

# Estructura de Computadores

Grado de Ingeniería Informática  
ETSINF

## Tema 8: Mecanismos de sincronización de la Entrada/Salida



Curso 2017-2018



### Objetivos

- Comprender la necesidad de la sincronización de los periféricos y estudiar los distintos mecanismos de sincronización de la Entrada/Salida.
- Escribir programas que se sincronicen con los periféricos mediante consulta de estado o respondiendo a interrupciones.
- Conocer el sistema de excepciones del MIPS R2000 así como los registros de configuración implicados .
- Escribir fragmentos del manejador de excepciones del MIPS R2000.
- Comprender la relación entre los mecanismos de excepciones y las llamadas a funciones del Sistema Operativo.

# Contenido

- 1 – Mecanismos de sincronización
- 2 – Sincronización por consulta de estado
- 3 – Sincronización por interrupciones
- 4 – Soporte a las interrupciones
- 5 – Caso de estudio: MIPS R2000
- 6 – Diseño del manejador de excepciones
- 7 – La entrada/salida mediante el Sistema Operativo

# Bibliografía

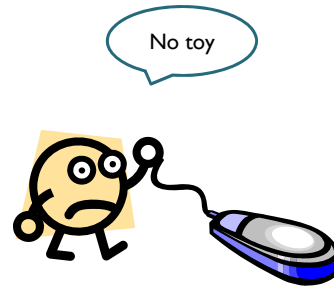
- Patterson, D.A., Hennessy, J.L.
  - ✓ Estructura y diseño de computadores. La interfaz hardware-Software (4ª ed.). Ed. Reverté, 2011
    - Cap 6
- Stallings, W.
  - ✓ Organización y arquitectura de computadores (7ª ed.). Ed. Prentice Hall, 2006
    - Cap. 7
- Hamacher, V.C., Vranesic, Z.G., Zaky, S.G.
  - ✓ Organización de computadores (5ª ed.). Ed. McGraw Hill, 2003

# I - Mecanismos de sincronización

## Necesidad de la sincronización



U.C. Pito



Mou Sito

Curso 2017-2018

5

## Sincronización

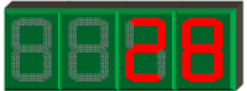
### • Conceptos

- ✓ Una operación de Entrada/Salida (E/S) siempre implica la transferencia de datos entre dos participantes:
  - El periférico (a través de su interfaz)
  - El sistema (UCP + Memoria)
- ✓ Ambos participantes funcionan a velocidades notablemente distintas.
- ✓ La transferencia deberá de ajustarse al ritmo de trabajo del participante más lento: el periférico.
  - Un teclado merece atención cada vez que se pulsa una tecla
  - Una impresora necesita tiempo para procesar un carácter (o un bloque de caracteres) antes de aceptar otro nuevo
- ✓ Habrá pues que determinar cuándo el periférico debe ser atendido.

Curso 2017-2018

6

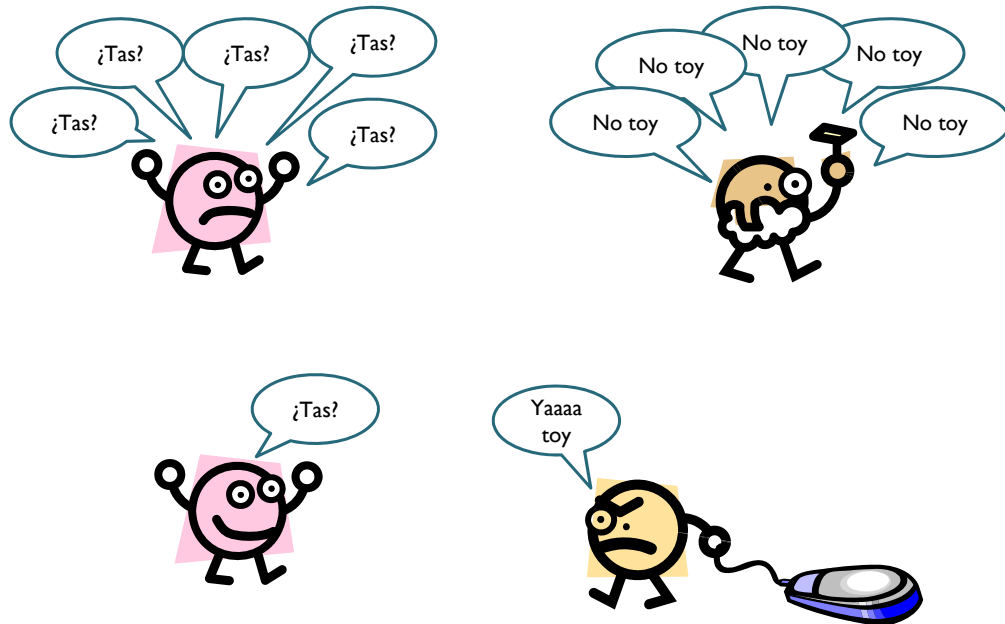
# Sincronización

- La **sincronización** es la operación que se realiza para determinar si un periférico está o no preparado para la transferencia de datos.
  - ✓ Un periférico de entrada está preparado (Ready) cuando dispone de nuevos datos para transferir al sistema.
  - ✓ Un periférico de salida está preparado cuando ya ha procesado la transferencia anterior y está listo para recibir nuevos datos.
- **Entrada/Salida Directa:** cuando el periférico está siempre preparado y no necesita sincronización.
  - ✓ Ejemplo: el visualizador  →
- **Entrada/Salida Sincronizada:** cuando si es necesaria la sincronización.

# Métodos de sincronización

- Sincronización por consulta de estado
  - ✓ La iniciativa la tiene el procesador
  - ✓ Mediante un bucle se consulta al periférico acerca de su disponibilidad
  - ✓ Cuando éste está preparado se ha producido la sincronización
- Sincronización por interrupciones
  - ✓ La iniciativa la tiene el periférico
  - ✓ El periférico avisa cuando está disponible mediante una señal de interrupción
  - ✓ Mientras tanto la UCP puede hacer otras tareas útiles

## 2 - Sincronización por consulta de estado



Curso 2017-2018

9

## Sincronización por consulta de estado

- Sincronización por consulta de estado:
  - ✓ Mediante un bucle se consulta al periférico acerca de su disponibilidad
  - ✓ Cuando éste está preparado se ha producido la sincronización
- La interfaz del periférico debe incorporar elementos para realizar la sincronización:
  - ✓ **Bit de preparado**
    - Suele aparecer en el registro de estado, con el nombre R, Ready, etc.
    - Se activa (por ejemplo, vale  $R=1$ ) cuando el dispositivo está preparado
  - ✓ **Cancelación** (reinicio, reset...)
    - Explícita: hay un bit en el registro de órdenes (CL, Clear, etc) que desactiva el bit de preparado
    - Automática: el controlador desactiva el bit de preparado cuando un programa accede a cualquier registro de la interfaz (o a alguno en concreto)

Curso 2017-2018

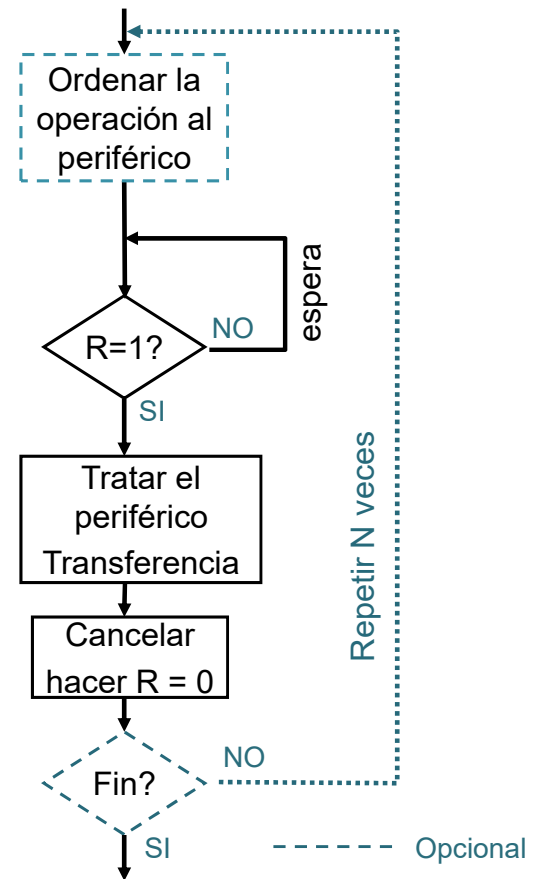
10



# Sincronización por consulta de estado

## • Esquema general

1. Primero hay que ordenar la operación al periférico.
  2. El programa entra en un bucle de espera hasta que el bit R de preparado se active (Consulta de Estado).
  3. Cuando  $R=1$  se ha producido la sincronización y hay que tratar el periférico (Leer/Escribir datos).
  4. Durante el tratamiento, hay que cancelar el bit de preparado (hacer el bit  $R = 0$ ) para que funcione la siguiente consulta de estado.
- ✓ Si procede hay que repetir la operación hasta que se transfieren todos los datos.

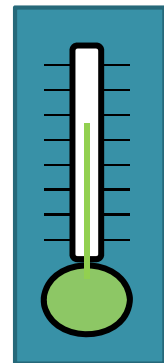


Curso 2017-2018

11

## Ejemplo I: Termómetro

- Periférico de entrada
  - Registros de la interfaz
- ✓ Dir Base DB = 0xFFFF0010



Nombre	Dir.	Acceso	Estructura
Estado	DB	Lect.	<p>R (bit 0) Ready: <math>R=1</math> cuando la temperatura ha sido adquirida</p>
Órdenes	DB+4	Escr.	<p>A (bit 0) Adquirir: <math>A=1</math> adquiere nueva temperatura  <math>A=0</math> no tiene ningún efecto            C (bit 5) Cancelar R: <math>C=1</math> para hacer <math>R = 0</math>  <math>C=0</math> no tiene ningún efecto</p>
Datos	DB+8	Lect.	<p>T (bits 7...0): temperatura</p>

Curso 2017-2018

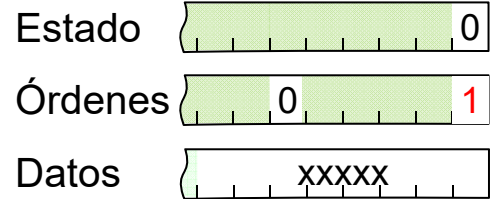
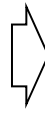
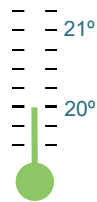
12

# Ejemplo I: Termómetro

## • Funcionamiento del sensor

Orden: Adquirir:  
la temperatura

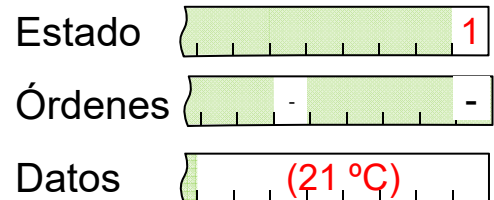
①



Sincronización por prueba de  
estado

Periférico preparado:  
la temperatura  
es de 21 °C

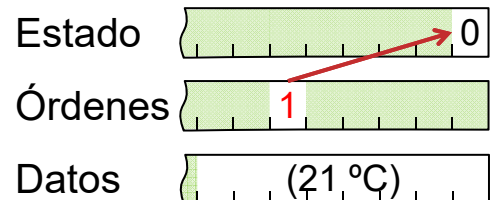
②



Lectura  
temperatura

③

```
la $t0, 0xFFFF0010
lb $t1, 8($t0)
sb $t1, Temp
```



Orden de  
cancelación

④

```
li $t1, 0x20
sb $t1, 4($t0)
```

Curso 2017-2018

13

# Ejemplo I: Termómetro

## • Programación

```
.data 0x1000000
Temp: .word 0
```

```
.text 0x00400000
inicio: .....
```

```
bucle: .....
la $a0, Temp
jal LeeTemp
```

```
.end
```

```
LeeTemp: la $t0, 0xFFFF0010
li $t1, 0x01
sb $t1, 4($t0)
```

```
CdE: lb $t1, 0($t0)
andi $t1, $t1, 1
beqz $t1, CdE
```

```
lb $t1, 8($t0)
sb $t1, 0($a0)
```

```
li $t1, 0x20
sb $t1, 4($t0)
jr $ra
```



Preparación de  
la dirección

Orden adquirir

①

Sincronización  
consulta de  
estado

②

Transferencia

③

Cancelación

④

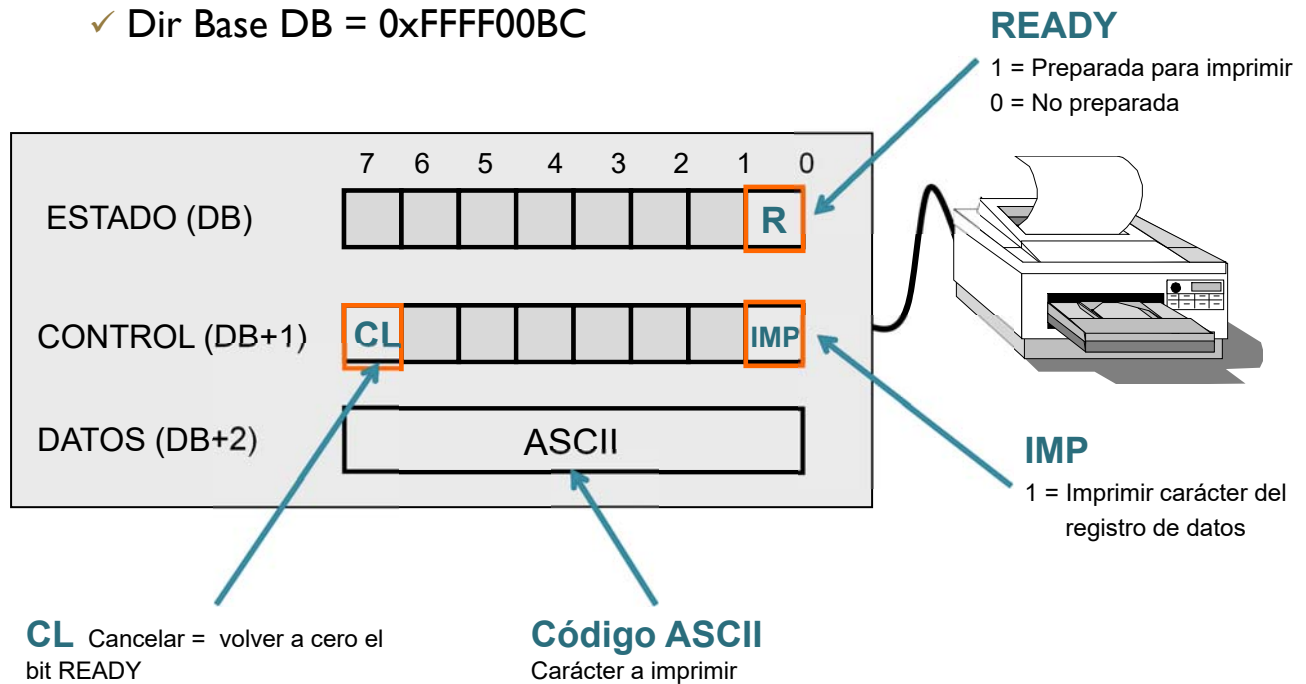
↑ Observe cómo se usa el desplazamiento  
para acceder a los distintos registros

Curso 2017-2018

14

## Ejemplo 2: Impresora

- Periférico de salida
- Registros de la interfaz
  - ✓ Dir Base DB = 0xFFFF00BC



Curso 2017-2018

15

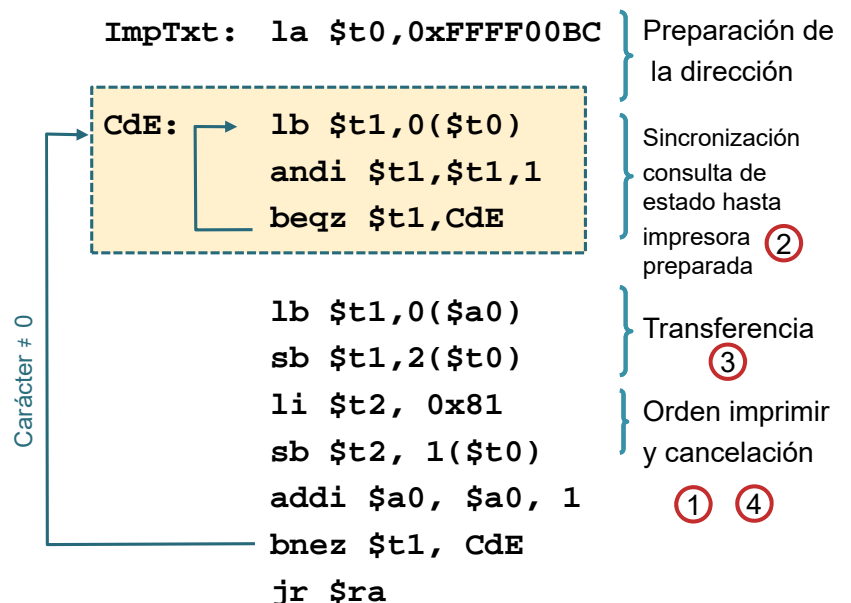
## Ejemplo 2: Impresora

- Programación

```
.data 0x1000000
LineaTxt: .asciiz 'texto'
```

```
.text 0x00400000
inicio: .....
        .....
        .....
        la $a0, LineaTxt
        jal ImpTxt
        .....
        .....

.end
```



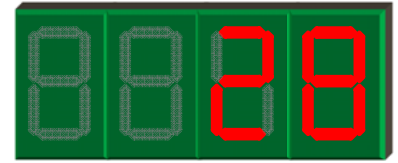
Curso 2017-2018

16



## Ejercicio: Termómetro + Visualizador

- Termómetro visto en el ejemplo 1
- + Visualizador de 4 dígitos
  - Visualiza un valor entero con signo de 8 bits
  - Dir. Base DB = 0xFFFF0020
  - Registros de la interfaz



Nombre	Dir.	Acceso	Estructura
Órdenes	DB+0	Escr.	

