor que se limite a esperar por consulta de estado que se pulse el botón de pedir turno e imprima un tiquet con el número "255". Este programa, útil para el diagnóstico del sistema, se ejecutará en modo supervisor y podrá acceder sin restricciones a la interfaz de los periféricos.

2. Observe este tratamiento de la interrupción $\overline{Int_4}$ dentro del manejador. Explique qué hace.

```
int4:     la $t0,0xFFFF8040
          li $t1,0x14
          sb $t1,4($t0)
          lb $t1,0($t0)
          li $t2,0xD
          bne $t1,$t2,L0
          sb $zero,clientes
          sb $zero,servicio
          la $t0,0xFFFFB000
          sb $zero,0($t0)
          j retexc
L0:       # aquí ampliará más adelante
          j retexc
```

3. Complete el tratamiento del apartado 2 a partir de la etiqueta `L0` para que, al pulsar uno de los dos botones del terminal, se incremente o decremente la variable `servicio` y su valor resultante se muestre en el marcador. Ignore la posibilidad de desbordamiento, porque nunca vendrán más de 50 clientes en un día cualquiera, y por la noche se apaga el sistema.

4. Escriba el tratamiento correspondiente a la interrupción $\overline{Int_1}$. Cuando un cliente pulse el botón del expendedor, hay que incrementar la variable `clientes` e imprimir en el tiquet su valor.

5. Escriba el tratamiento de la función de sistema `get_clients` que tiene este perfil:

Servicio	Índice	Parámetros de salida
`get_clients`	$\$v0 = 666$	$\$v0$ = Número de turnos dados

   Suponga que el manejador salta a la etiqueta `fun666` cuando la causa de excepción es la instrucción `syscall` y `$v0` $= 666$. Sólo ha de escribir el código apropiado a partir de esta etiqueta.

SOLUCIÓN:

1. Programa de prueba del expendedor.

```
            la $t0,0xFFFF0010
            li $t1,0x04
            sb $t1,4($t0)
consulta:   lb $t2,0($t0)  # bucle de consulta de estado
            andi $t2,$t2,1
            beqz $t2,consulta
            li $t2,0xFF  # deposita 255 en registro de datos
            sb $t2,8($t0)
            sb $t1,4($t0) # orden de escritura y cancelación
            j consulta
```

2. Tratamiento inicial de la interrupción $\overline{Int_4}$. El tratamiento sólo atiende al caso que estén pulsados ambos botones, e inicializa las variables `clientes` y `servicio` a 0.

3. Tratamiento completo de la interrupción $\overline{Int_4}$.

```
L0:       li $t2,0x8
           bne $t1,$t2,L1
           # tratamiento de B+
           lb $t1,servicio
           addi $t1,$t1,+1
           sb $t1,servicio
           la $t0,0xffffb000
           sb $t1,0$(t0)
           j retexc
L1:        # tratamiento de B-
           lb $t1,servicio
           addi $t1,$t1,-1
           sb $t1,servicio
           la $t0,0xffffb000
           sb $t1,0$(t0)
           j retexc
```

4. Tratamiento de la interrupción $\overline{Int_1}$.

```
int1:    la $t0,0xFFFF0010
         lb $t1,clientes
         addi $t1,$t1,1
         sb $t1,clientes
         sb $t1,8($t0)
         li $t1,0x14
         sb $t1,4($t0)
         j retxec
```

5. Tratamiento de la función del sistema `get_clients`.

```
fun666:  lb $v0,clientes
         j retxec
```

■

EJERCICIO 11. Un horno dispone de dos espacios A y B separados por una pared (vea la figura 8.11, parte izquierda). Los dos espacios se comunican por un ventilador que hace circular el aire. Para control de la temperatura, se han instalado dos termómetros T_A y T_B, uno en cada espacio.

