

ESCUELA DE INGENIERÍA INFORMÁTICA

**Grado en Ingeniería Informática**



## **PERIFÉRICOS E INTERFACES**

PRÁCTICAS DE LABORATORIO 17/18

### **Módulo 1 - Práctica 1**

***MIPS-32: Excepciones e interrupciones***

***“Práctica básica y mejorada”***

***Ana Isabel Santana Medina***

*Grupo: 1.18.43*

*Periodo de realización: 1ª – 5ª semana*

## Índice

- Objetivos de la práctica y competencias logradas
- Tareas y práctica
  - a. Tareas
  - b. Código práctica final
  - c. Modificación exception.s
- Mejora de la práctica final
  - a. Breve descripción
  - b. Código
- Conclusiones
- Comentarios finales
- Recursos utilizados y bibliografía

# 1. Competencias y objetivos básicos de la práctica

Los siguientes datos e información detallada corresponden a la primera práctica base de la asignatura Periféricos e Interfaces y la mejora de la misma.

Una breve descripción de esta práctica es que se centra en el uso de los mecanismos de sincronización básicos entre CPU y dispositivos de entrada/salida. Para ello, hemos hecho uso del simulador PCSpim versión 8 que simula la arquitectura MIPS-32.

El simulador PCSpim simula varios dispositivos hardware entre los que podemos encontrar: un timer, un dispositivo de entrada o receptor (teclado) y un dispositivo de salida o transmisor (pantalla). El comportamiento y uso de estos dispositivos se define a través de los registros de sus interfaces. Es tarea nuestra tarea como usuarios desarrollar el software de bajo nivel para programar el modo de funcionamiento de estos dispositivos y realizar las transferencias de datos haciendo uso de diferentes técnicas de sincronización entre dispositivos de entrada/salida y CPU.

Hemos hecho uso de la sincronización por consulta de estado e interrupciones y transferencias de datos por programa. Las competencias de esta práctica se han basado en lo siguiente:

1. Entender los diversos aspectos software y hardware involucrados en los métodos de sincronización.
2. Capacidad para diseñar e implementar el software necesario para la gestión de interrupciones y excepciones en un sistema con arquitectura de procesador tipo MIPS-32 simulado por el PCSpim.
3. Capacidad para hacer uso de los periféricos simulados en el PCSpim para el intercambio de información entre el sistema y su entorno utilizando como métodos de sincronización la consulta de estado y las

interrupciones.

4. Capacidad para aprender y aplicar nuevos conceptos de forma autónoma e interdisciplinar.
5. Capacidad para emplear la creatividad en la resolución de los problemas.
6. Capacidad para trabajar en equipo y colaborar eficazmente con otras personas.

Para alcanzar estas competencias se planteó el seguimiento de los siguientes objetivos:

1. Conocer y usar el simulador PCSpim para la ejecución y depuración de programas sencillos realizados en ensamblador del procesador MIPS.
2. Conocer y entender los aspectos básicos de funcionamiento de los dispositivos hardware simulados en el PCSpim.
3. Realizar programas sencillos que impliquen operaciones de entrada/salida con los periféricos simulados en el PCSpim utilizando la consulta de estado como método de sincronización.
4. Conocer y entender la gestión de interrupciones en la arquitectura del MIPS-32 a través del estudio del manejador de excepciones e interrupciones “exceptions.s” del PCSpim.
5. Diseñar y programar una aplicación sencilla que implique operaciones de entrada/salida con los periféricos simulados en el PCSpim, combinando la consulta de estado y las interrupciones como métodos de sincronización y las transferencias de datos por programa.

## 2. Realización de tareas y práctica final

Para la realización de la práctica básica he desarrollado de forma modular las siguientes actividades que se planteaban que se fueron encadenando para dar lugar a la aplicación final.

La estructura principal de las tareas la he enfocado en el en mismo sentido, todas poseen la misma estructura del Apéndice B que se nos proporcionó en el enunciado de la práctica. Esto se refiere a algo así:

```

# Definición de los segmentos de datos

.data 0xFFFF0000

#Registros de los dispositivos de entrada/salida

tControl:      .space 4 #Dirección del registro de control del
                  teclado (0xFFFF0000)
tData:         .space 4 #Dirección del registro de datos del
                  teclado (0xFFFF0004)
pControl:      .space 4 #Dirección del registro de control de
                  pantalla (0xFFFF0008)
pData:         .space 4 #Dirección del registro de datos de
                  pantalla (0xFFFF000C)

.data 0x10000000

# Frase (o carácter) a imprimir por el programa principal, espacios para almacenar
# frases o caracteres, variables adicionales
...
# Segmento de texto

.text 0x00400000
.globl main

# Programa principal

```

Esta estructura no la incluiré en cada una de las tareas, solo la incluiré en la práctica final y en la mejora de la misma. Las tareas desarrolladas han sido las siguientes:

