

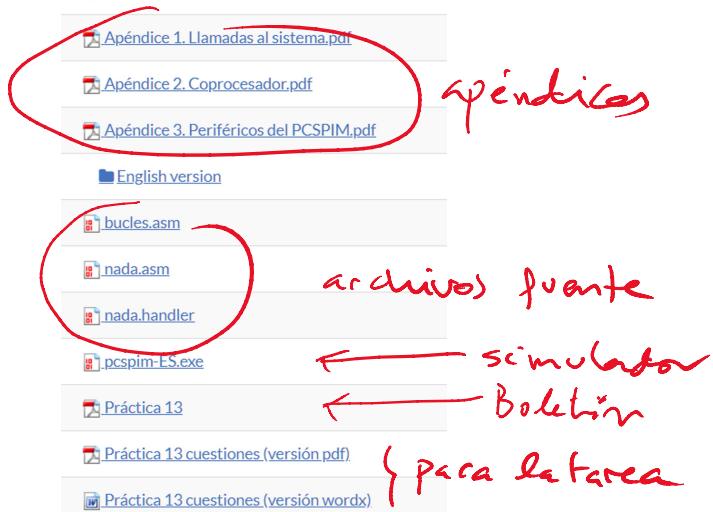
Práctica 13: Sincronización por interrupciones