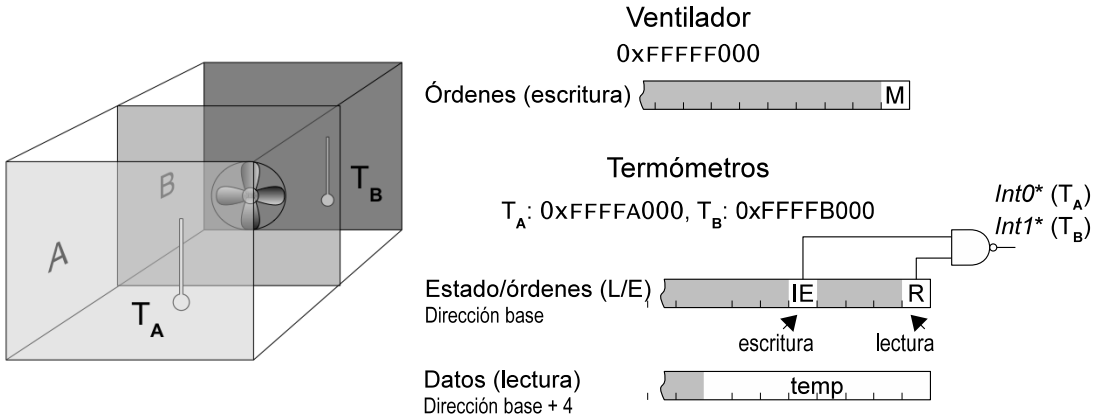

Figura 8.11: Esquema del horno y de las interfaces del ventilador y de los termómetros (ejercicio 11).

El ventilador y los termómetros están controlados por un computador mediante adaptadores con las interfaces mostradas en la figura 8.11 (derecha) que detallamos a continuación:

Ventilador: El registro de órdenes (sólo escritura, ubicado en la dirección base de la interfaz) contiene el bit M para controlarlo. Escribir $M = 1$ activa el ventilador y $M = 0$ lo para.

Termómetros: Las interfaces de T_A y T_B son idénticas y contienen dos registros: estado / órdenes y datos:

- El registro de estado / órdenes (ubicado en la dirección base) contiene dos bits significativos: R (sólo lectura) en la posición 0 y el bit IE (sólo escritura) en la posición 4
- El bit R, de periférico preparado, toma el valor $R = 1$ cuando cambia la temperatura y el valor $R = 0$ cuando hay un acceso de lectura al registro de datos

- El bit *IE* habilita las interrupciones. Si $IE = 1$, la interfaz activa la línea de interrupción a la que está conectada mientras $R = 1$.
- El registro de datos (lectura) contiene el valor de la temperatura en los 8 bits menos significativos. Se trata de un valor sin signo, de manera que el rango de representación es de $[0 \ldots 255]$ ºC.

Los adaptadores están conectados al computador con las direcciones base de la tabla 8.1; además, los adaptadores de los termómetros están conectados a las líneas de interrupción $\overline{Int_0}$ y $\overline{Int_1}$.

Periférico	Dirección base	Línea de interrupción
Termómetro T_A	*0xFFFFA000*	$\overline{Int_0}$
Termómetro T_B	*0xFFFFB000*	$\overline{Int_1}$
Ventilador	*0xFFFFF000*	—

Tabla 8.1: Conexión de los periféricos del problema 11

Suponiendo que la UCP del computador es un MIPS R2000:

1. Escriba un programa (que se ejecutará en modo supervisor) que espere, por consulta de estado, que $T_A > 100^oC$ para activar el motor.

2. Escriba el tratamiento de las interrupciones de los termómetros apropiado para poner en marcha el ventilador si las temperaturas en los espacios *A* y en *B* son diferentes y lo pare si las temperaturas son iguales. Escriba sólo el código que hay que incluir en el manejador a partir de las etiquetas `int0` y `int1` hasta la instrucción `j retexc`. Si necesita definir alguna variable del manejador, escriba también su declaración en el segmento de datos `kdata`.

3. Escriba el código de la llamada al sistema `stop_fan` que detiene el ventilador y deshabilita las interrupciones de los termómetros.

4. Explique qué hace la llamada al sistema `functionxxx` si su código y el tratamiento de la interrupción $\overline{Int_0}$ son estos:

```
functionxxx:
            jal suspende_este_proceso
            j retexc

int0:       la $t0, 0xFFFFA000
            li $t1, 0x10
            sb $t1, 0($t0)
            lbu $t1, 4($t0)
            li $t0, 200
            bne $t0, $t1, retexc
            jal activa_proc_en_espera
            j retexc
```

SOLUCIÓN:

1. Programa que espera, por consulta de estado, que $T_A > 100^oC$ para activar el ventilador

```
# Cargo dirección base y deshabilito interrupciones
            la $t0, 0xFFFFA000
```

```
            sb $0, 0($t0)
# Consulta del estado
espera:     lb $t1, 0($t0)
            andi $t1, $t1, 1
            beqz $t1, espera
# Comprueba valor de temperatura
            lbu $t1, 4($t0)
            li $t2, 100
            sub $t1, $t1, $t2
            blez $t1, espera
# Activa el ventilador
            la $t0, 0xFFFFF000
            li $t1, 1
            sb $t1, 0($t0)
# Final
            li $v0, 10
            syscall
```

2. Código dentro del manejador:

   Variables:

```
tempA:      .byte 0
tempB:      .byte 0
```

   Tratamiento de las interrupciones.

```
int0:       la $t0, 0xFFFFA000    # dirección base del termómetro
            lbu $t1, 4($t0)       # lee temperatura (y cancela)
            sb $t1, tempA         # almacena valor en la variable
            lbu $t0, tempB        # compara con última temperatura
            beq $t0, $t1, parar   #    del otro termómetro
            j ventilar

int1:       la $t0, 0xFFFFB000
            lbu $t1, 4($t0)
            sb $t1, tempB
            lbu $t0, tempA
            beq $t0, $t1, parar

ventilar:   la $t0, 0xFFFFF000    # dirección base del ventilador
            li $t1, 1
            sw $t1, 0($t0)        # hace bit M=1
            j retexc

parar:      la $t0, 0xFFFFF000
            sw $0, 0($t0)         # hace bit M=0
            j retexc
```

3. Código de la llamada al sistema `stop_fan`

```
stop_fan:   la $t0, 0xFFFFA000
            sw $0, 0($t0)
            la $t0, 0xFFFFB000
            sw $0, 0($t0)
            la $t0, 0xFFFFF000
            sw $0, 0($t0)
            j retexc
```

4. La llamada al sistema `functionxxx` permite a los programas esperar hasta que la temperatura en el recinto A sea $T_A = 200^{o}C$.

   El código que trata la llamada `stop_fan` se limita a poner el programa en estado de espera. El tratamiento de la interrupción $\overline{Int_0}$ lee la temperatura del termómetro y sólo si $T_A = 200$ cambia el estado del programa a preparado.

■

EJERCICIO 12. Se dispone de un termómetro a partir del cual se debe implementar la funcionalidad de un termógrafo sobre un sistema basado en el MIPS R2000. Un termógrafo es un dispositivo que permite almacenar valores de temperatura registrados a lo largo del tiempo para su procesamiento posterior. El termómetro presenta la interficie de la figura 8.12.

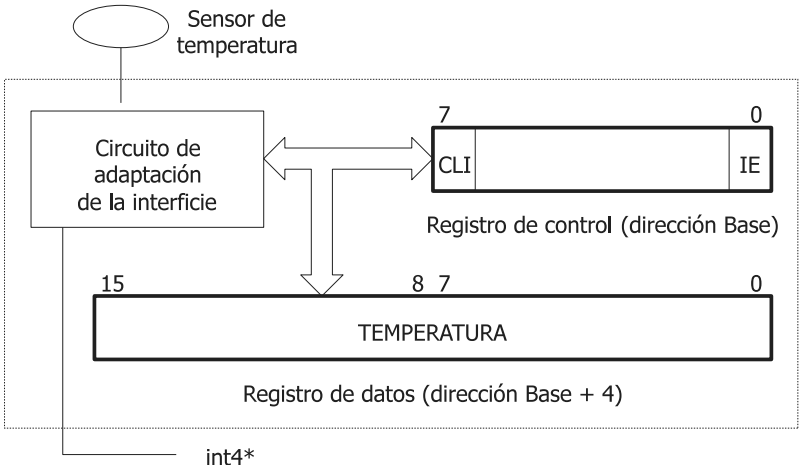

Figura 8.12: El termómetro y su interfaz (ejercicio 12).

La dirección base en la que se ubica el termómetro es la `0xFFFFFF00`. El registro de datos contiene la temperatura actual, representada mediante un valor de 16 bits (*half word*). El registro de control es solo de escritura y contiene dos bits de interés (el resto de bits son ignorados por el hardware):

E : *Interrupt Enable*. Cuando está a 1, el termómetro provoca una interrupción cada 10 minutos por la línea $\overline{Int4}$.