**Tarea 1.** Realizar una subrutina, de nombre ***PrintCharacter***, para imprimir un carácter al dispositivo de salida (pantalla) utilizando la sincronización por consulta de estado. El carácter a imprimir se le pasa en el registro \$a0.

```

main:
    jal PrintCharacter
    j main

PrintCharacter:

    la $s3, caracter          #Cargamos la dirección de carácter
    lb $a0, 0($s3)            #Cargamos el byte que se corresponde con el
                                carácter

    la $s1, pControl          #Cargamos la dirección del registro de estado de
                                pantalla
    lb $s2, 0($s1)            #Cargamos el byte que se encuentra en el registro
                                de estado de la pantalla

    beqz $s2, PrintCharacter   #Si ese byte es igual a cero retornamos a

```



```

    beqz $s2, PrintCharacter    #Si ese byte es igual a cero retornamos a
                                PrintCharacter pues la pantalla no está lista

    sb $a0, pData              #Guardamos el byte en el registro de datos de la
                                pantalla para, de esta forma, imprimir dicho
                                carácter

    jr $ra

```

**Tarea 4.** Realizar un programa que, de forma indefinida, imprima una frase en la pantalla. Establezca una pausa entre la impresión de cada carácter de la frase (por ejemplo un carácter por segundo) y haga uso de la subrutina *PrintCharacter* desarrollada anteriormente. Observe si hay pérdida de caracteres.

```

main:
    jal PrintCharacter
    j main

PrintCharacter:
    la $s1, frase              #Cargamos la dirección de la frase

checkPantalla:
    la $s2, pControl            #Con esta rutina comprobamos si la pantalla está
                                lista
    lb $s3, 0($s2)

    beqz $s3, checkPantalla

printPantalla:
    lb $s4, 0($s1)              #Si está lista, almacenamos el byte de
                                la frase
    sb $s4, pData               #Y lo guardamos en pData
    beqz $s4, PrintCharacter     #Comprobamos que no es el final de la
                                frase

    addi $s1, $s1, 1            #Sumamos 1 para avanzar al siguiente carácter de
                                la frase
    jal Delay                    #Saltamos al retardo
    j printPantalla              #Cuando volvemos del retardo saltamos a imprimir
                                por pantalla para seguir imprimiendo la frase

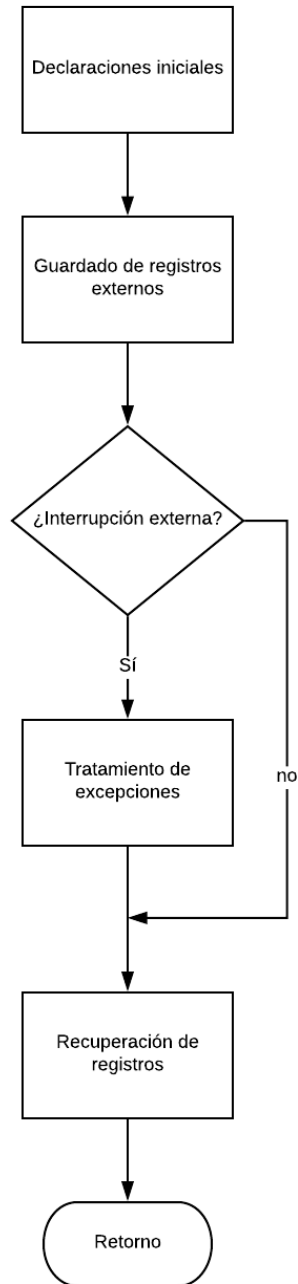
Delay:
    li $s5, 50000               #Cargamos un valor inmediato que será el tiempo
                                que tarde en mostrar el carácter

keep:
    addi $s5, $s5, -1           #Decrementamos el tiempo
    bnez $s5, keep              #Hasta que no sea cero no retornamos

    jr $ra

```

**Tarea 5.** Analizar y estudiar el código del manejador de excepciones suministrado con PCSpim (exceptions.s) y mostrado en el Apéndice A. Realizar un diagrama de flujo del manejador. (Sin modificaciones)



**Tarea 6.** Realizar la programación que estime oportuna para atender al teclado por interrupciones. Habilitar el nivel de interrupción pertinente en el procesador y habilitar generación de interrupciones en el teclado. Cada vez se pulse una tecla se generará una interrupción que será atendida por una rutina de servicio cuyo fin será leer el código de la tecla pulsada, actualizar un contador de pulsaciones que indique el número de pulsaciones realizadas hasta ese momento, enviar a pantalla el mensaje " *[Pulsación(número) = tecla pulsada]* " y retornar al programa principal quedando todo listo para una nueva pulsación.