Activa el Visualizador y el parpadeo:

- ON (bit 0): a 0 apagado, a 1 visualiza VALOR
- Frec (bits 6..4): frecuencia de parpadeo en Hz
- Frec = 0: continuo

Datos	DB+4	Escr.	
-------	------	-------	--

VALOR a visualizar:

Número de 8 bits en Ca2 (-127 .. +127)

El valor de -128 visualiza '- E r r'

Se visualiza al poner el bit ON a 1

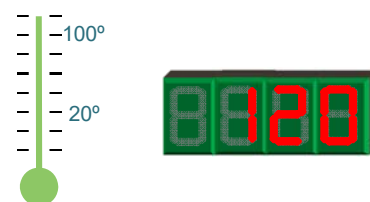
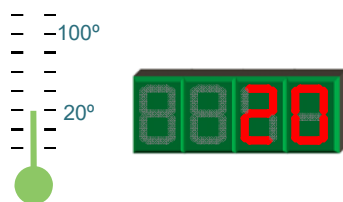
Curso 2017-2018

17

## Ejercicio: Termómetro + Visualizador



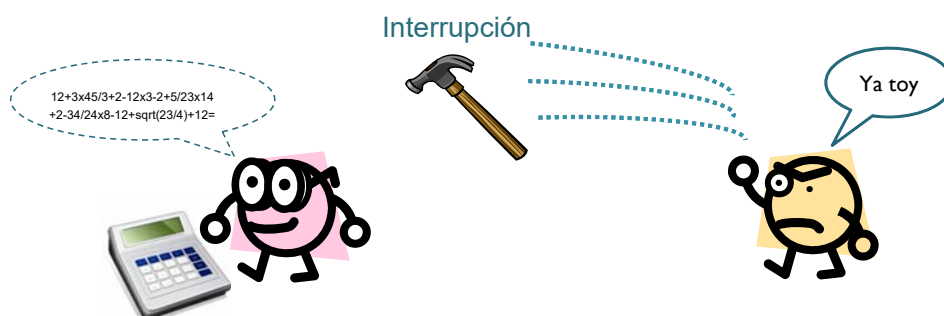
- Realice una subrutina 'LeeTemp' que adquiere una temperatura del termómetro y la muestra en el visualizador:
  - ✓ Si la temperatura > 100°C -> intermitente (frec 2 Hz)
  - ✓ Si la temperatura <= 100°C -> continuo (frec 0 Hz)



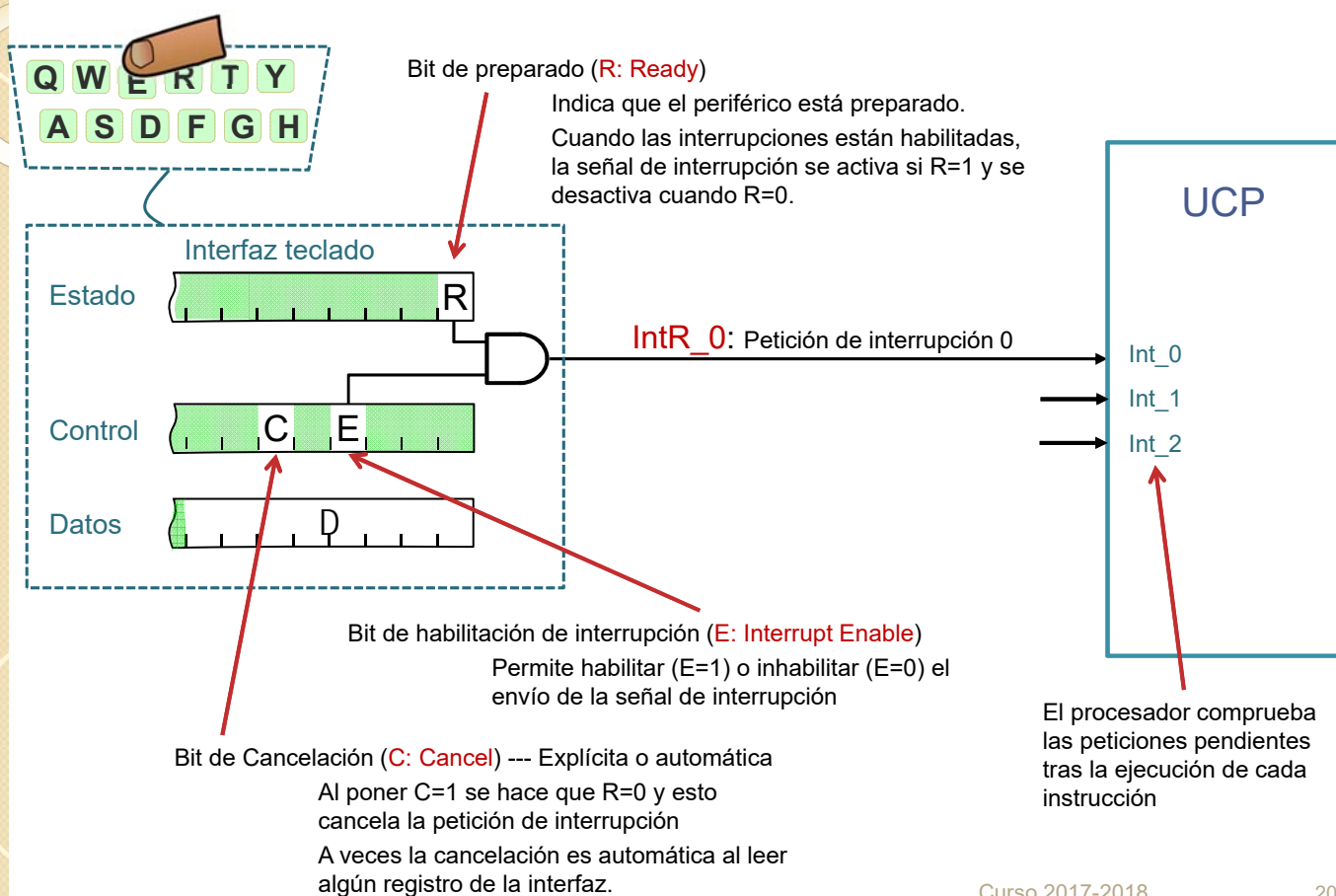
Curso 2017-2018

18

### 3 - Sincronización por interrupción



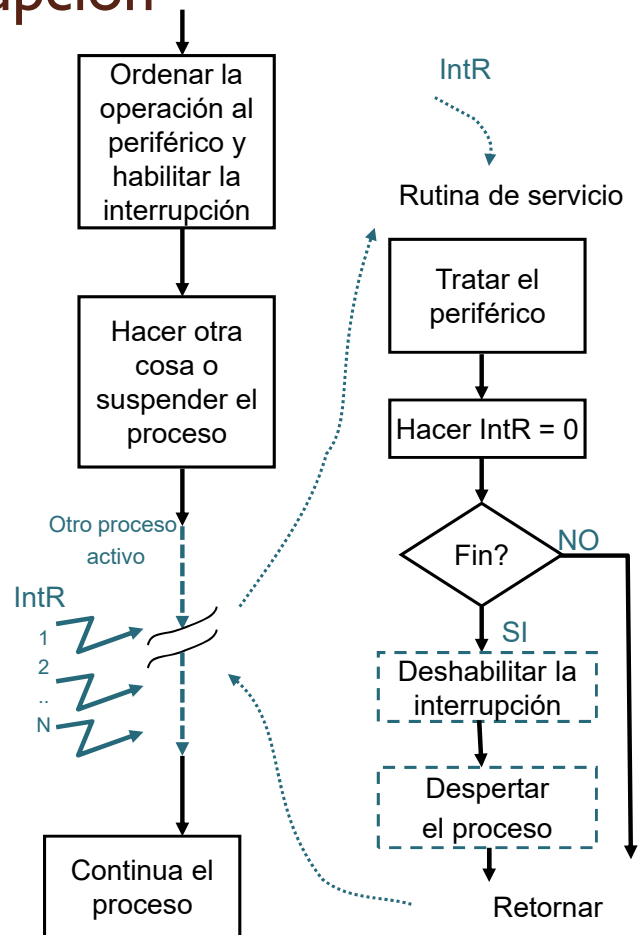
### Recursos necesarios



# Sincronización por interrupción

## • Esquema general

- ✓ Primero hay que ordenar la operación al periférico y habilitar la interrupción.
- ✓ El programa se puede dedicar a otra cosa o se puede suspender (multitarea)
- ✓ Cuando se activa la interrupción (IntR) se ejecuta la rutina de servicio.
- ✓ Como ya está sincronizado, solo hay que tratar al periférico (leer/Escribir datos). Esta rutina DEBE cancelar la interrupción (IntR=0).
- ✓ La interrupción se puede repetir N veces.
- ✓ Al finalizar hay que deshabilitar la interrupción y despertar al proceso.



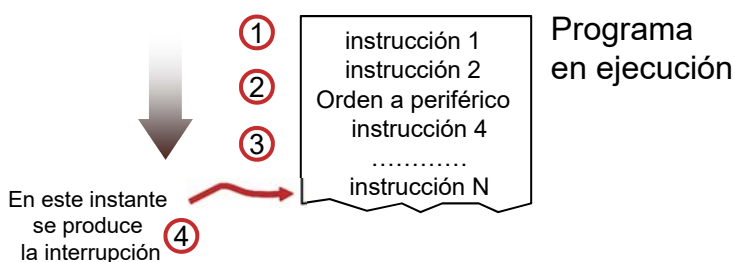
Curso 2017-2018

21

# Sincronización por interrupción

## • Secuencia de eventos:

1. Programa en ejecución
2. Se programa la operación de E/S en el periférico
3. El programa sigue o se conmuta otra tarea (multitarea)
4. El periférico está preparado (R=I) → Activa la interrupción
  - El procesador comprueba las interrupciones al terminar la instrucción en curso.



Curso 2017-2018

22

# Sincronización por interrupción

- El programa queda interrumpido

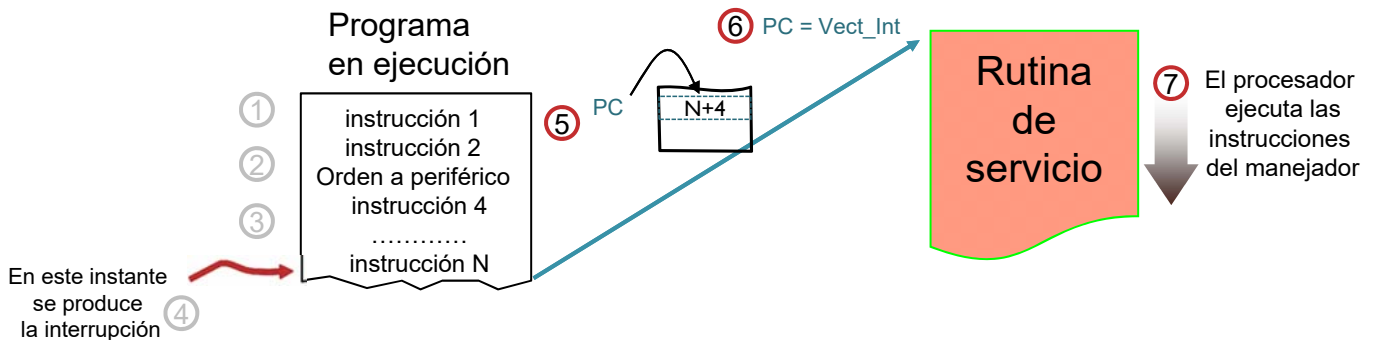
5. Se guarda la dirección de la siguiente instrucción del programa

- Se salta a la rutina de servicio

6. Se pone en el contador de programa el vector de interrupción

7. Se ejecuta la rutina de servicio

Automático



Curso 2017-2018

23

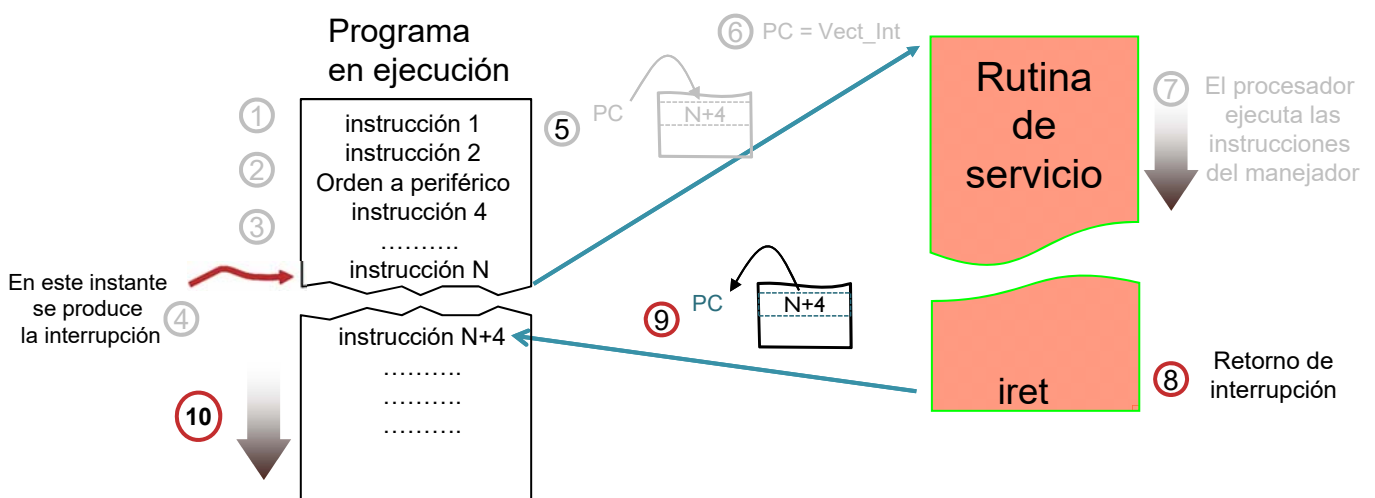
# Sincronización por interrupción

- La rutina de servicio termina:

8. Se ejecuta una instrucción de retorno de interrupción (iret)

9. Esto restaura el contador de programa del programa interrumpido

10. El programa interrumpido continua su ejecución



Curso 2017-2018

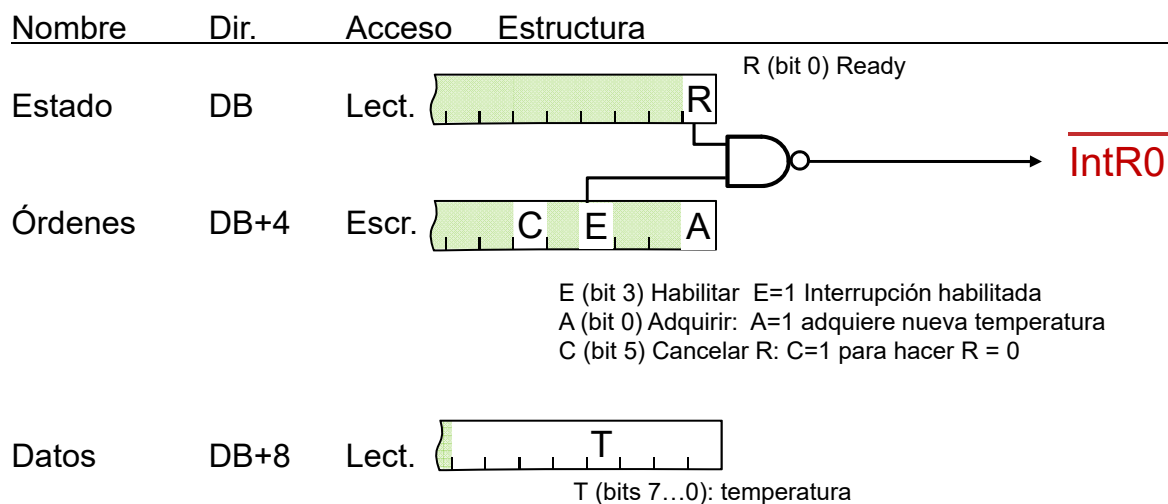
24

## ¿Qué hace una rutina de servicio?

- Como la sincronización ya está realizada, la rutina de servicio debe atender a la transferencia de datos a/desde el periférico.
  - ✓ **NO** hay que volver a sincronizar por prueba de estado
  - ✓ Los datos se transfieren leyendo o escribiendo los registros de datos de la interfaz.
- Muy importante:
  - ✓ Se le **DEBE** decir al periférico que quite su petición de interrupción pues ya ha sido atendido (**CANCELACIÓN**)
  - ✓ La rutina **DEBE** garantizar que los registros que utiliza mantengan su contenido original antes de retornar (Salvar/restaurar registros)
    - Su contenido original era el que tenían antes de producirse la interrupción