viernes, 27 de marzo de 2020 19:14

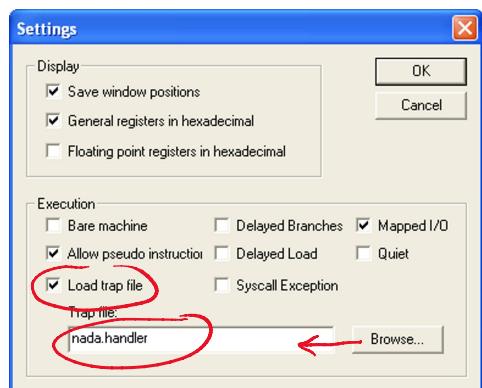
TAREA 1 - Preparación de la práctica

Descargamos los archivos de la práctica

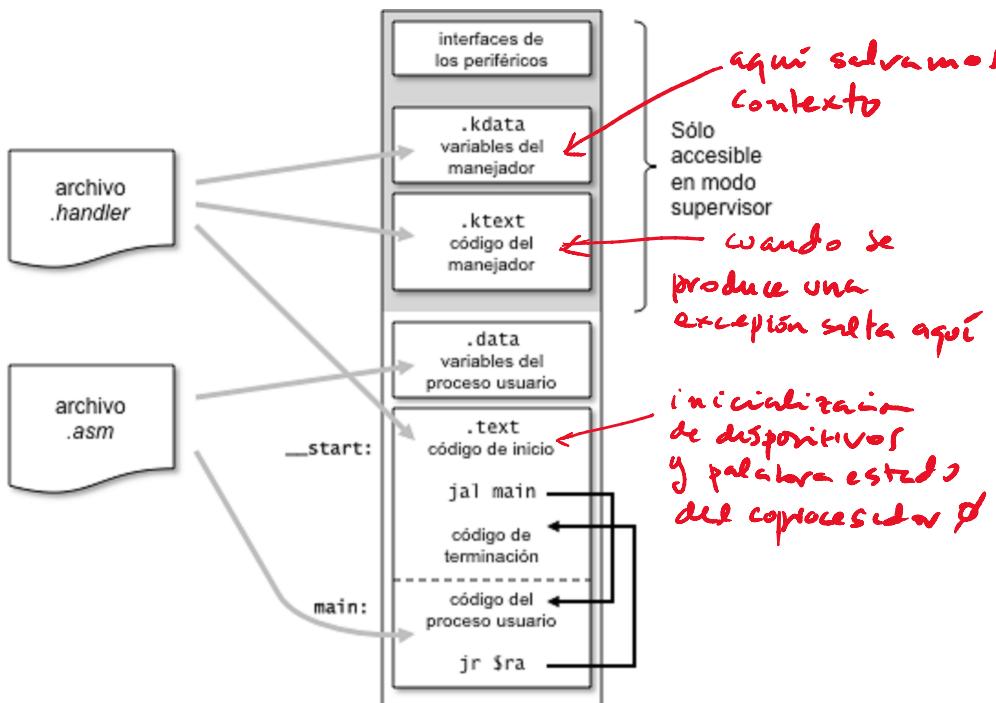
P13



Preparación del simulador



Ubicación en memoria del código y los datos de los archivos .handler y .asm



Manejador inicial nada.handler

```
#####
## ÁREA DE DATOS DEL MANEJADOR ##
#####

.kdata
var1: .word 0xFFFFFFFF

#####
## CÓDIGO DEL MANEJADOR ##
#####

.ktext 0x80000080 dirección de las excepciones
## Preserva el contexto del programa de usuario
## (nada de momento)

## Identifica y trata excepciones
## (nada de momento)

## Restaura el contexto
## (nada de momento)

## Restaura el modo usuario y vuelve al programa
## (falta algo)
rfe
jr $k0 ¿cuánto vale $k0?

#####
## CÓDIGO DE INICIO ##
#####

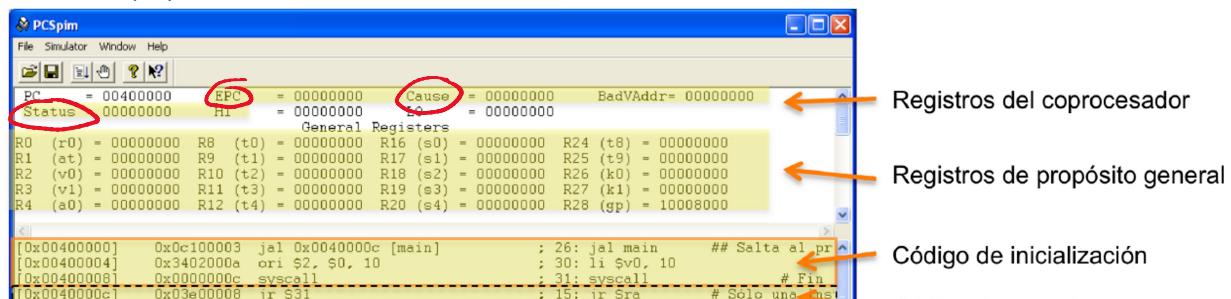
.text
.globl __start
_start:
## Código de inicio
## (nada de momento)

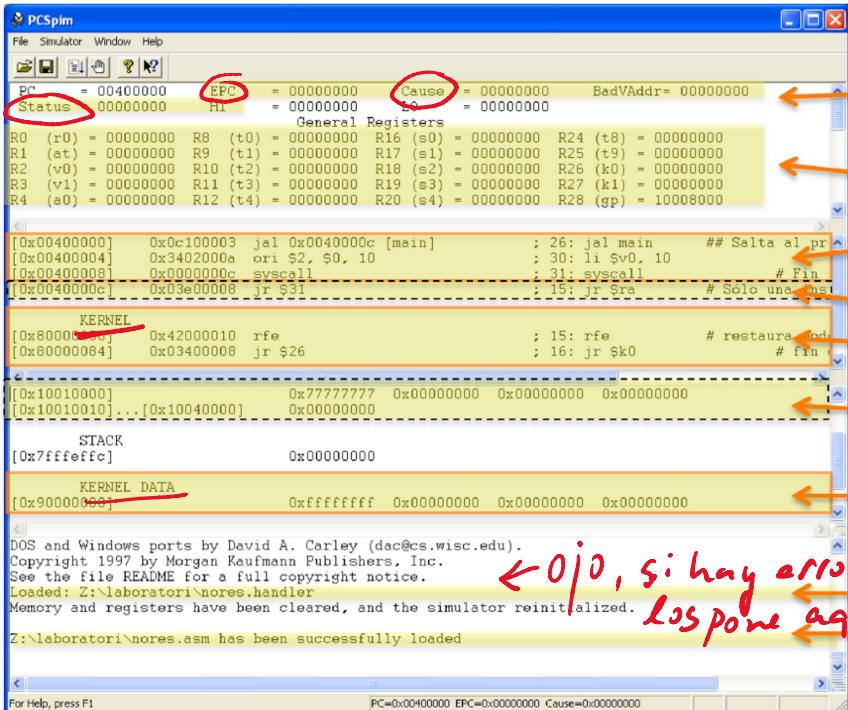
## Salto al programa de usuario
jal main → sale al programa .asm
## Código de terminación
li $v0, 10
syscall # Función exit
```

Annotations on the code:

- dirección de las excepciones** (exception address) is circled around the **.ktext 0x80000080** line.
- Sólo se ejecuta si se produce una excepción** (executes only if an exception occurs) is written next to the **rfe** instruction.
- Se ejecuta siempre al inicio** (executes always at startup) is written next to the **jal main** instruction.