LI : *CLear Interrupt*. Cuando se escribe un 1 en este bit, el termómetro desactiva la línea de petición de interrupción. Esta acción debe hacerse cada vez que se atiende la interrupción para que el termómetro pueda producir una nueva interrupción posteriormente.

El termógrafo se implementa a través de una serie de funciones del sistema. Cada vez que se produce la interrupción del termómetro ($\overline{Int4}$) el sistema debe obtener el valor de temperatura actual y agregarlo a la primera posición libre de un vector de valores de temperatura. Las estructuras de datos asociadas a la gestión de este vector en el sistema son:

```
                .kdata
                ...
temperaturas:   .half 0,0,0,...0 # Un número suficiente de valores
num_temps:      .word 0          # Número de temperaturas registradas
                ...
```

1. Implemente la rutina de atención a la interrupción $\overline{Int4}$. Además de las acciones necesarias para la gestión del hardware, se debe leer el nuevo valor de temperatura y almacenarlo en la primera posición libre del vector de temperaturas. La variable `num_temps` debe incrementarse adecuadamente. Dado que cada valor de temperatura ocupa dos bytes, la primera posición libre del vector puede obtenerse como `num_temps` multiplicado por dos (o sumado a sí mismo) más el desplazamiento `temperaturas`. Se supone que el vector de temperaturas es ilimitado, por lo tanto no se pide implementar la comprobación de límites del vector.

A fin de que un programa de usuario pueda acceder a los datos almacenados en el termógrafo, debe proporcionarse la siguiente función de sistema:

Servicio	Índice ($v0)	Parámetros de entrada	Parámetros de salida
get_temps	31416	$*a*0 = Dirección vector de usuario	$*v*0 = Número medidas devueltas

A la función `get_temps` se le pasa como parámetro un puntero a la zona de memoria del programa de usuario, donde éste debe disponer de espacio suficiente para almacenar los valores de temperatura que se puedan haber acumulado desde la última vez que se llamó a la función. La función almacenará los elementos del vector `temperaturas` a partir de esa dirección. En el registro `$v0` retornará el número de valores de temperatura que se han copiado al espacio de usuario. La función debe tener además el efecto de vaciar el vector de temperaturas del sistema, simplemente poniendo a cero el contador `num_temps`.

2. Implemente la función del sistema `get_temps`. Solo debe aportar el código propio de la función, ignorando la parte de identificación de función y retorno. Puede utilizar los registros `$t0`, `$t1`, `$t2` y `$t3`.

3. Además de `get_temps`, suponga que hay una función (`wait_n_temps`) que, cuando se invoca, pone al proceso de usuario a la espera de que el termógrafo registre *n* medidas. Su perfil es el siguiente:

Servicio	Índice ($v0)	Parámetros de entrada	Salida
wait_n_temps	31417	$a0 = Número de datos por los que se esperará	—

Implemente un programa de usuario que, cuando se hayan registrado 50 temperaturas, las almacene en un vector del tamaño adecuado. Por simplicidad, puede suponer que este programa de usuario es el único que accede a las funciones del termógrafo.

SOLUCIÓN:

Apartado 1

```
int4: li $t0,0xFFFFFF00         # Dirección Base del termómetro
      lh $t1,4($t0)             # Leer temperatura
      lw $t2,num_temps          # Leer número de temperaturas
      add $t3,$t2,$t2           # Obtener 2*num_temps
      sh  $t1,temperaturas($t3) # Añadir nueva temperatura al vector
```

```
        addi $t2,$t2,1
        sw $t2,num_temps          # Incrementar num_temps
        li $t1,0x81
        sb $t1,0($t0)             # CLI = 1; IE = 1
        b retexc
```

Apartado 2

```
get_temps:     lw $v0,num_temps($0) # $v0 = num_temps
               or $t0,$v0,$0        # $t0 = número de medidas acumuladas
               la $t1,temperaturas  # $t1 apunta a inicio vector kernel
               or $t2,$a0,$0        # $t2 apunta a inicio vector usuario
         bucle: beq $t0,$0,fin_get_temps
               lh $t3,0($t1)        # Copiar temperatura sobre $t3
               sh $t3,0($t2)        # Copiar temp. a espacio de usuario
               addi $t0,$t0,-1      # Decrementar contador de medidas
               addi $t1,$t1,2       # Mover puntero vector kernel
               addi $t2,$t2,2       # Mover puntero vector usuario
               b bucle
fin_get_temps: b retexc
```