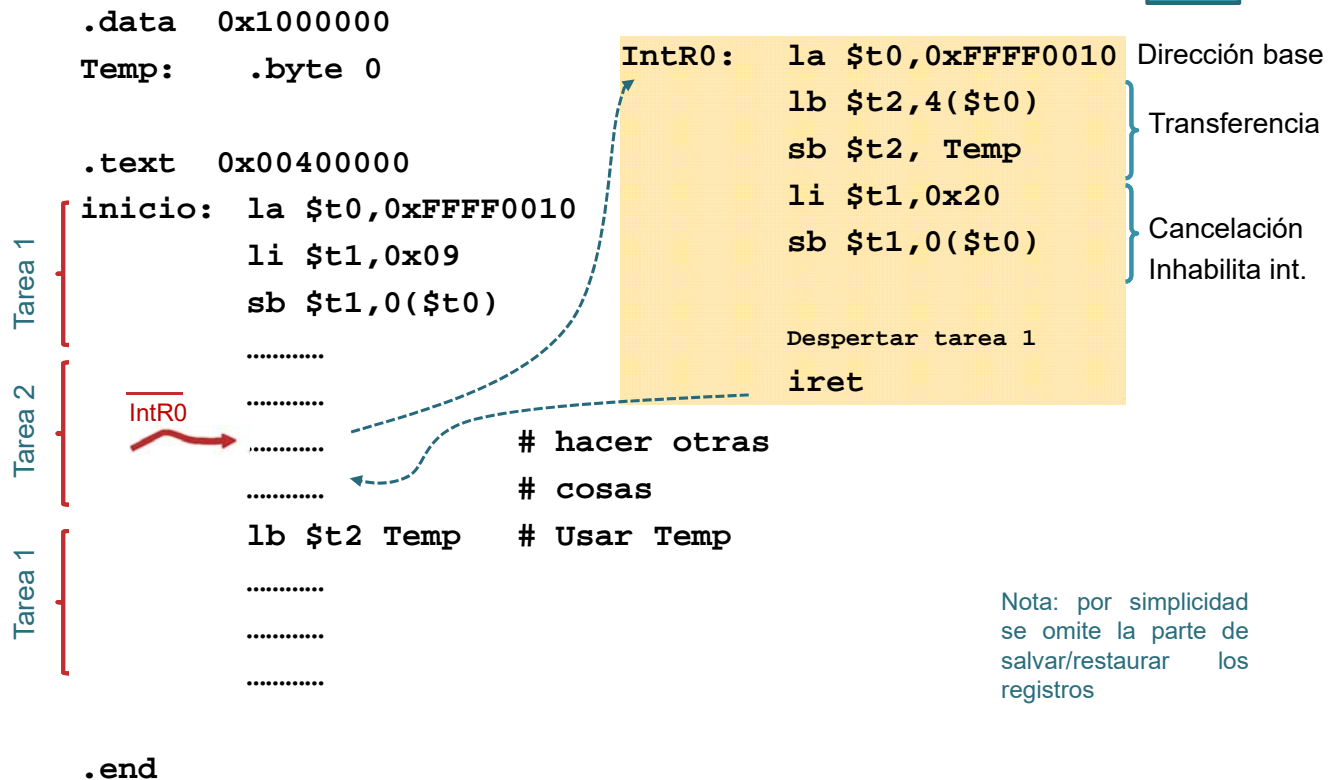
## Ejemplo 3: Termómetro (con interrupción)

- Ampliación de la interfaz del termómetro
  - ✓ Conectado a la línea **IntR0**
  - ✓ Bit E=1 para habilitar la interrupción





## Ejemplo: 3 Termómetro (con interrupción)



## Conclusión: esquemas de sincronización

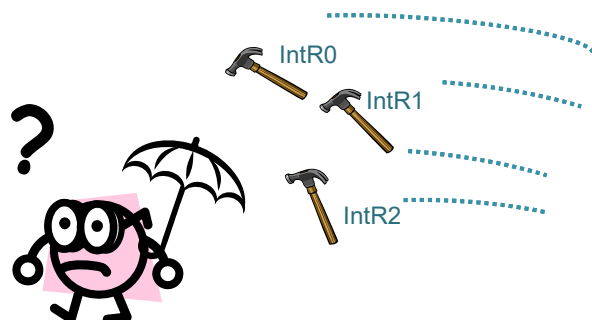
- Por consulta de estado
  - ✓ Configuración del periférico
    - Interrupciones inhibidas
  - ✓ Bucle:
    - leer estado
    - iterar mientras no preparado
  - ✓ Tratamiento
  - ✓ Cancelación explícita o automática
- Por interrupción
  - ✓ Configuración del periférico
    - Interrupciones habilitadas
  - ✓ No hay bucle
  - ✓ Hay que codificar sólo el tratamiento
  - ✓ Cancelación explícita o automática

# Dudas

- ¿Qué ocurriría si una rutina de servicio NO cancela la interrupción?
- **AFIRMACIÓN:**
  - ✓ Cuando se salta a una rutina de servicio, las interrupciones en el procesador DEBEN quedar automáticamente inhabilitadas. Al retornar de dicha rutina de servicio (iret) las interrupciones se vuelven a habilitar automáticamente.
  - ✓ ¿Porqué?

## 4 – Soporte a las interrupciones

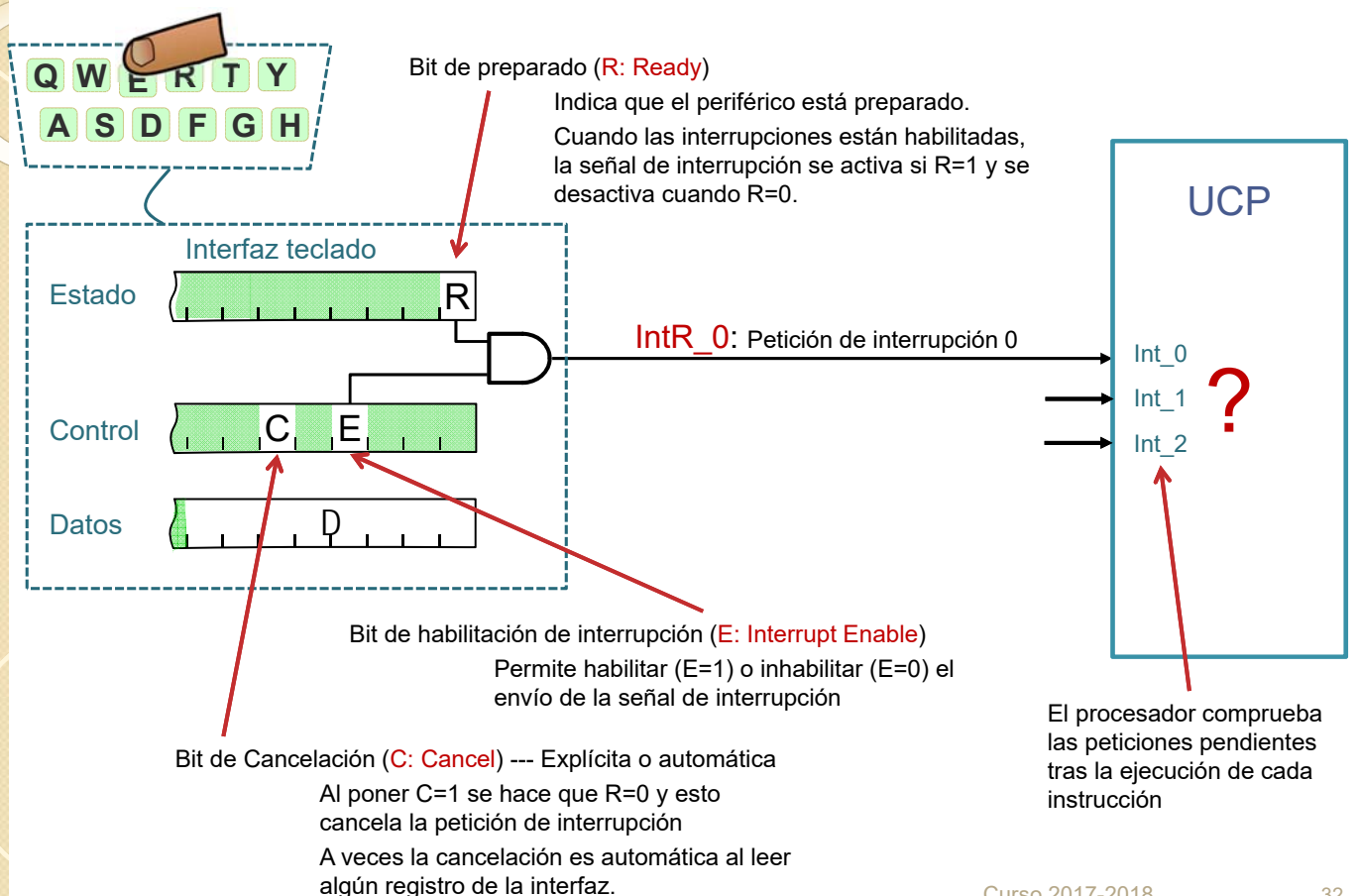
- ¿Se puede impedir que un procesador sea interrumpido?
- ¿Cómo se inhabilitan las interrupciones?
- ¿Pueden haber varias interrupciones?
- ¿Dónde están las rutinas de servicio de la interrupciones?
- ¿Puedo habilitar unas interrupciones e inhabilitar otras?



## Soporte a las interrupciones

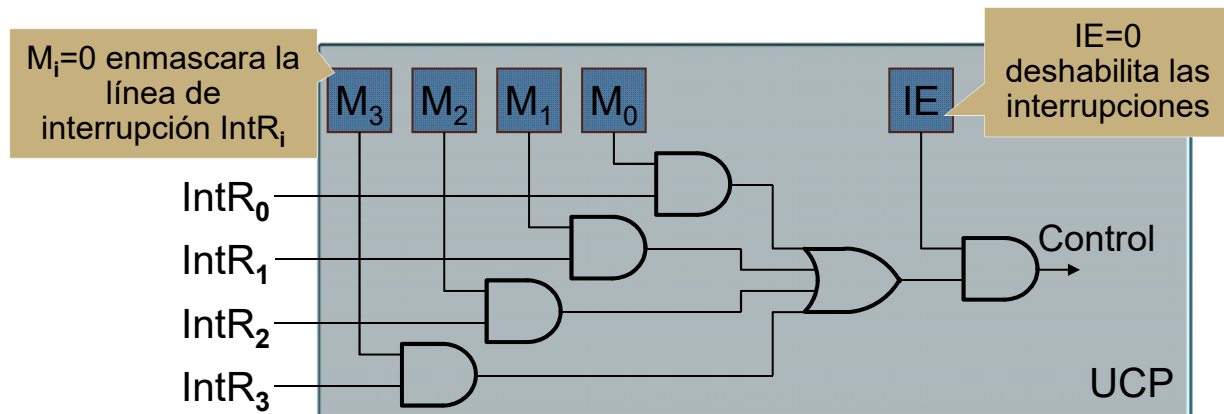
- Las interrupciones en el procesador
  - ✓ Los procesadores dan soporte a las interrupciones dentro de un esquema más general de evento: las excepciones
  - ✓ El circuito de control y el juego de instrucciones permiten la detención de los programas y su reanudación posterior
  - ✓ La gestión de las interrupciones requiere la presencia de registros de control adicionales dentro del procesador
- Las interrupciones en el adaptador
  - ✓ Para que un periférico admita sincronización por interrupciones, ha de disponer de la lógica apropiada
  - ✓ La interfaz del adaptador incluye bits y registros de configuración del mecanismo de interrupciones

## Recursos necesarios



# Las interrupciones en el procesador

- ¿ Pueden haber varias interrupciones?
  - ✓ El procesador dispone de una o varias entradas de interrupción
  - ✓ El procesador puede ignorar todas o algunas de las entradas  $IntR_i$
  - ✓ Para ello dispone de diversos bits de control:
    - Los bits de máscara ( $M_i$ ) permiten enmascarar (= ignorar) cada entrada
    - El bit de habilitación general (IE) permite ignorarlas todas
  - ✓ Cada interrupción tiene su rutina de servicio

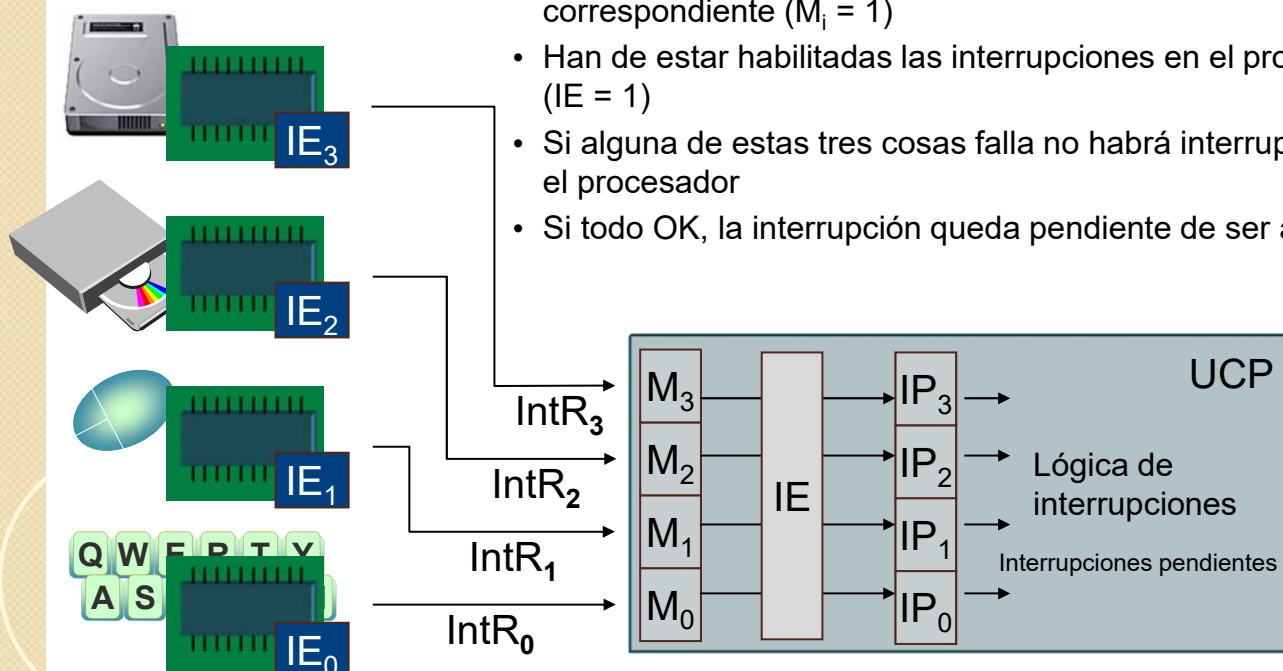


Curso 2017-2018

33

## El camino de las interrupciones

- Para que un periférico provoque una interrupción:
  - Ha de tener habilitada la interrupción en su interfaz ( $E_i = 1$ )
  - Ha de estar desenmascarada la línea de interrupción correspondiente ( $M_i = 1$ )
  - Han de estar habilitadas las interrupciones en el procesador ( $IE = 1$ )
  - Si alguna de estas tres cosas falla no habrá interrupción en el procesador
  - Si todo OK, la interrupción queda pendiente de ser atendida

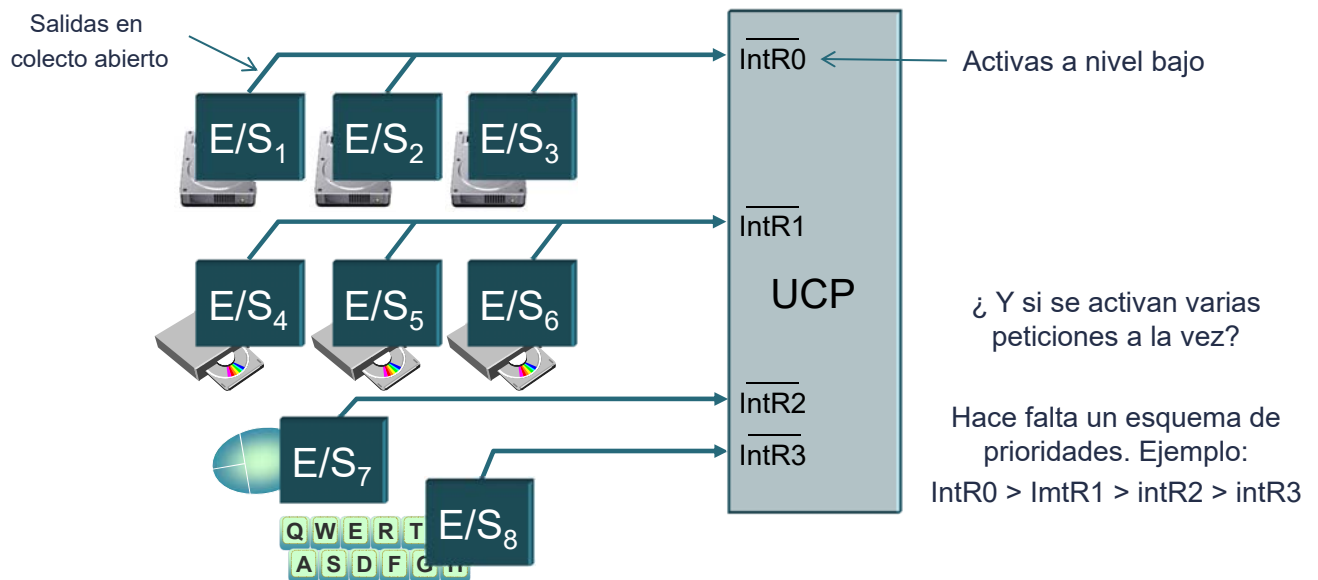


Curso 2017-2018

34

# Multiplicidad de interrupciones

- Si yo tengo 8 periféricos, ¿cómo los conecto al procesador que sólo tiene 4 líneas IntR?
- Esquemas de conexión múltiple