Visión en el pcspim-ES





Registros del coprocesador

Registros de propósito general

Código de inicialización

Código de usuario

Código del manejador

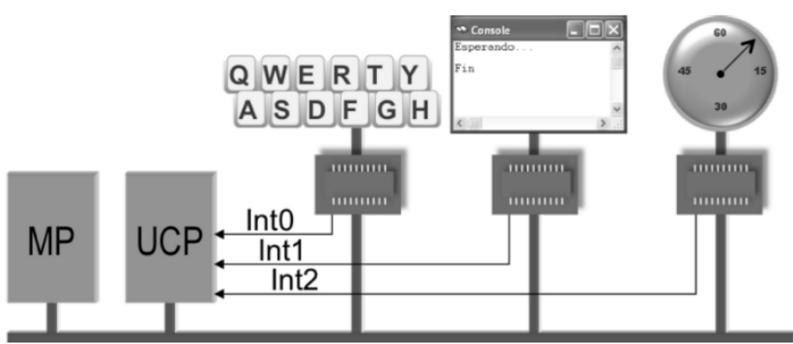
Variables del proceso usuario

Variables del manejador

Mensaje: archivo .handler cargado

Mensaje: archivo lasm cargado

Conexión del teclado, consola y reloj a las líneas de interrupción del MIPS



TAREA 2 - Preparación de bucle.asm

Ajustar el valor de la línea:

```
bucleE: li $t1, 1000000
```

para que el programa tarde de 5 a 10 segundos en terminar

TAREA 3 - Inicialización del sistema

Primera aproximación: solo usamos el teclado.

Copiamos nada.handler --> teclado.handler

Editamos teclado.handler

```
## Código de inicio
## (nada de momento) aquí corribuiremos el código
```

(usamos apéndice 3 para programar el teclado)

Interfaz del teclado (DB = 0xFFFF0000, Int0)

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

Habilitar las interrupciones en el registro de estado/órdenes del teclado

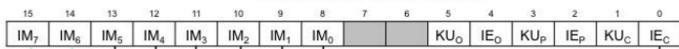
(usamos el apéndice 2 para escribir en el registro de estado (\$12) del coprocesador 0

Registros

Número	Nombre	Descripción
\$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.

Registros de estado y de causa

REGISTRO DE ESTADO



Escribir el valor adecuado en la palabra de estado del coprocesador para que la línea de interrupción Int0* quede desenmascarada, las interrupciones habilitadas y el modo usuario seleccionado. Para ello serán útiles las instrucciones ~~mtc0~~ y mtc0.

*Como es una inicialización
no es necesario leer el registro de
estado, si lo haces nos llevas todos
los bits.*

TAREA 4 - Comprensión de la estructura del manejador

Vamos a ir construyendo un manejador correcto por aproximaciones sucesivas.

Versión 1

En primer lugar (cuestión 3) vamos a resolver el problema de la dirección de retorno, para ello cogeremos del registro EPC (\$14) del coprocesador 0 la dirección de retorno y la pondremos en \$K0. Esto lo haremos justo antes de la instrucción rfe. Nota: en esta práctica vamos a suponer que es imposible que dentro del manejador se produzca otra excepción y por eso podemos leer el registro EPC al final (asumimos que no va a cambiar), justo antes de volver, en teoría lo leemos al principio y lo salvamos con el contexto mínimo.

Ahora vamos a añadir una rutina para el tratamiento del teclado, la propuesta es:

```
li $v0,11  
li $a0, '*'  
syscall
```

Nos indica la práctica que es incorrecta, la pista del porqué es la pregunta de la cuestión 4:

Cuestión 4. ¿Por qué se escriben tantos asteriscos al pulsar una tecla?

¿Qué tiene que hacer cualquier tratamiento de una interrupción con el dispositivo antes de acabar?

Nota adicional a efectos informativos: hacemos una pequeña trampa en esta rutina que podemos hacer porque usamos un simulador, que es usar una instrucción syscall dentro del manejador. Recordad que la instrucción syscall genera una excepción 8 y por tanto estaríamos en contra de la regla de no generar una excepción anidada. En este caso el syscall lo gestiona el simulador y no genera tal excepción y por eso lo podemos hacer, pero en un entorno real (o haciendo que el simulador genere la excepción 8 como haremos en las próximas prácticas) no es posible.

Continuad vosotros con las cuestiones 5 y 6

Versión 2

En la versión 1 hemos llegado a un manejador que tiene básicamente la parte del tratamiento de la interrupción de teclado y retorna correctamente, pero no salva el contexto mínimo por lo que cualquier registro que modifique y que use el programa de usuario provocará que el mal funcionamiento de dicho programa.