Apartado 3

```
            .data
mis_temps:  .half 0,0,0...,0  # 50 posiciones

            .text
            ...
            li $v0,31417
            li $a0,50
            syscall           # Llamada a wait_n_temps
            li $v0,31416
            la $a0,mis_temps
            syscall           # Llamada a get_temps
            ...
```

■

Apéndice A

El sistema de excepciones del MIPS R2000

La figura A.1 muestra el detalle del registro de estado del coprocesador de excepciones del MIPS R2000. La figura A.2 muestra la relación entre los registros de estado y de causa.

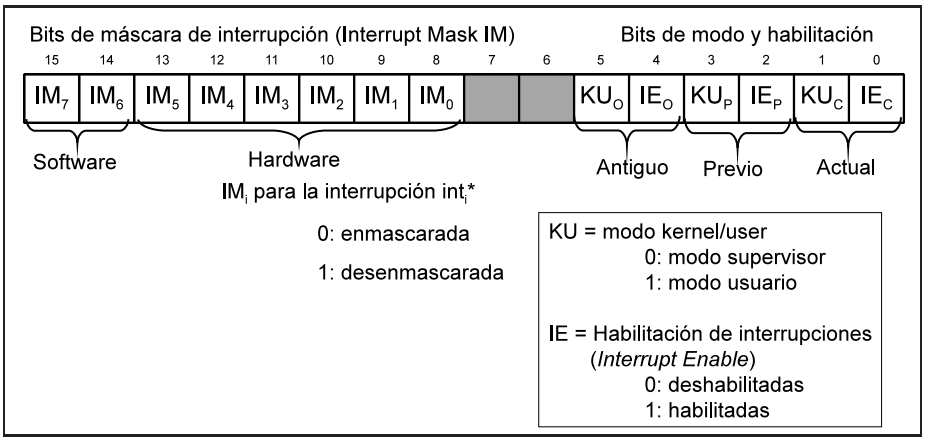

Figura A.1: Registro de estado del MIPS R2000 (número $12).

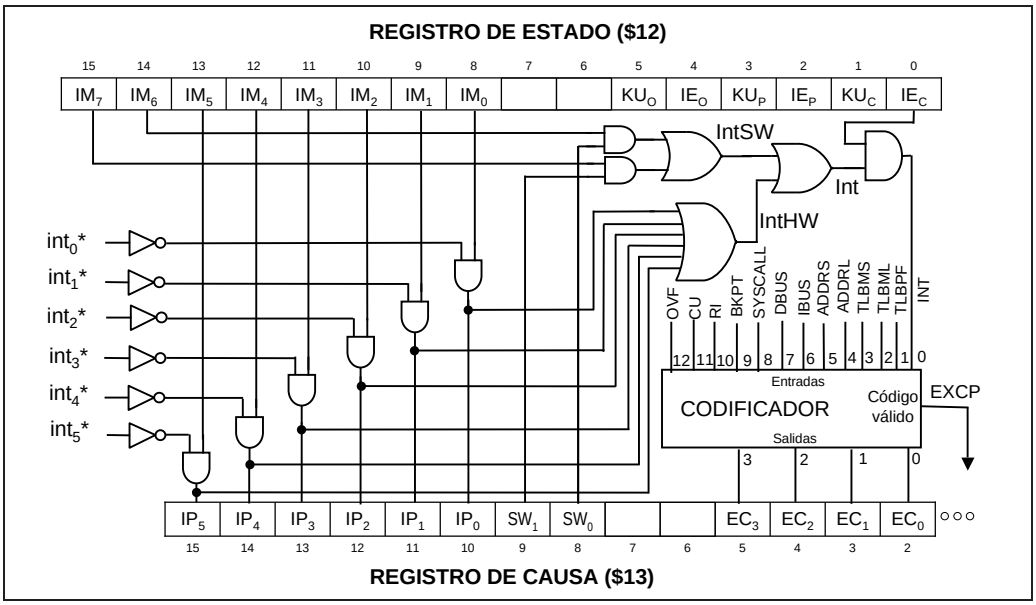

Figura A.2: Relación entre los registros de estado y de causa del MIPS R2000.