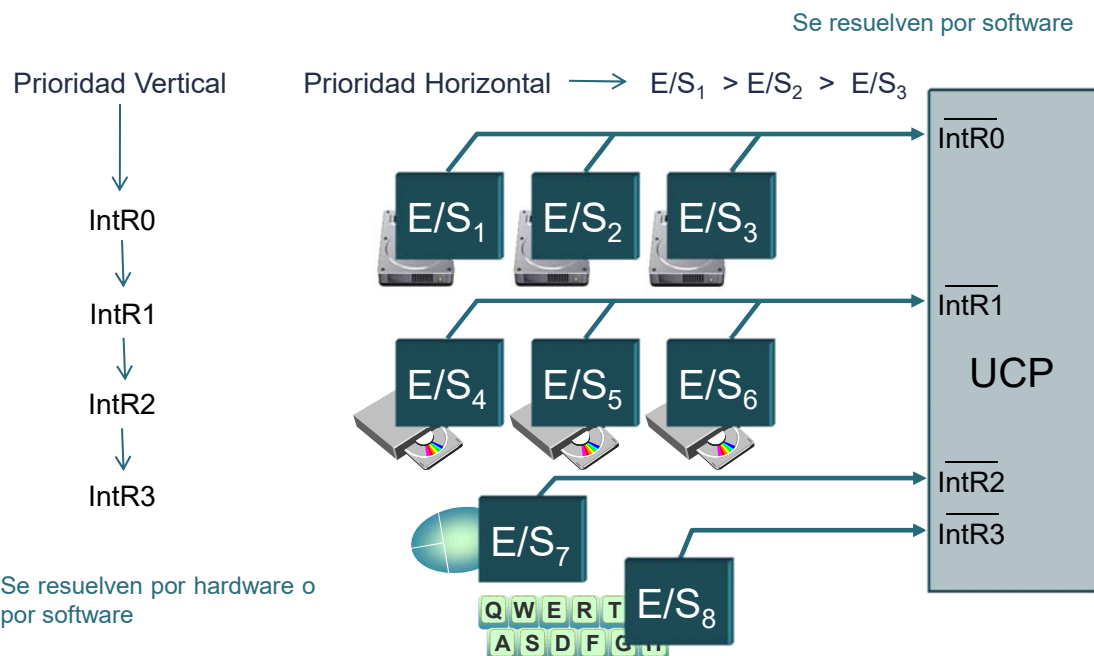


Curso 2017-2018

35

# Prioridades de las interrupciones

- Hay varios esquemas de prioridad:
  - Prioridades verticales
  - Prioridades horizontales



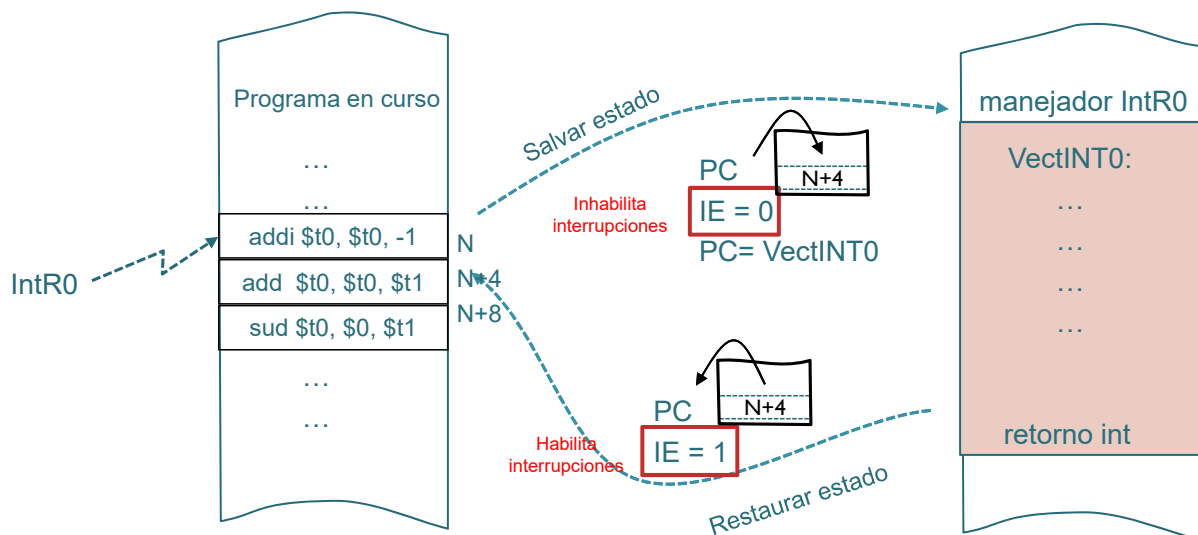
Curso 2017-2018

36



# Rutinas de servicio

- ¿Cómo se llega a la rutina de servicio?
  - ✓ La dirección inicial se una rutina de servicio se denomina: VECTOR DE INTERRUPCIÓN
  - ✓ La UCP comprueba en cada ciclo las interrupciones pendientes
  - ✓ Si hay alguna se inicia el servicio a la interrupción

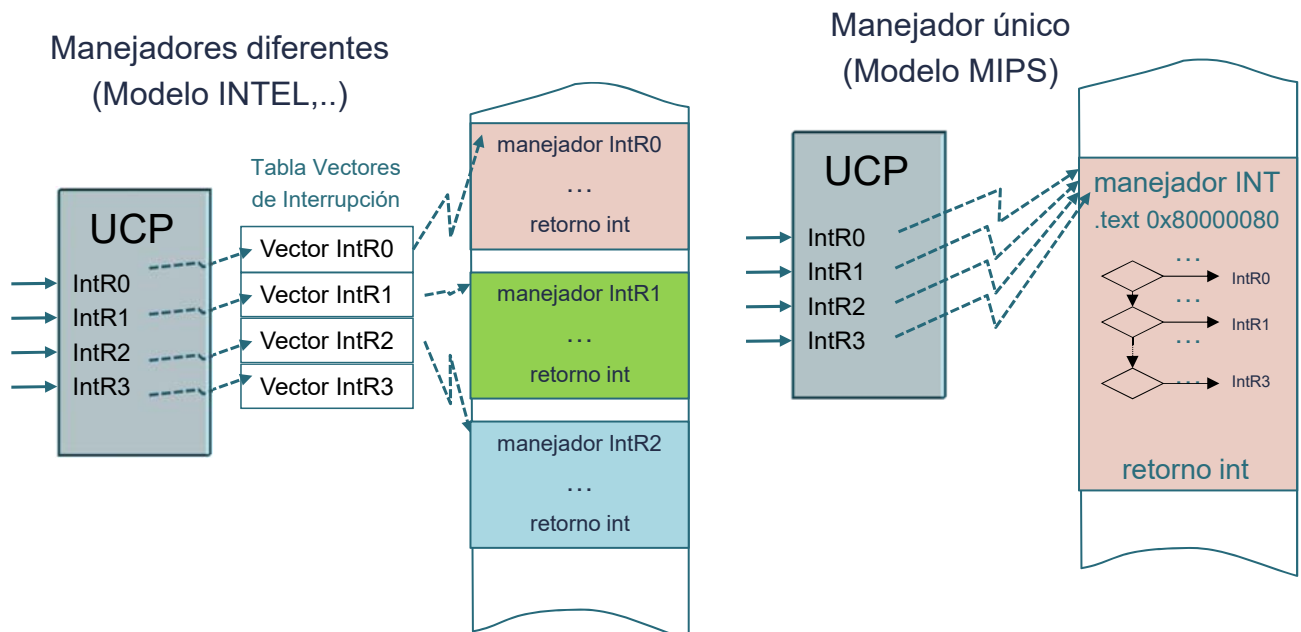


Curso 2017-2018

37

# Rutinas de servicio

- ¿Dónde están la rutinas de servicio?
  - ✓ Cada petición de interrupción tiene una rutina de servicio diferente
  - ✓ Existen diversos esquemas:



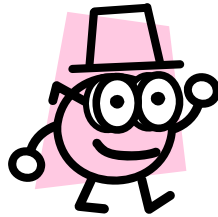
Curso 2017-2018

38



## 5 – Caso de estudio

### Las interrupciones en el MIPS R2000



M.I.P. Sito

Curso 2017-2018

39

## Las interrupciones y las excepciones

### • Definición

- ✓ Una excepción es un mecanismo que, ante una situación especial, cambia el flujo normal de ejecución

### ✓ Fuentes de excepción

- Las líneas de interrupción de periférico
- Dentro de la UCP: la UAL, la lógica de acceso al bus, la TLB
- Ciertas instrucciones de los programas

Las interrupciones son un caso particular de excepción

### ✓ Aplicaciones diversas

- Sincronización con los periféricos
- Tratamiento de errores de los programas en ejecución
- Soporte a la memoria virtual
- Ejecución controlada de programas
- Implementación eficiente de funciones del sistema operativo

# Tipos de excepciones en el MIPS R2000

## EXCEPCIONES

6 interrupciones hardware externas activas por nivel ( $int_5^* .. int_0^*$ )

2 interrupciones software ( $SW_1, SW_0$ )

3 excepciones para manejo de tabla de páginas

2 excepciones para manejo de formación de direcciones

2 excepciones para fallos de bus

1 excepción de desbordamiento aritmético (*overflow*)

1 excepción de instrucción reservada (ilegal)

1 excepción de coprocesador no presente

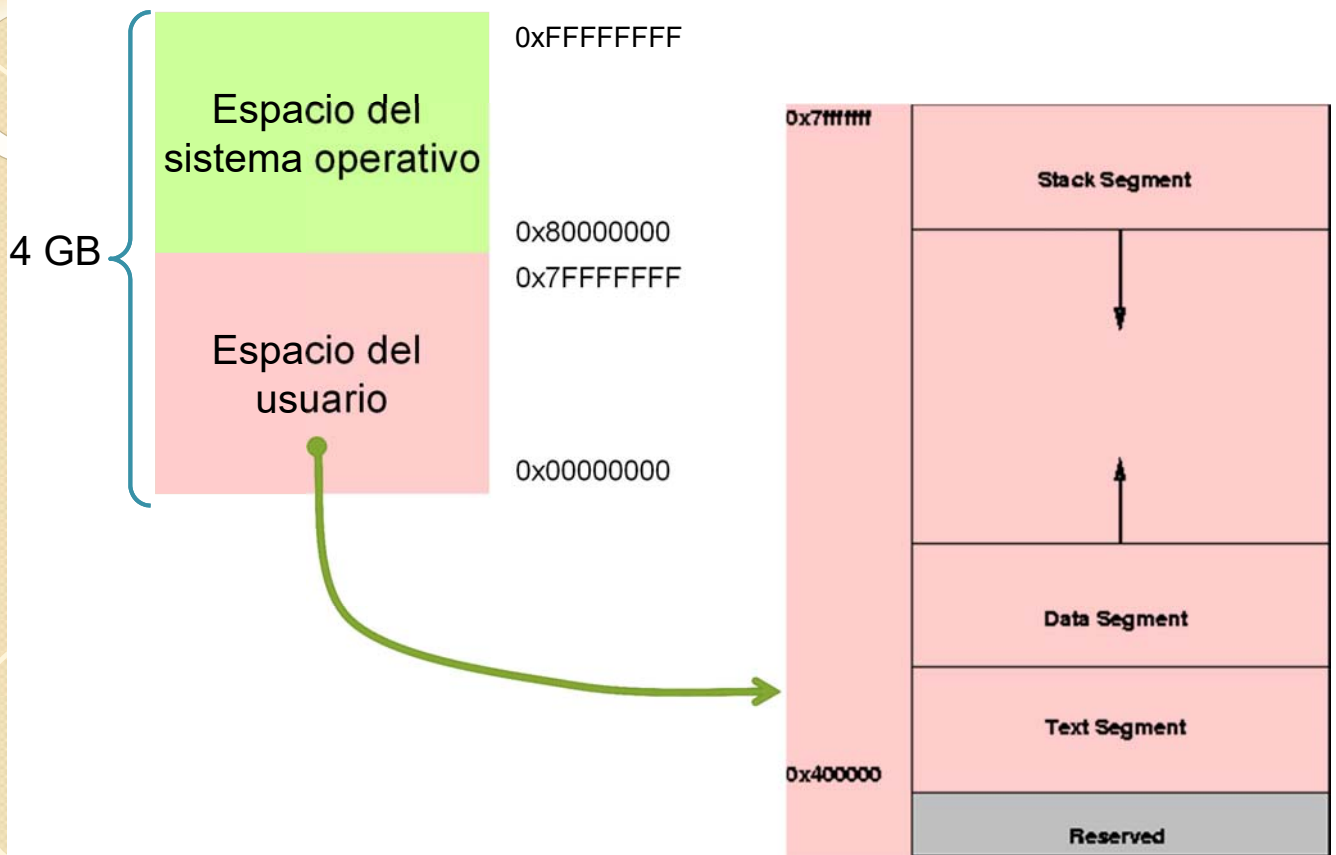
1 excepción de llamada al sistema (instrucción **syscall**)

1 excepción de punto de ruptura (instrucción **break**)

# Modos de operación en el MIPS R2000

- El procesador tiene dos modos de operación
  - ✓ Modo Usuario (user mode)
  - ✓ Modo Supervisor (kernel mode)
- Se precisa estar en modo supervisor para:
  - ✓ Acceder a un conjunto adicional de registros
  - ✓ Ejecutar un conjunto adicional de instrucciones
  - ✓ Acceder a todo el espacio de memoria
    - Modo Usuario: de 0x00000000 a 0x7fffffff
    - Modo Supervisor: de 0x00000000 a 0xffffffff
  - ✓ Uso más frecuente
    - Los programas corrientes funcionan en modo usuario
    - El núcleo del sistema operativo y los programas especialmente habilitados funcionan en modo supervisor

# Espacio de memoria del MIPS R2000

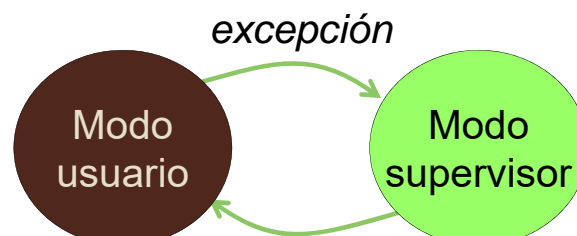


Curso 2017-2018

43

## El cambio de modo

- **De modo Usuario a Supervisor**
  - ✓ Es la reacción del procesador cuando se produce la excepción
    - Las interrupciones quedan inhibidas.
    - Cambia a modo supervisor.
    - Comienza a leer instrucciones en el vector (Manejador de excepciones).
- **De modo Supervisor a Usuario**
  - ✓ Se ejecuta la instrucción rfe (return from exception)
    - Esta instrucción privilegiada sólo se puede ejecutar en modo supervisor.

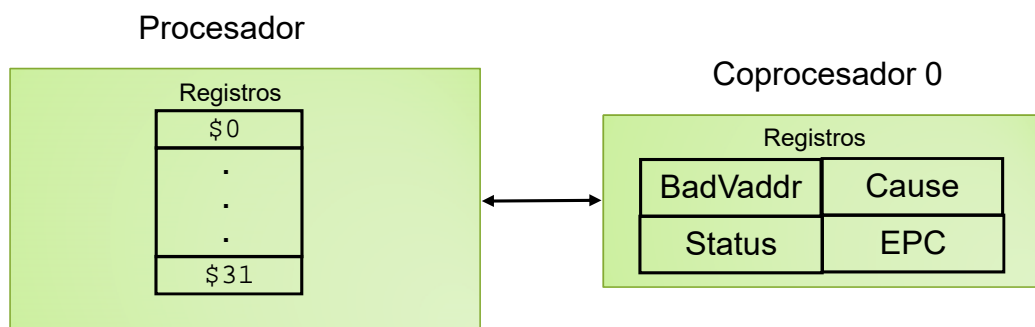


Curso 2017-2018

44

# Coprocesador de excepciones

- Para soportar los dos modos, la arquitectura MIPS incluye el coprocesador CP0
  - ✓ Sólo es accesible en modo supervisor.
  - ✓ Contiene los registros necesarios para el manejo y control de las excepciones: tipo de excepción, dirección donde se produce y otros detalles relevantes.



## Registros de manejo de excepciones (en CP0)

Número	Nombre	Descripción
\$8	Bad virtual address	(si aplicable) Dirección virtual que ha generado el fallo de página
\$12	Status	Máscara y habilitación de interrupciones
\$13	Cause	Tipo de excepciones e interrupciones pendientes
\$14	EPC	Dirección de la instrucción (PC) donde se produce la excepción.

### Instrucciones de acceso a los registros del coprocesador 0

Move from CP0: `mfc0 Rgeneral, RCP0`       $R_{\text{general}} \leftarrow R_{\text{CP0}}$

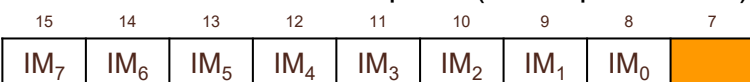
Move to CP0: `mtc0 Rgeneral, RCP0`       $R_{\text{CP0}} \leftarrow R_{\text{general}}$

Ejemplo: lectura del registro de causa en \$t0: `mfc0 $t0, $13`



# Registro de estado (\$I2: Status Register, SR)

Bits de máscara de interrupción (Interrupt Mask IM)



Software

IM<sub>7</sub> para SW<sub>1</sub>

IM<sub>6</sub> para SW<sub>0</sub>

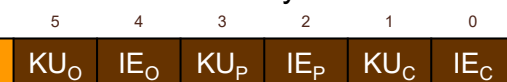
Hardware

IM<sub>i</sub> para la interrupción int<sub>i</sub>\*

0: enmascarada

1: desenmascarada

Bits de modo y habilitación



Antiguo

Previo

Actual

KU = modo kernel/user

0: modo supervisor

1: modo usuario

IE = Habilidad de interrupciones  
(*Interrupt Enable*)

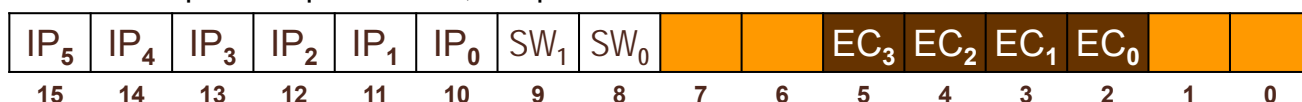
0: deshabilitadas

1: habilitadas

# Registro de causa (\$I3: Cause Register CR)

Interrupciones pendientes, 1: pendiente

Código de Excepción



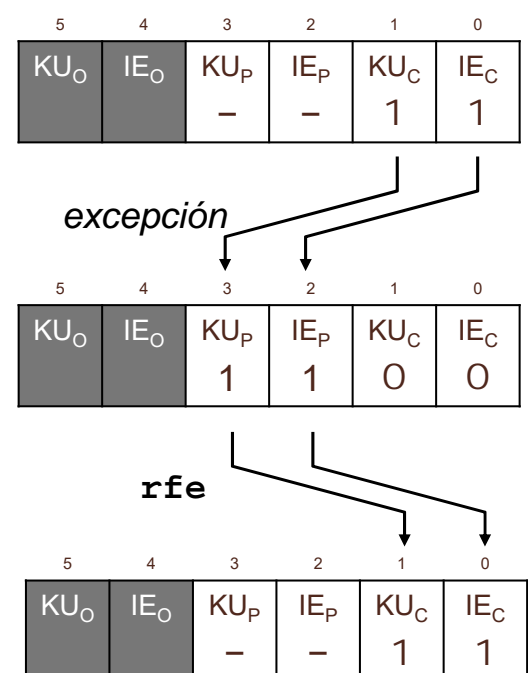
EC <sub>3..0</sub>	Nombre	Motivo que provoca la excepción
0	INT	Interrupción (hardware o software)
1	TLBPF	(TLB) Intento de escritura en página protegida
2	TLBML	(TLB) Intento de lectura de instrucción en página inválida
3	TLBMS	(TLB) Intento de lectura de datos en página inválida
4	ADDRL	Error en dirección durante una operación de lectura (I o D)
5	ADDRS	Error en dirección durante una operación de escritura
6	IBUS	Error en el bus externo (I)
7	DBUS	Error en el bus externo (D)
8	SYSCALL	Ejecución de instrucción de llamada al sistema operativo
9	BKPT	Ejecución de instrucción de punto de ruptura
10	RI	Ejecución de instrucción reservada (ilegal)
11	CU	Coprocesador no utilizable
12	OVF	Desbordamiento aritmético

## Dirección de excepción (\$I5: EPC)

- EPC: Exception Program Counter: Registro que contiene el valor del PC cuando se produzca la excepción
  - ✓ Contiene la dirección de la instrucción afectada. Esta instrucción no ha podido acabar su ejecución.
- El EPC apunta, según el tipo de excepción, a:
  - ✓ Fallos de página: la instrucción (de cualquier tipo) que no se ha podido leer o la instrucción load o store que ha generado el acceso.
  - ✓ Errores aritméticos, de bus, etc: la instrucción causante.
  - ✓ Interrupciones: la instrucción que se hubiera ejecutado si no fuera porque ha llegado la interrupción.