En la versión 2 añadiremos el salvar el contexto mínimo y recuperarlo antes de volver.

El objetivo final es llegar al manejador completo estudiado en el tema 8.

De momento en esta versión no necesitamos identificar la causa porque solo atendemos interrupciones (excepción 0) y de éstas la interrupción 0 (el teclado)

A la hora de decidir cuál es el contexto mínimo nos vamos a basar en el código de la rutina de tratamiento de nuestro manejador y vemos que usa los registros \$t0, \$a0 y \$v0 (que no son exactamente los que se usan como ejemplo en el tema 8) y además añadiremos el \$at. Éste lo añadimos porque es el que usa el ensamblador como registro intermedio en la expansión de las pseudo-instrucciones con operandos de más de 16 bits.

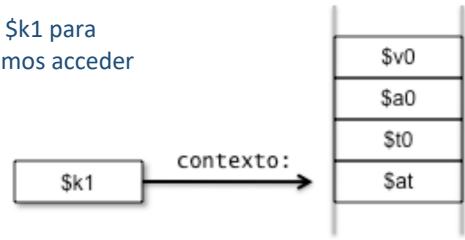
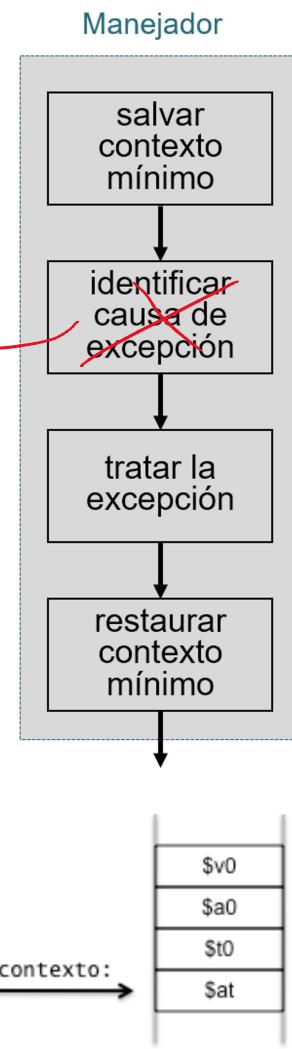
Como ya he comentado anteriormente, a diferencia del manejador de clase no vamos a incluir en el contexto la dirección de retorno, ya que vamos a suponer que el valor del registro EPC no se modificará.

Hay que tener en cuenta que para que el ensamblador (que incluye el simulador) nos permita usar el registro \$at tenemos que usar la directiva `.set noat` antes del uso y la directiva `.set at` después del uso.

Como indica en el boletín, y como se hace en el tema 8, usaremos el registro \$k1 para acceder al contexto, una vez tengamos \$k1 correctamente inicializado podremos acceder a las posiciones de memoria con 0(\$k1), 4(\$k1), etc.

El valor del registro \$k1 no se modificará en ningún punto del manejador.

Continuad con la cuestión 7.



Tarea 5. Tratamiento de varias interrupciones

En este apartado vamos a trabajar con el reloj (ver apéndice 3), que os presento a continuación:

El reloj

Interfaz del reloj (DB = 0xFFFF0010, Int2)