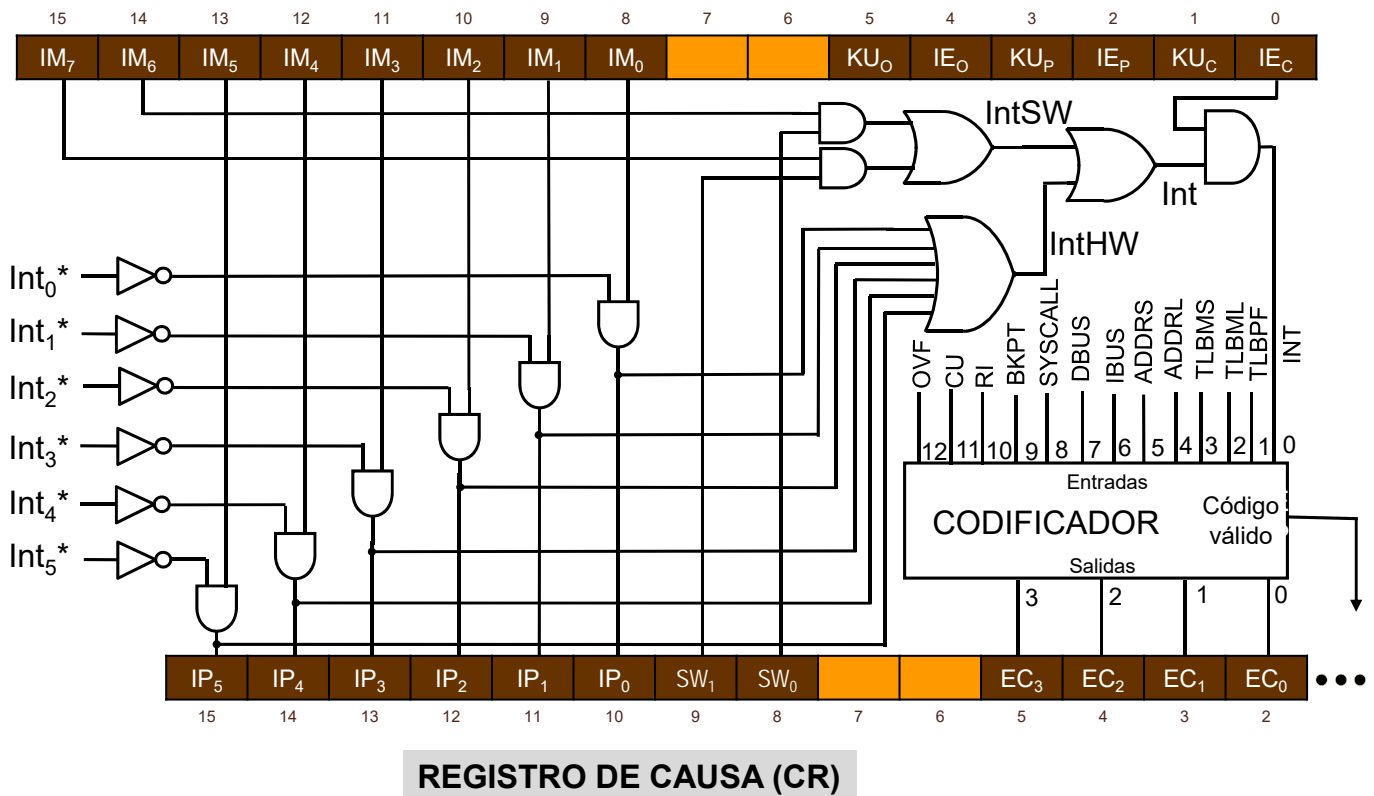
## El cambio de modo en CP0

- Programa de usuario en ejecución
  - Interrupciones habilitadas
  - Modo usuario
- Se produce la excepción
  - Bits de modo y habilitación = 0
  - EPC = valor del PC
  - CR y BadVaddr se actualizan
  - PC = dirección de inicio del manejador
- Ejecución del manejador
  - Interrupciones inhibidas
  - Modo supervisor
- Retorno (rfe):
  - Vuelta al estado inicial



# Hardware asociado a las excepciones

## REGISTRO DE ESTADO (SR)



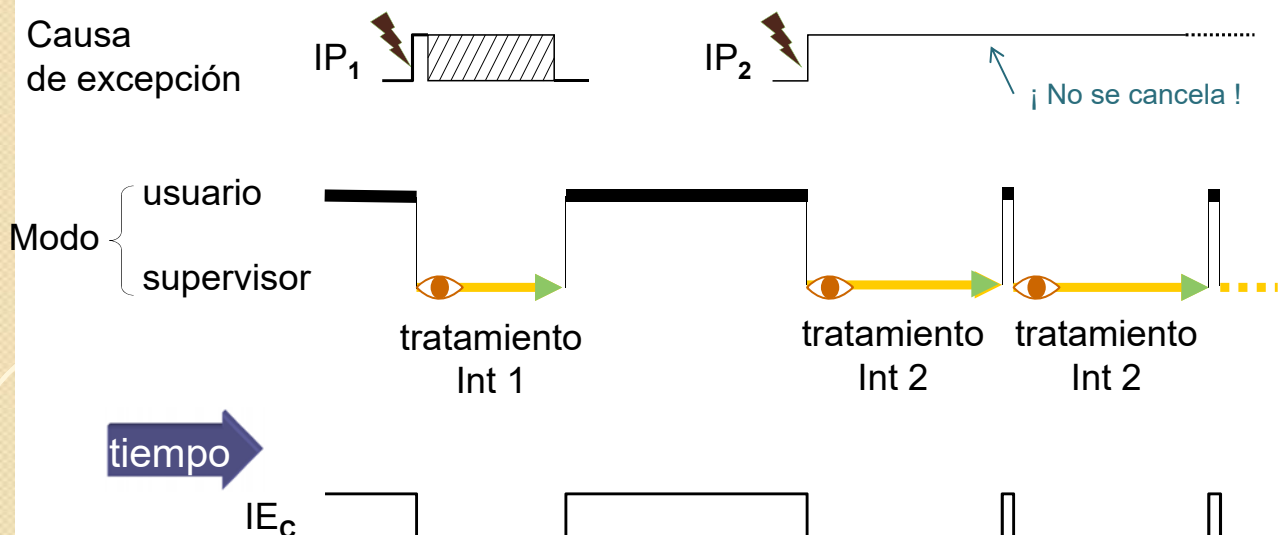
Curso 2017-2018

51

## Manejo de interrupciones

### • Cronograma ejemplo 1:

- ✓ El tratamiento que se dé al periférico ha de garantizar que éste cancele la petición de interrupción.
- ✓ Ejemplo: tratamiento Int 1 correcto, Int 2 incorrecto



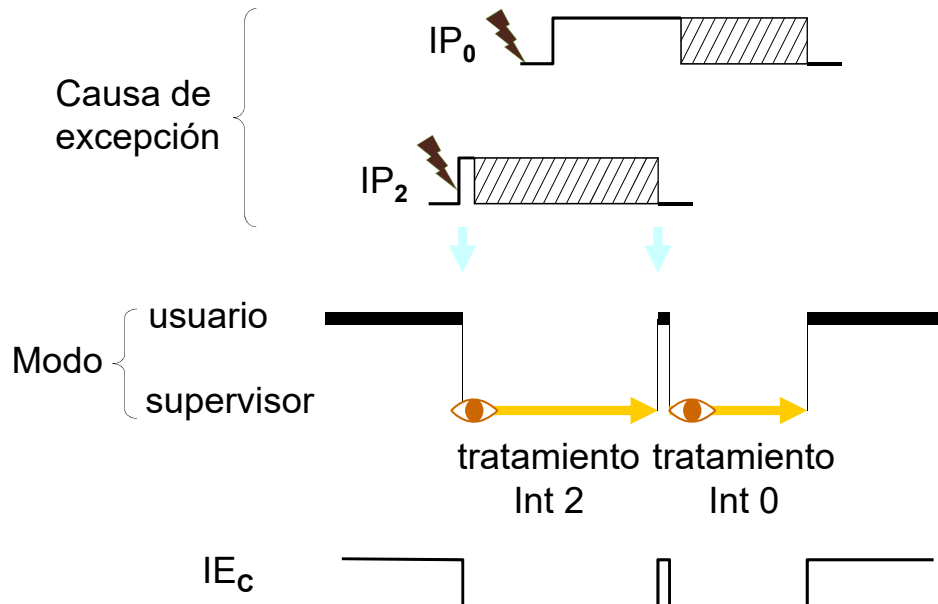
Curso 2017-2018

52

# Interrupciones durante la ejecución del manejador

- Cronograma ejemplo 2:

✓ Mientras se atiende la interrupción 2, llega la interrupción 0

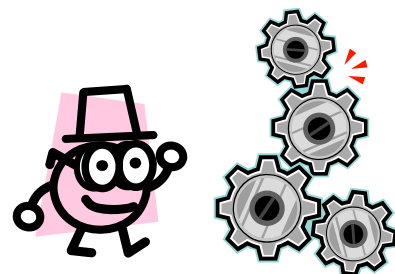


Curso 2017-2018

53

## 6 - Diseño del manejador de excepciones

Caso del MIPS R2000



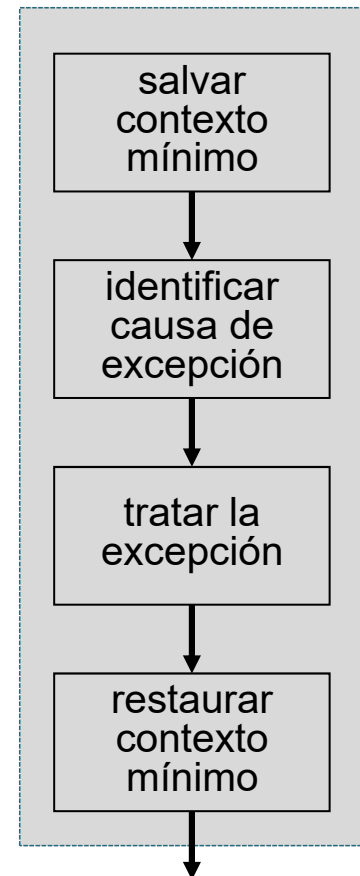
Curso 2017-2018

54

# Diseño del manejador

- **Gestión del contexto**
  - ✓ El manejador se ha de poder ejecutar de manera que no afecte el programa
- **Identificación de causa y tratamiento**
  - ✓ Cualquier causa de excepción provoca la ejecución del manejador
- **Atomicidad**
  - ✓ La ejecución del manejador no se puede interrumpir
  - ✓ El código ha de ser correcto (para que no provoque excepciones)
  - ✓ Las interrupciones están inhibidas

## Manejador



Curso 2017-2018

55

# Variables del manejador

- **Necesidades:**
  - ✓ Espacio para guardar temporalmente el contexto mínimo
    - Registros usados en el manejador \$t0, \$t1 y \$at (usado por el ensamblador)
    - Espacio para ubicar la dirección de retorno
  - ✓ Variables que hagan falta para cada tratamiento
- **Registros dedicados**
  - ✓ Los registros \$k0 y \$k1 están dedicados al sistema operativo y no forman parte del contexto del programa
    - \$k0 actuará como registro temporal del manejador y contendrá la dirección de retorno en el último momento
    - \$k1 contendrá la dirección base del contexto mínimo

```

.kdata
## espacio para contexto mínimo
salvareg: .word 0,0,0    # para salvar $t0,$t1 y $at
dirret:   .word 0        # para salvar la dir. de retorno
  
```

Curso 2017-2018

56



# Gestión del contexto mínimo

Salvar  
contexto  
mínimo

```
.ktext 0x80000080 # Punto de entrada al manejador
sw $at, 0($k1)      # Salvo $at
sw $t0, 4($k1)      # $t0 contendrá direcciones
sw $t1, 8($k1)      # $t1 contendrá datos
mfc0 $k0, $14        # leo EPC
sw $k0, dirret       # guardo dirección de retorno
```

Identificar  
causa de  
excepción

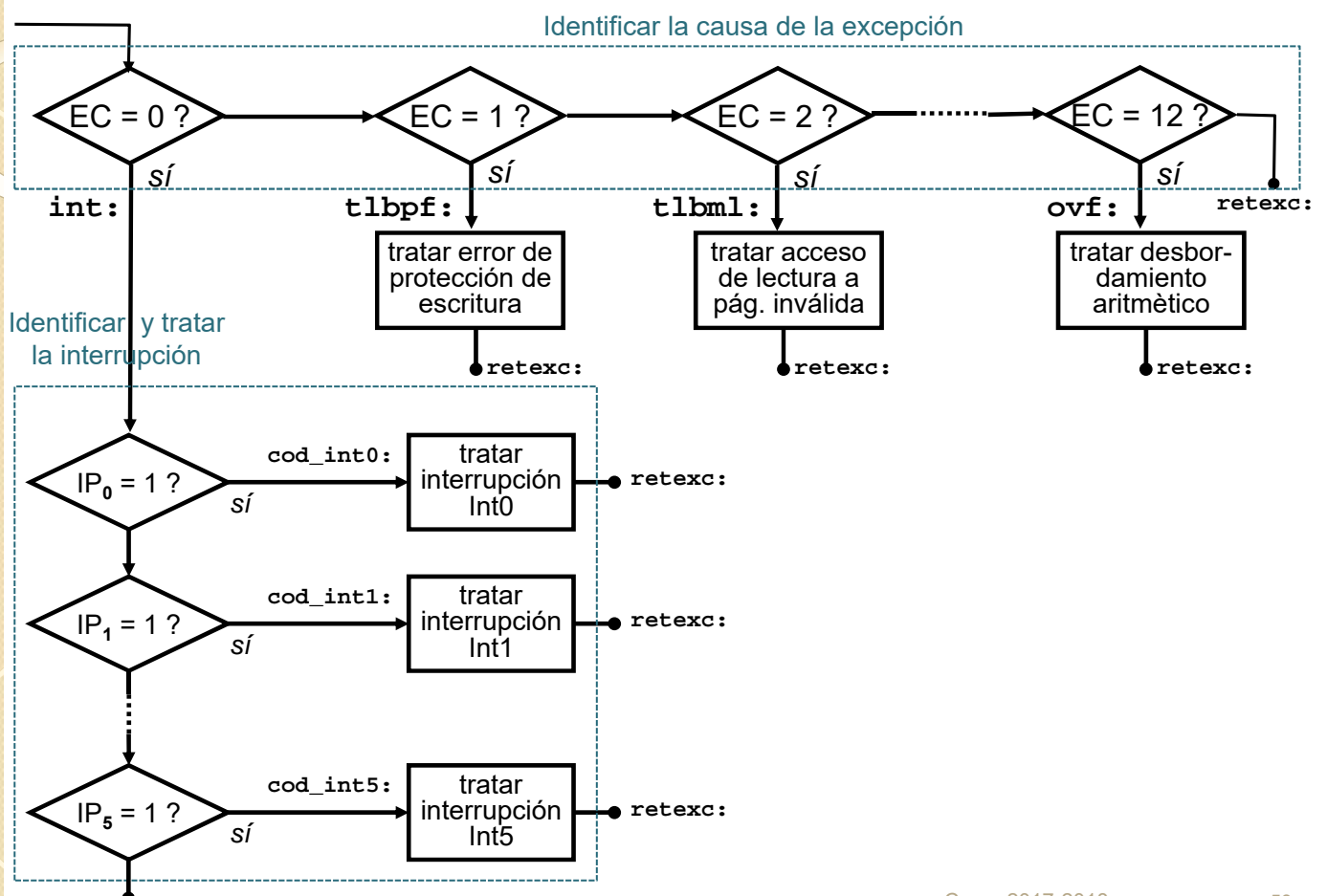
Tratar la  
excepción

Restaurar  
contexto  
mínimo

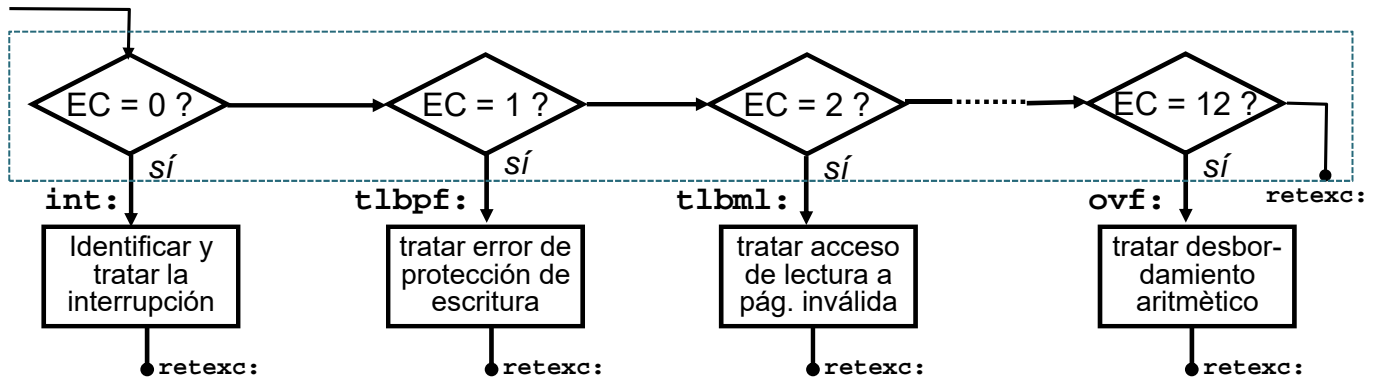
```
retexc: lw $k0, dirret      # dirección de retorno en $k0
        lw $at, 0($k1)      # restaura $at
        lw $t0, 4($k1)      # restaura $t0
        lw $t1, 8($k1)      # restaura $t1
        rfe                  # paso a modo usuario y IE=1
        jr $k0              # vuelvo al programa
```

Consideraremos que las dos últimas instrucciones (*rfe* y *jr*) forman un bloque indivisible. Esto evitará el procesamiento de una petición de interrupción al finalizar *rfe*.

## Manejador de excepciones: Diagrama de flujo



# Identificar la causa de la excepción



CR

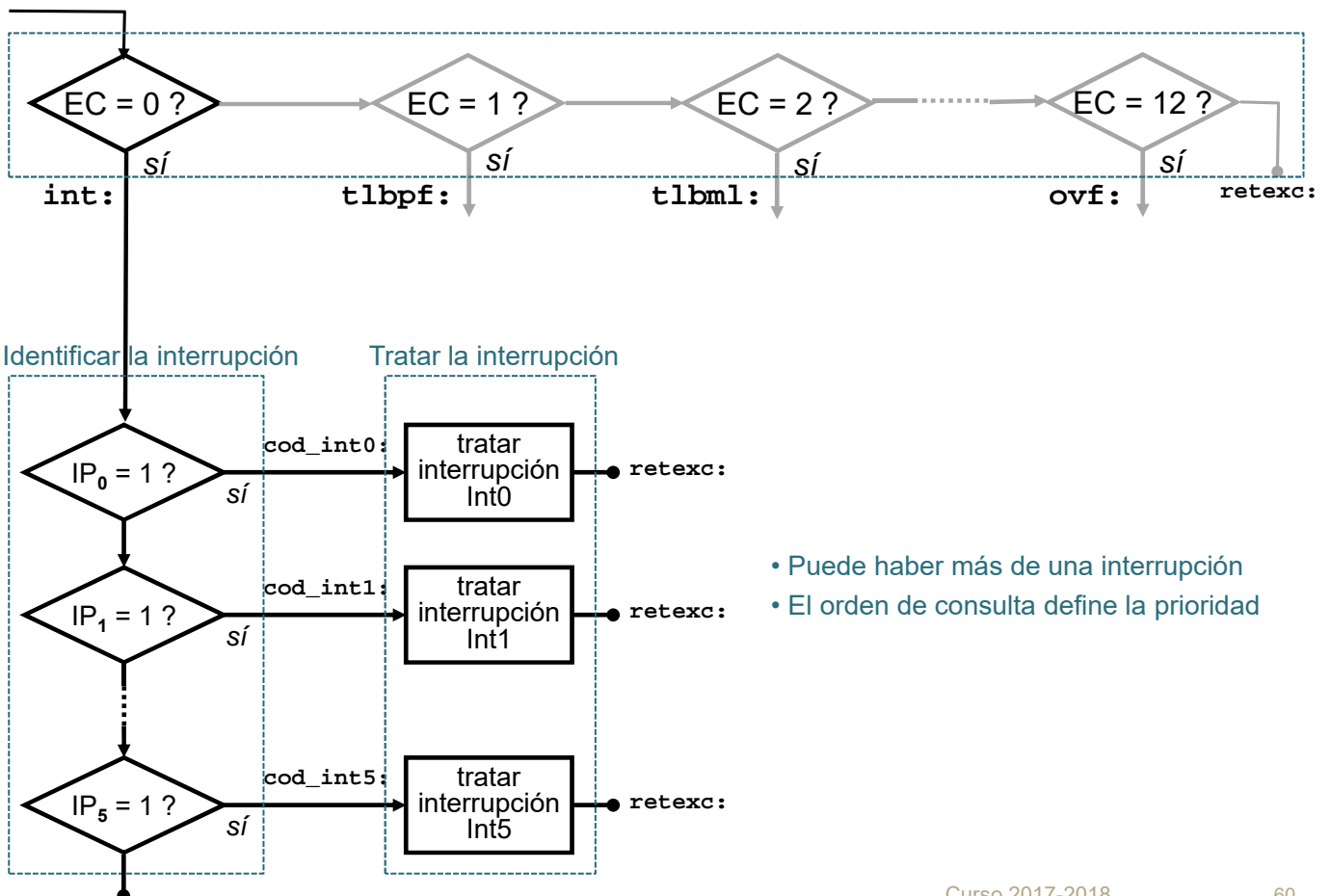
Código de excepción



```

mfc0 $k0, $13          # Lee registro de Causa
andi $t0, $k0, 0x003c  # Aísla el código*4
beq $t0, $zero, int     # Compara con 0 y salta a int:
li $t1, 4               # Código 1*4
beq $t0, $t1, tlbpf     # compara y salta si igual
li $t1, 8               # Código 2*4
beq $t0, $t1, tlbml     # compara y salta si igual
...
li $t1, 0x30            # Código 12*4
beq $t0, $t1, ovf       # compara y salta si igual
b retexc
    
```

# Identificar la interrupción



# Identificar la interrupción: ensamblador

```

int:
    andi $t0, $k0, 0x400      # miro IP0
    bne  $t0, $zero, cod_int0 #
    andi $t0, $k0, 0x800      # miro IP1
    bne  $t0, $zero, cod_int1 #
    ...
    andi $t0, $k0, 0x8000     # miro IP5
    bne  $t0, $zero, cod_int5 #
    b retexc                   # Interrupción espúrea

cod_int0: ### código de tratamiento interrupción int0*
    ...
    b retexc ### fin del código de tratamiento int0*

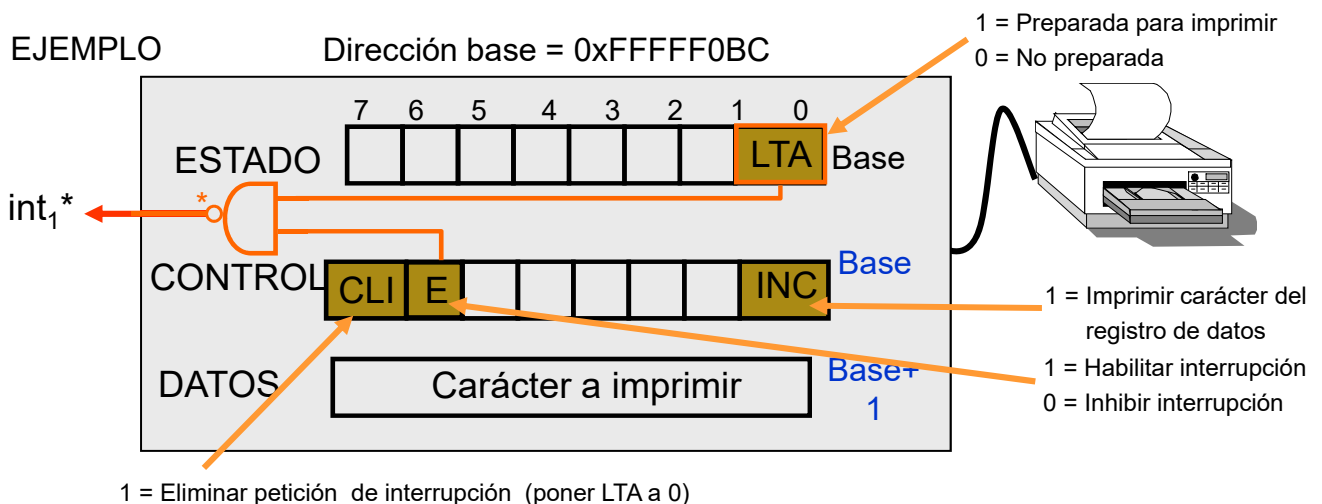
cod_int1: ### código de tratamiento interrupción int1*
    ...
    b retexc ### fin del código de tratamiento int1*

cod_int5:  ### código de tratamiento interrupción int5*
    ...
    b retexc ### fin del código de tratamiento int5*
    
```

Curso 2017-2018

61

## Tratamiento de la interrupción



```

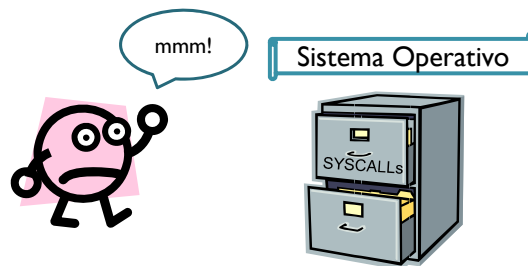
cod_int1:      ### Código de tratamiento de la int. de impresora
               ### Para llegar aquí: IEC=IM1=E=1 (int. habilitada)
li    $t0,0xFFFFF0BC    # Dirección base del dispositivo
lb    $t1,car_A          # Lee carácter a imprimir de memoria
sb    $t1,1($t0)         # Escribe carácter en Registro de Datos
li    $t1,0xC1           # Coloca a 1 los bits 0, 6 y 7: CLI, E, INC
sb    $t1,0($t0)         # Escribe R. Control la palabra 11000001
b retexc              # Fin del código de tratamiento
    
```

Curso 2017-2018

62

## 7– La entrada/salida mediante el Sistema Operativo

- Enlace entre código de usuario y el SO
- La instrucción 'syscall'
- Implementación de las funciones del SO



## Excepciones y sistema operativo

- Las excepciones permiten resolver de forma homogénea muchas de las necesidades de los sistemas operativos:
  - ✓ El hardware gestiona de forma simple y segura los modos de ejecución de usuario y supervisor.
  - ✓ Son una forma general de evento que provoca la atención del sistema operativo.
  - ✓ Simplifican el cambio de contexto durante la conmutación de procesos.
  - ✓ Implementan el cambio de estado de los procesos (activo, en espera, etc...).

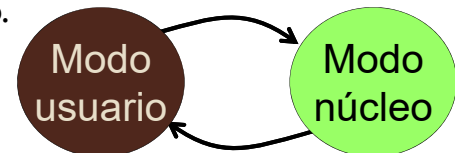
# Excepciones y sistema operativo

## • Organización de un sistema operativo

- El sistema operativo está formado por diversos fragmentos de código que gestionan el procesador, la memoria y la entrada/salida.
- Los programas corrientes se ejecutan en modo usuario.
- Un conjunto de eventos provocan la ejecución de fragmentos de código del sistema operativo en modo supervisor.

## • El sistema operativo gestiona la entrada/salida

- Las interfaces de los periféricos se mapean en direcciones restringidas y sólo son accesibles en modo supervisor.
- Las interrupciones provocan el cambio al modo supervisor y la ejecución del manejador incluido en el sistema operativo.



Curso 2017-2018

65

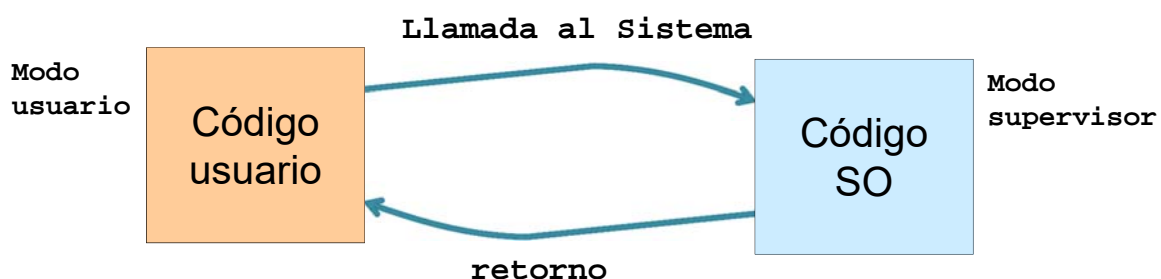
# Enlace entre código de usuario y el SO

## • Independencia entre los programas de usuario y los del sistema operativo.

- Los programas de usuario han de ejecutarse bajo diferentes configuraciones de un computador y con diferentes versiones del sistema operativo.

## • Las **llamadas al sistema** permiten enlazar de forma segura los programas de usuario y el código del sistema operativo.

- Los programas provocan la ejecución de código del sistema mediante instrucciones de llamada al sistema.



Curso 2017-2018

66

# Funciones del sistema operativo

- Mecanismo de llamada

- ✓ Para llamar una función, del SO los programas han de construir unos parámetros (o argumentos) y ejecutar una instrucción de llamada al sistema (syscall, INT n, ...).
- ✓ Cuando el programa recupere el control encontrará los resultados apropiados.
- ✓ Parámetros y resultados:
  - Los pequeños (una palabra o menos) se suelen transmitir mediante los registros del procesador.
  - Los grandes (i.e. una cadena de caracteres) se suelen transmitir mediante un área de la memoria y un registro con un puntero.
- ✓ En cada SO se especifica el conjunto de funciones disponibles y para cada una de ellas los parámetros apropiados y los resultados.

## ¿ Cómo se implementan las llamadas al sistema ?

- Algunos procesadores hacen uso del mecanismo de excepciones:
  - ✓ MIPS Arch. → instrucciones **SYSCALL** / **RFE** → Excepción 8
  - ✓ INTEL Arch. (IA-32) → instrucciones **INT n** / **IRET** → Excepciones 32.. 255
  - ✓ DEC ALPHA Arch. → instrucción **CALL\_PAL trap\_num** → Varios números
  - ✓ POWER Arch. → instrucciones **SC (system call)** → Interrupción 8
- Otros procesadores incorporan instrucciones específicas para este propósito:
  - ✓ INTEL (IA-64) → instrucciones **SYSCALL** / **SYSRET** y **SYSEENTER** / **SYSEXIT**



# Caso de estudio: MIPS R2000

- La instrucción **SYSCALL**

- ✓ Provoca la excepción con el código número 8 en el Registro de Causa (CR, Cause Register).
- ✓ Por convenio se usa el registro \$v0 como índice que identifica la función del sistema que se solicita.
  - Ejemplo con una función de índice 45, que pide un argumento en \$a0 y deja un resultado en \$v0.