Nombre	Dirección	Acceso	Estructura
Estado/órdenes	DB	LE	R,E

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

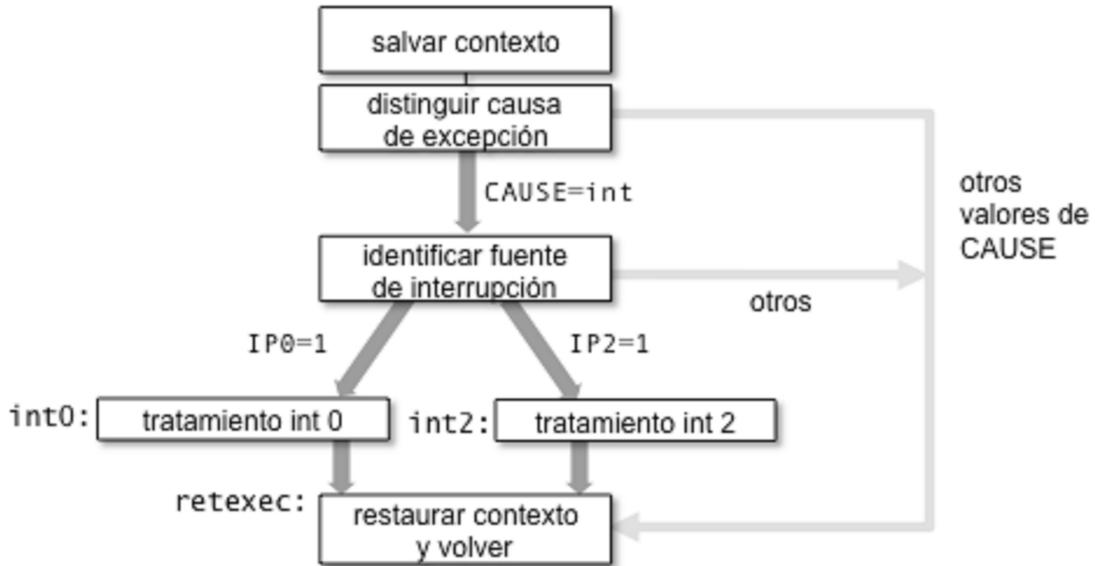
- R (bit 1, lectura/escritura). Indicador de dispositivo preparado: R = 1 a cada segundo.
- Cancelación (R = 0): se escribirá en bit R un 0.
- E: (bit 0, lectura/escritura). Habilitación de la interrupción (mientras E = 1, el valor R = 1 activa la línea de interrupción del dispositivo)

Este interfaz activa su bit de preparado cada segundo y si tiene las interrupciones habilitadas generará una interrupción cada segundo. Ojo, si no borramos su bit R las interrupciones se regenerarán de forma continua como con cualquier dispositivo en el que no se cancele la interrupción.

Continuad con las cuestiones 8 y 9.

Versión 3

En esta versión identificaremos la causa de la interrupción y pondremos un tratamiento específico para cada interrupción



Haced los tres primeros apartados de la cuestión 10.

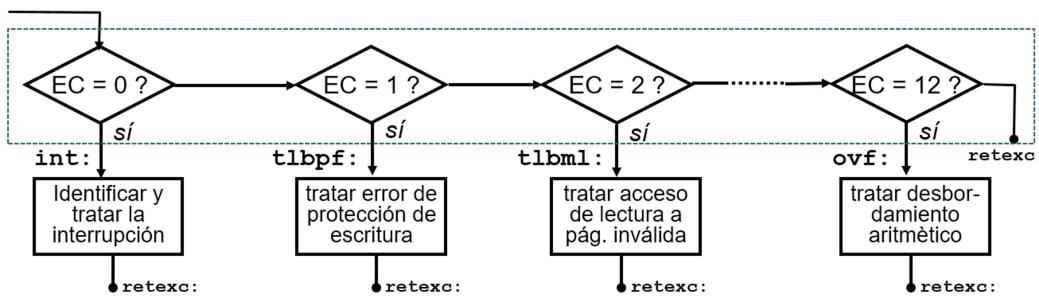
En el apartado 4, hay una errata:

Añade las instrucciones que lean y aíslen el código de causa de excepción del coprocesador en la sección rotulada “## Identifica y trata excepciones” (la tabla de códigos aparece en el archivo [“Apéndice 1. Llamadas al sistema.pdf”](#)). Si el código es de interrupción, seguirá el análisis; en otro caso ha de saltar a la etiqueta que vale para finalizar.

Se debería referir al Apéndice2. El coprocesador de excepciones

Para hacer los apartados 4 y 5 de la cuestión 10 os aconsejo que uséis como punto de partida las transparencias del tema 8

Identificar la causa de la excepción



CR

Código de excepción

...			EC 3	EC 2	EC 1	EC 0		
8	7	6	5	4	3	2	1	0

```

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, tlbfp    #     compara y salta si igual
li $t1, 8              # Código 2*4
beq $t0, $t1, tblml   #     compara y salta si igual
...
li $t1, 0x30            # Código 12*4
beq $t0, $t1, ovf      #     compara y salta si igual
b reexec
  
```

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*
```

Y haced caso de la advertencia que hay al final:

- En todo momento, ten cuidado con los registros que se utilizan en el manejador. En el paso 5 se escribió el código que guarda y restaura los registros \$at, \$t0, \$v0 y \$a0.

Y con esto termina la práctica.

Sugerencia extra para probar que la interrupción de reloj se produce cada segundo, cambiad la rutina de tratamiento de la interrupción 2 para que escriba el carácter '+', de esa forma tendremos un '+' en consola cada segundo y un '*' cada vez que pulsemos una tecla.