## Programa de usuario

```
...  
li $v0,45  
li $a0,argumento  
syscall  
# resultado está en $v0  
...
```

## Manejador de excepciones

```
si (causa = syscall)  
    && ($v0 = 45) /* índice de función  
    /* en $a0 está el argumento */  
    { operar }  
    { dejar resultado en $v0 }  
fin de manejador
```

# Funciones implementadas en el PCSpim

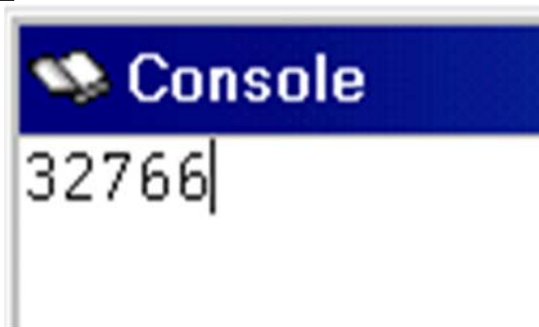
Función	Código	Argumentos	Resultado
print_int	\$v0=1	\$a0 = entero	
print_float	\$v0=2	\$f12 = coma flotante	
print_double	\$v0=3	\$f12 = doble precisión	
print_string	\$v0=4	\$a0 = puntero a cadena	
read_int	\$v0=5		Entero (en \$v0)
read_float	\$v0=6		Coma flotante (en \$f0)
read_double	\$v0=7		Doble precisión (en \$f0)
read_string	\$v0=8	\$a0 = puntero a cadena \$a1 = longitud	
print_char	\$v0=11	\$a0 = carácter	
read_char	\$v0=12		Carácter (en \$v0)

# Ejemplo de llamada al sistema en PCSPim

## Programa de usuario

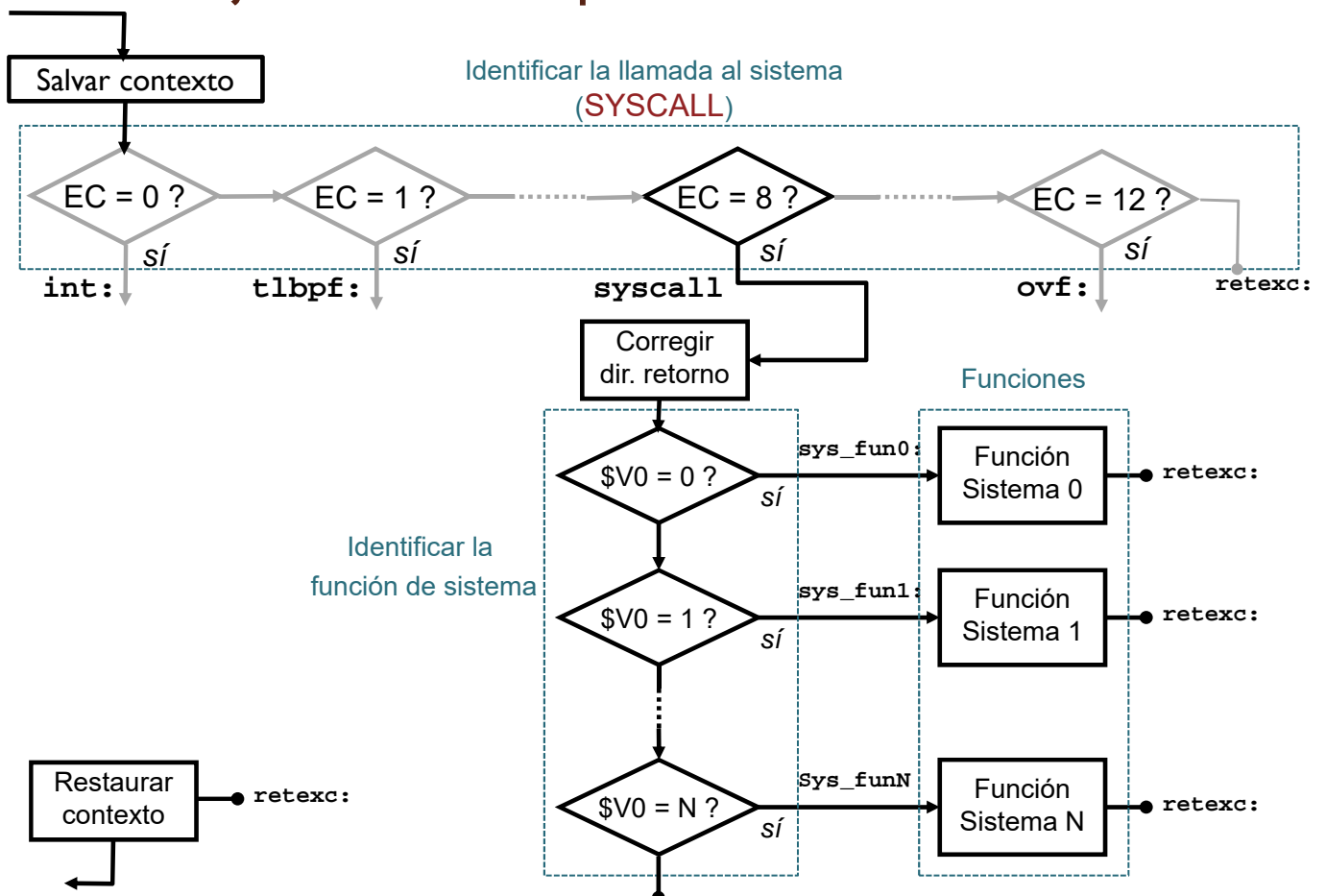
```
li $v0,1          # Índice 1: print_int (a la consola)
li $a0,0x7ffe      # Valor del entero
syscall           # Invocación a syscall
```

## Efecto



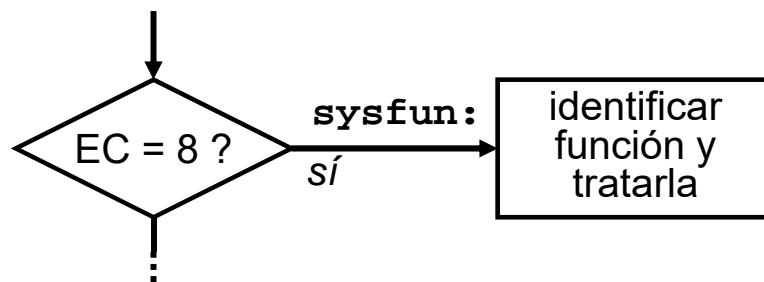
$$7FFE_{16} = 32766_{10}$$

## El manejador de excepciones



## Identificación de la llamada al sistema

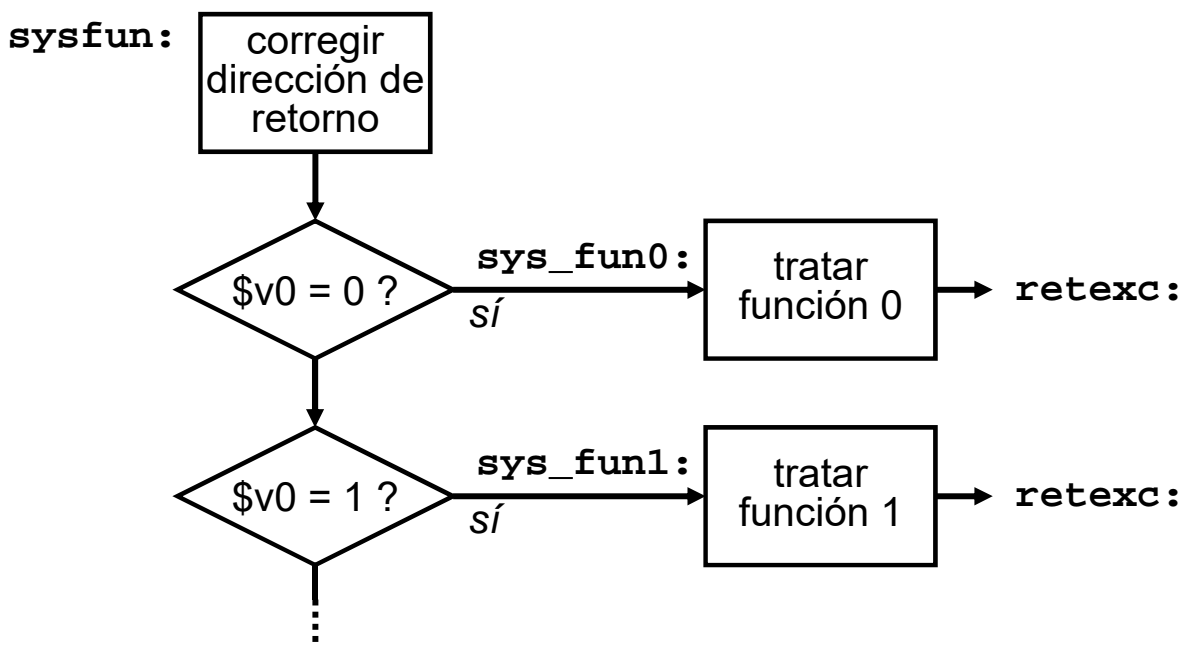
- En la sección de análisis del código de excepción, el código n° 8 es el que corresponde a **SYSCALL**.



```
mfc0 $k0, $13          # Lee registro de Causa
andi $t0, $k0, 0x003c   # Aisla el código*4
beq $t0, $zero, int     # Compara con 0 y salta
li $t1, 4               # Código 1*4
beq $t0, $t1, tlbf     # compara y salta si igual
...
li $t1, 0x20           # Código 8*4
beq $t0, $t1, sysfun    # compara y salta si igual
...
```

## Identificar la función de sistema

- Hay que corregir la dirección de retorno, identificar el código presente en \$v0 y tratar cada una por separado.



# ¿Porqué hay que modificar la dirección de retorno?

- El saltar al manejador, el registro EPC apunta a la primera instrucción que no se ha podido completar.
- Casos significativos:
  - ✓ En caso de interrupción hay que volver a EPC.
  - ✓ En caso de fallo de memoria virtual, EPC apunta a la instrucción causante y hay que volver a ella para que se ejecute de nuevo después de que el SO cargue la página implicada.
  - ✓ En caso de llamada al sistema EPC apunta a la instrucción SYSCALL pero hay que volver a la siguiente (EPC+4).



Hay que incrementar en 4 el valor del EPC  
o su copia en la variable del manejador 'dirret'

## Tratamiento de las funciones: ensamblador

```

Sysfun: ### tratamiento de syscall
        # corrige la dirección de retorno
        lw $t0,dirret
        addi $t0,$t0,4                # dirret = dirret + 4
        sw $t0,dirret

        # salta según el índice presente en $v0
        beq $v0, $0, sys_fun0
        li $t0, 1
        beq $v0, $t0, sys_fun1
        li $t0, 2
        beq $v0, $t0, sys_fun2
        ...
        ...
Sys_fun0: ### tratamiento de la llamada al sistema con índice 0
        ...
        b retexc

Sys_fun1: ### tratamiento de la llamada al sistema con índice 1
        ...
        b retexc

```

## Caso I. Funciones de acceso a información

- Especificación

- ✓ **P\_model:** Devuelve un código que identifica el procesador.
- ✓ **Sys\_ver:** Devuelve el código de la versión del SO.

- Comentarios

- ✓ Para una instalación dada estos dos códigos son constantes que están en variables del sistema; el manejador no tiene que calcular nada.
- ✓ El tratamiento no ha de acceder a ningún periférico.

Función	Código	Argumentos	Resultado
<b>P_model</b>	\$v0 = 9991	—	\$v0 = código de procesador
<b>Sys_ver</b>	\$v0 = 9992	—	\$v0 = código de versión del SO

## Caso I. Implementación del manejador

```

Sysfun:  lw $t0,dirret
         addi $t0,$t0,4    # dirret = dirret + 4
         sw $t0,dirret

# Salta según el índice en $v0
         li $t0, 9991      # ¿$v0 = 9991?
         beq $t0, $v0, P_model # Salta a P_model
         li $t0, 9992      # ¿$v0 = 9992?
         beq $t0, $v0, Sys_ver # Salta a Sys_ver
         ...

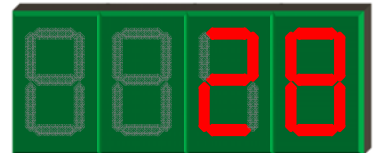
P_model: lw $v0, codigo_model_proc
         b retexc

Sys_ver: lw $v0, codigo_version
         b retexc

```

## Caso 2. Acceso a la interfaz de un periférico

- ✓ Dado que los periféricos no son accesibles en modo usuario, el acceso a los registros de su interfaz se ha de hacer mediante funciones del sistema.
- ✓ Ejemplo con el periférico de E/S directa (visualizador de 4 dígitos) en la dirección base 0xFFFF0020.



Nombre	Dir.	Acceso	Estructura
Órdenes	DB+0	Escr.	

Activa el Visualizador y el parpadeo:

- ON (bit 0): a 0 apagado , a 1 visualiza VALOR
- Frec (bits 6..4): frecuencia de parpadeo en Hz
- Frec = 0: continuo

Datos	DB+4	Escr.	
-------	------	-------	--

VALOR a visualizar:

Número de 8 bits en Ca2 (-127 .. +127)

El valor de -128 visualiza '- E r r'

Se visualiza al poner el bit ON a 1

## Caso 2. Acceso a la interfaz de un periférico

- Funciones del visualizador:
  - ✓ Display\_show: Muestra un valor en el visualizador.
  - ✓ Display\_blink: Define el parpadeo en el visualizador.
  - ✓ Display\_clear: Apaga el visualizador

Función	Código	Argumentos	Resultado
Display_show	\$v0 = 9980	\$a0 = value	Se muestra 'value' en el visualizador
Display_blink	\$v0 = 9981	\$a0 = Frecuencia (0 = continuo)	Define la frecuencia a la que parpadeará el visualizador a partir de ese instante
Display_clear	\$v0 = 9982	-----	Apaga el visualizador



## Caso 2. Detalle del tratamiento

```

        .kdata
Blink:  .byte  0  # Frecuencia actual de parpadeo

Display_show:  # Muestra el valor en $a0
        la $t0,0xFFFF0020  # VISUALIZADOR
        sb $a0, 4($t0)      # reg datos = valor
        lb $t1, blink
        ori $t1, $t1, 0x01
        sb $t1,0($t0)       # orden al reg de control
        b retexc

Display_blink:  # Parpadeo a la frecuencia en $a0
        la $t0,0xFFFF0020
        sll $t1,$a0,4
        sb $t1, blink
        ori $t1, $t1, 0x01
        sb $t1,0($t0)       # orden al reg de control
        b retexc

Display_clear:  # apaga el visualizador
        la $t0,0xFFFF0020  # VISUALIZADOR
        sb $0, 0($t0)       # reg control = 0 apaga
        b retexc

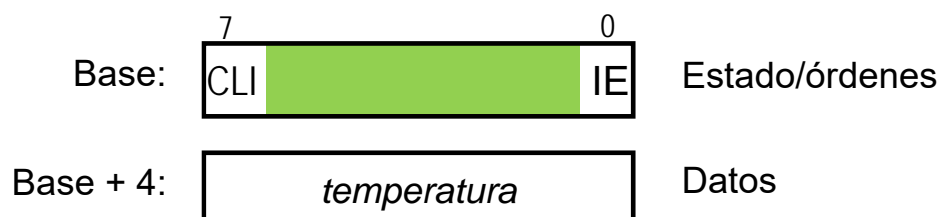
```

Curso 2017-2018

81

## Caso 3. Interrupciones sin espera

- Un sensor de temperatura: termómetro
  - ✓ Ante un cambio de temperatura el periférico provoca una interrupción que afecta a una variable privada del núcleo del SO.
  - ✓ Especificación de la interfaz del periférico:
  - ✓ Base **0xFFFFB9000**, interrupción por la línea **int4\***



Función	Código	Argumentos	Resultado
Get_Temp	\$v0 = 9975	—	\$v0 = temperatura

Curso 2017-2018

82

## ¿Cómo consulta la temperatura el usuario?

- Utiliza una llamada al sistema **Get\_Temp** con código 9975, y el valor de la temperatura obtenido lo guarda en memoria y lo imprime en la consola mediante la llamada **'print\_int'**.

```

.data
valor: .byte 0

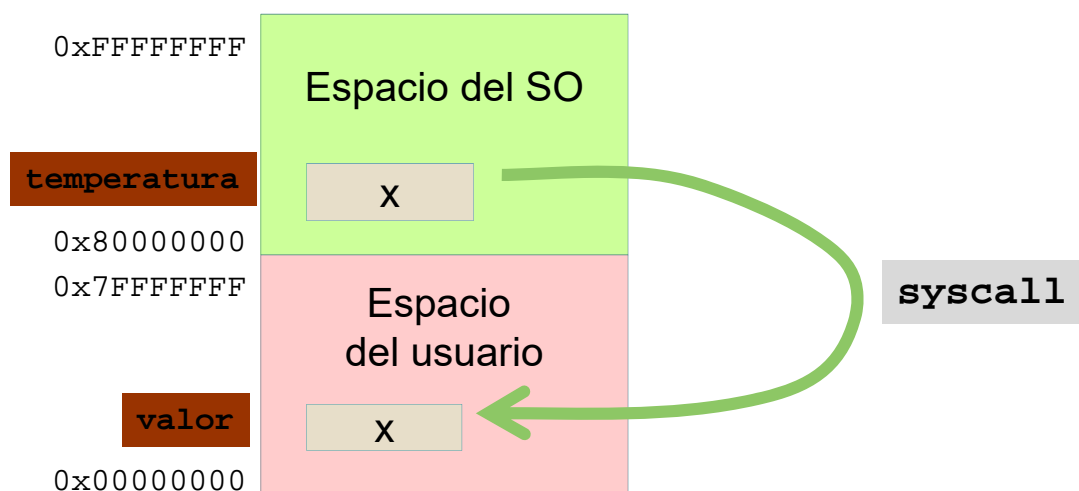
.text
# Lee la temperatura
li $v0, 9975
syscall          # Deja la temperatura en $v0
sb $v0, valor    # La almacena en memoria

# Ahora la imprime en la consola
move $a0, $v0    # Argumento en $a0
addi $v0, $0, 1  # Código 1 para print_int
syscall          # Impresión en la consola

```

## Intercambio entre los dos espacios

- ✓ El valor de la temperatura se ha copiado desde el espacio del SO hasta el del usuario mediante la llamada al sistema **Get\_Temp**.
- ✓ El valor de la variable **temperatura** lo actualizará la rutina de la interrupción **Int\_4\***. Se supone que el código de inicialización del sistema ha habilitado dicha interrupción.



## Caso 3. Detalle del tratamiento

```

        .kdata
temperatura:  .byte 0 # Variable privada del SO

        .ktext
        ...

```

```

# En la sección de tratamiento de interrupciones
int4:      la $t0,0xFFFFB9000    # Dirección base
           lb $t1,4($t0)         # Lee nueva temperatura
           sb $t1,temperatura    # Escribe temperatura
           li $t1,0x81           # Máscara para CLI=1
           sb $t1,0($t0)         # Cancelo interrupciones
           b retexc
           ...

```

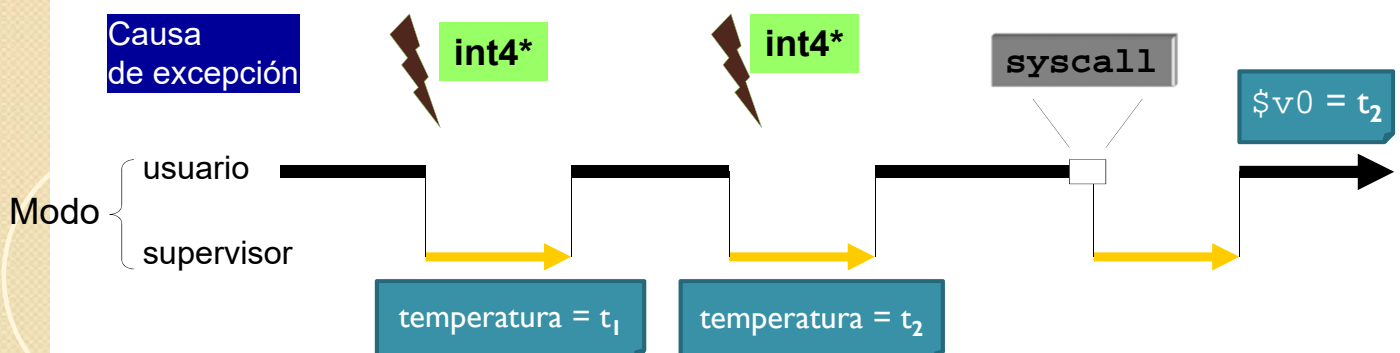
```

# En la sección de llamadas al sistema
Get_Temp:  lb $v0,temperatura     # Lee temperatura tomada
           b retexc
           ...

```

## Caso 3. Ejemplo de evolución de las variables

- El sensor provoca interrupciones que actualizan la variable **temperatura** almacenada en memoria.
- Al llamar a la función **Get\_Temp** el programa obtiene en \$v0 la copia del último valor anotado en temperatura.
- En el cronograma: el sensor hace dos interrupciones con dos valores  $t_1$  y  $t_2$  antes de que el programa haga la consulta.



## Entrada/Salida con espera

- ¿Cómo hacer que un proceso espere a que se produzca una operación de E/S?
  - ✓ Mediante el planificador de tareas del SO.
  - ✓ Las excepciones pueden ayudar al planificador de tareas.
- Las excepciones pueden:
  - ✓ Detener un proceso en ejecución.
  - ✓ Devolver un proceso detenido a ejecución:
    - Por ejemplo el proceso detenido por la excepción.
    - O cualquier otro proceso → CONMUTACIÓN DE PROCESOS.

## Excepciones y conmutación de procesos

- Casos significativos de conmutación de procesos mediante excepciones:
  - ✓ Una llamada al sistema puede detener un proceso, enviándolo a la cola de procesos detenidos.
    - Por ejemplo, una llamada de E/S que se debe esperar a que la operación de E/S termine.
  - ✓ Con planificación cíclica (round robin), la interrupción de reloj puede enviar el proceso detenido a la cola de procesos preparados cuando ha expirado el cuanto (quantum) de tiempo.
  - ✓ Una interrupción de periférico puede reactivar un proceso en espera (distinto del detenido) y pasarlo a la cola de preparados.
  - ✓ Errores fatales (aritméticos, de direccionamiento, etc.).
  - ✓ Fallos de página suspenden el proceso y lo envían a una cola de espera.

## El cambio de contexto

- El manejador de excepciones conserva el contexto máquina del proceso detenido, compuesto de:
  - ✓ La posición de memoria etiquetada como 'salvareg' sirve para preservar los registros \$at, \$t0 y \$t1 del proceso.
  - ✓ El resto de registros de propósito general que aún conservan su valor.
  - ✓ El valor del EPC (contador de programa del proceso interrumpido).
  - ✓ Otras informaciones acerca del proceso detenido se encuentran en su BCP (Bloque de control de proceso).
- El manejador puede cambiar el contexto máquina de un proceso P por el contexto de otro proceso Q:
  - Tendrá que transferir el contexto del proceso P desde los registros al BCP correspondiente.
  - Tendrá que transferir el contexto del proceso Q desde su BCP a los registros.

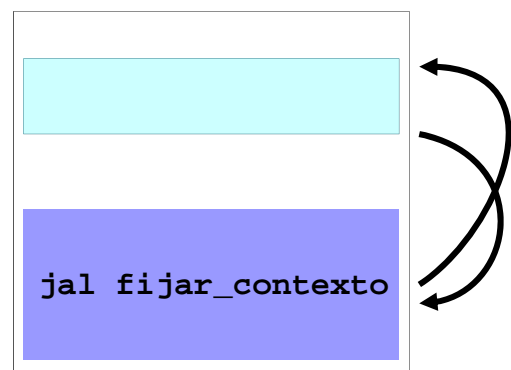
Curso 2017-2018

89

## Planificación de procesos

- Supondremos que hay planificación de procesos en el código del manejador, mediante el uso de tres primitivas:
  - ✓ fijar\_contexto
  - ✓ suspende\_esto\_proceso
  - ✓ activa\_proc\_en\_espera
- Las primitivas están implementadas como subprogramas sin parámetros.

fijar\_contexto:



Curso 2017-2018

90

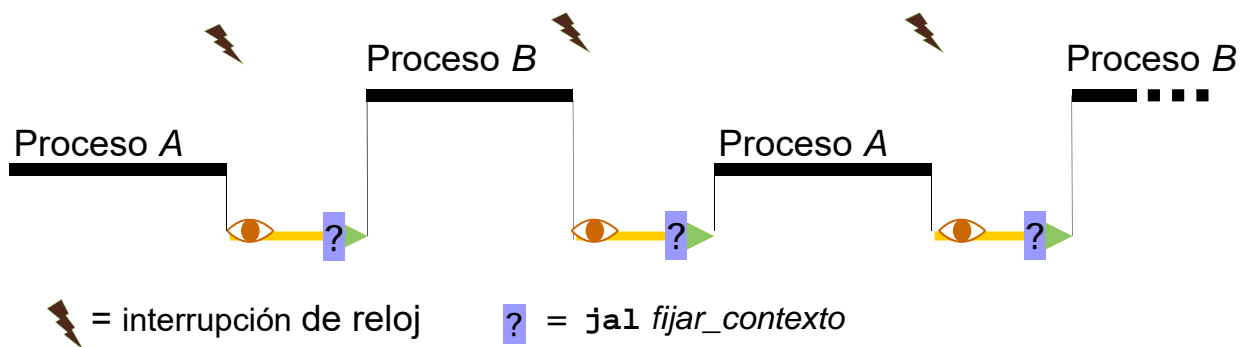
# Cambio de contexto

- **fijar\_contexto**

- ✓ Es el proceso del planificador de tareas que determina cuál es el proceso activo que entra en ejecución al final del manejador.
- Si el proceso escogido es el mismo que se ha detenido por la excepción, no hace ningún cambio de contexto.
- Si el proceso escogido es distinto al detenido, hará el cambio completo de contexto (registros, dirección de retorno y otros datos) y gestionará la cola de procesos preparados.

## ¿Dónde se puede hacer el cambio de contexto?

- En el retorno del manejador de excepciones



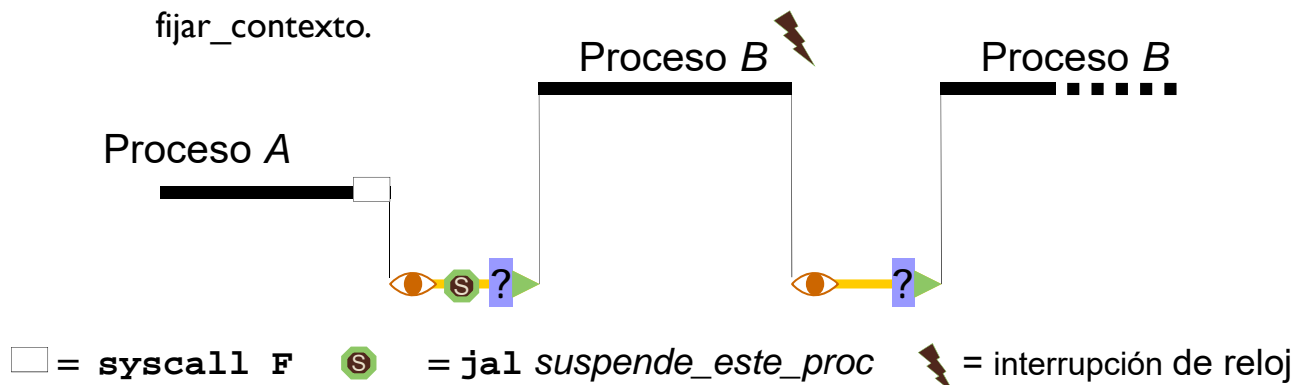
```
retexc: jal fijar_contexto # Posible cambio de contexto
        lw $k0, direton   # dir de retorno en $k0
        lw $at, 0($k1)    # )
        lw $t0, 4($k1)    # ) restaura contexto mínimo
        lw $t1, 8($k1)    # )
        rfe               # paso a modo usuario
        jr $k0            # vuelvo al programa
```



# Suspensión de un proceso

- `suspende_este_proc`

- ✓ Cambia a **suspendido** el estado del proceso interrumpido e inserta su BCP en la cola de E/S correspondiente.
- ✓ Ejemplo: función `syscall F` que permite al proceso A esperar hasta que el periférico P esté preparado.
  - El tratamiento de F suspende el proceso A.
  - Mientras A no cambie de estado, este proceso nunca será escogido por `fijar_contexto`.



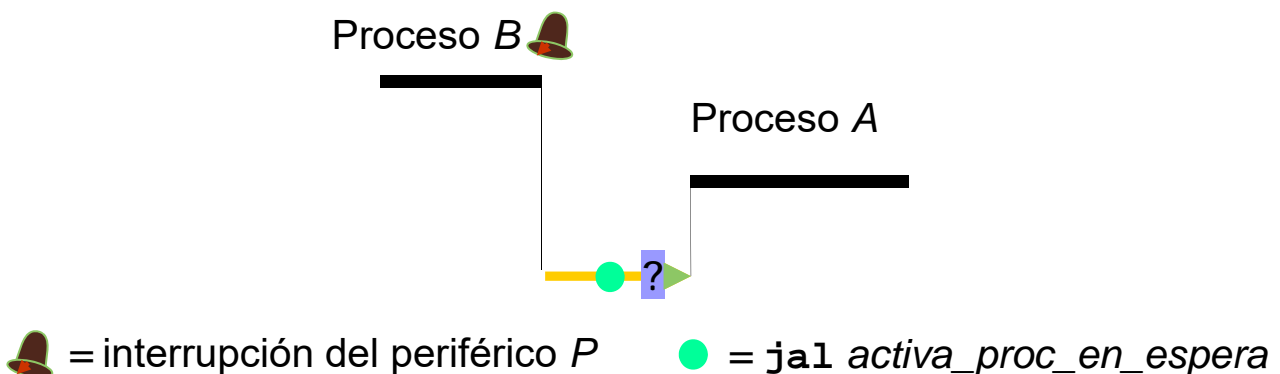
Curso 2017-2018

93

# Reactivación de un proceso

- `activa_proc_en_espera`

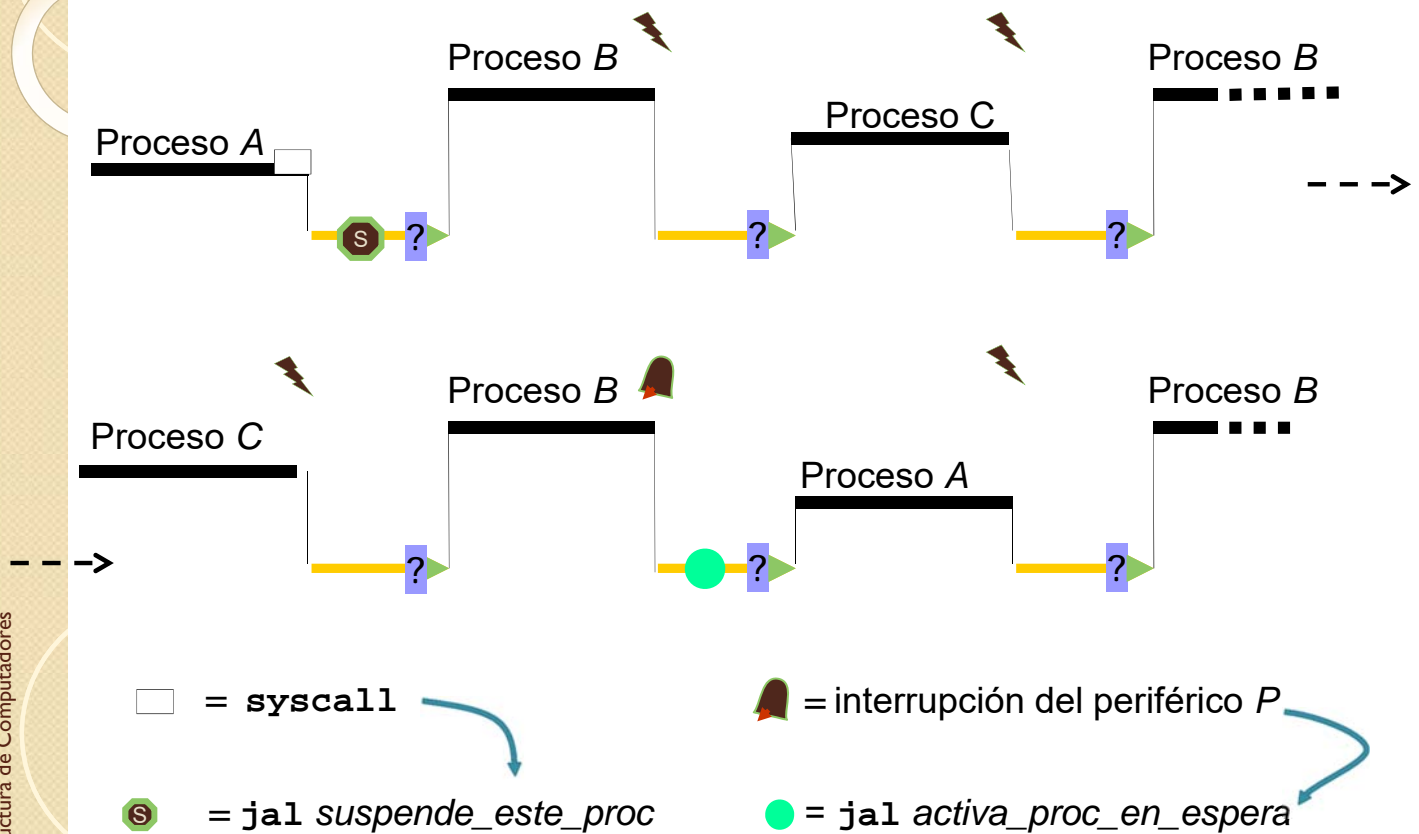
- ✓ Cambia a **activo** el proceso que se encuentra esperando a un periférico e inserta su BCP en la cola de procesos preparados.
- ✓ Continúa el ejemplo anterior:
  - El tratamiento de la interrupción del periférico P tendrá que reactivar el proceso A.
  - El planificador podrá escogerlo de nuevo y proseguir su ejecución.



Curso 2017-2018

94

# Ejemplo



Curso 2017-2018

95

## Caso 4. Entrada con espera

- **Periférico:**
  - El sensor que provoca una interrupción en int4\* al cambiar la temperatura.
- **Función 'get\_temp\_wait' :**
  - Permite que un programa espere hasta que el sensor suministre una temperatura nueva. Devuelve la temperatura en \$V0.
  - Debe ejecutar el procedimiento 'suspende\_este\_proceso' para suspender el programa y dejarlo en la cola de espera del sensor.
- **Interrupción:**
  - El procedimiento 'transmite\_valor' modifica el contenido de \$v0 en el contexto máquina de los posibles procesos (0, 1 o más) que están esperando al sensor.
  - El procedimiento 'activa\_proc' contiene el código que hace activos los posibles procesos que esperan al sensor.

Curso 2017-2018

96

## Caso 4. Implementación de la función

- Sólo ha de cambiar el estado del proceso:
  - ✓ El proceso queda en espera de una nueva lectura de temperatura.

```

                .kdata
temperatura:   .byte 0

                .ktext
                ...
                # Función del sistema
get_temp_wait:
                jal suspende_este_proceso
                b retexc
                ...

```

- ✓ En 'retexc' el planificador conmutará el contexto porque el proceso detenido ya no está activo.

## Caso 4. Manejo de la interrupción

- El tratamiento debe transmitir el valor a los procesos que esperan y cambiar su estado.
- Cuando acabe el manejador, cualquiera de estos procesos puede entrar en ejecución.

```

                .ktext
                # Tratamiento de la interrupción int4*
int4:          li $t0,0xFFFFB9000 # Dirección base
                lb $t1,4($t0)       # Lee temperatura
                sb $t1,temperatura($0) # Escribe temperatura

                jal transmite_valor
                jal activa_proc_en_espera

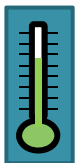
                li $t1,1             # Máscara para IE=1
                sb $t1,0($t0)        # Habilita interrupciones
                b retexc

```

## Tema 8

# FIN

## Ejercicio: Termómetro + Visualizador



```
LeeTemp:  la $t0,0xFFFF0010    # TERMÓMETRO
          li $t1, 0x01
          sb $t1, 4($t0)        # Orden adquirir
CdE:      lb $t1,0($t0)          # Espera
          andi $t1,$t1,1        # R = 1
          beqz $t1,CdE
          li $t1,0x20
          sb $t1,4($t0)        # Cancela
          lb $t2,8($t0)        # Lee temp.
          la $t0,0xFFFF0020    # VISUALIZADOR
          sb $t2, 4($t0)        # Valor = temp
          li $t3,100           # Si temp<=100
          bgt $t2,$t3,intermit
          li $t1, 0x01          # Mostrar valor
          sb $t1,0($t0)        # continuo
          jr $ra
intermit: li $t1,0x21           # si no
          sb $t1,0($t0)        # mostrar inter-
          jr $ra               # mitente
```

## Ejemplo 4: Termómetro + visualizador



- Programación (solo la rutina de servicio)

- ✓ Leer temperatura
- ✓ Mostrar en el visualizador
- ✓ Si temperatura > 100 °C:
  - mostrar con intermitencia (2 Hz)
- ✓ Habilitación inicial de las interrupciones

```

RdS:  la $t0,0xFFFF0010    # TERMÓMETRO
      li $t1,0x28
      sb $t1,4($t0)         # Cancela Intr
      lb $t2,8($t0)         # Lee temp.
      la $t0,0xFFFF0020    # VISUALIZADOR
      sb $t2, 4($t0)        # Valor = temp
      li $t3,100            # Si temp<=100
      bgt $t2,$t3,intermit
      li $t1, 0x01          # Mostrar valor
      sb $t1,0($t0)         # continuo
      return int
intermit: li $t1,0x21       # si no
          sb $t1,0($t0)     # mostrar inter-
          return int       # mitente
  
```

Comparar