**main:**

```
jal KbdIntrEnable      #Llamamos a la subrutina que activa las interrupciones
                        en el teclado
```

# Dejamos el programa en bucle continuo esperando interrupción por el teclado

bucle:

```
j bucle
```

**PrintCharacter:**

```
la $s0, pControl        #Comprobamos el estado de la pantalla
```

```
li $s1, 0x1
```

```
beq $s0, $s1, PrintCharacter
```

```
la $s3, pData           #Cargamos la dirección del registro de datos de
                        la pantalla
```

```
sb $t1, 0($s3)          #Almacenamos la tecla pulsada
```

```
jr $ra
```

**KbdIntrEnable:**

```
la $s0 tControl         #Cargamos la direccion del registro de control del
                        teclado
```

```
li $s1, 0x2             #Cargamos un 0010 en hexadecimal
```

#Almacenamos en el registro de control los 4 bits anteriores en las 4 posiciones menos significativas de manera que queda un 1 en el segundo bit y se activan las interrupciones

```
sw $s1, 0($s0)
```

```
mfc0 $t1, $12           #Movemos desde el coprocesador 0 hasta t1, el contenido
                        del registro status
```

#Hacemos un or del contenido de status con 1000 0000 0001 para poner a 1 el bit IE(Activa interrupciones) y para poner un 1 en el campo IM3 que activa la máscara de interrupciones del teclado en el procesador

```
ori $t1, $t1, 0x801
```

```
mtc0 $t1, $12           #Movemos al procesador los cambios que hemos realizado
                        en el registro status
```

```
jr $ra
```

**CaseIntr:** #Detecta origen de la interrupción y llama a la rutina de servicio

```
sw $ra, pila            #Almacenamos en la variable pila la dirección de retorno
```

```
mfc0 $t1, $13           #Movemos desde el coprocesador 0 hasta t1, el
                        contenido del registro cause
```

```
andi $a0, $t1, 0x800
```

```
bnez $a0, KbdIntr       #Si a0 != 0, la interrupción proviene del teclado
                        y saltamos a la rutina de servicio del teclado
```

```

        lw $ra, pila                #Recuperamos de la variable pila la dirección de
                                    retorno

        jr $ra

KbdIntr:                        #Rutina de servicio de interrupción del teclado

        sw $ra, pila                #Almacenamos la dirección de retorno

        li $v0, 4

        la $a0, pulsacion           #Impresión "[Pulsación("

        syscall

        lw $s1, numero_pulsacion    #Entero que representa el número de veces que se
                                    ha generado una interrupción por teclado

        addi $s1, $s1, 1

        sw $s1, numero_pulsacion

        li $v0, 1                   #Imprimir entero

        lw $a0, numero_pulsacion

        syscall

        li $v0, 4

        la $a0, parentesis          #Impresión ") = "

        syscall

        lb $t1, tData               #Cargamos en t1 el carácter pulsado por teclado
                                    que será mandado a la rutina PrintCharacter

        jal PrintCharacter

        li $v0, 4

        la $a0, corchete             #Impresión "]"

        syscall

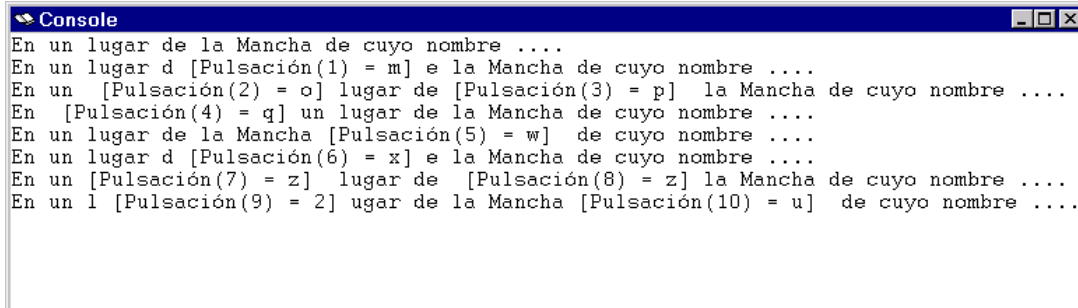
        lw $ra, pila                #Recuperamos la dirección de retorno

        jr $ra

```

**Tarea 7.** Combinar la impresión continua de la frase con el funcionamiento del teclado. Para comprobar el funcionamiento correcto del atendimento del teclado y del sistema en general, se desarrollará un programa principal consistente en un bucle indefinido que imprima en pantalla una frase carácter a carácter. Si se pulsa

una tecla debe interrumpirse la ejecución del programa principal y atender al teclado a través de su rutina de servicio. La salida será similar a la que figura en la siguiente ventana:



```
Console
En un lugar de la Mancha de cuyo nombre ....
En un lugar d [Pulsación(1) = m] e la Mancha de cuyo nombre ....
En un [Pulsación(2) = o] lugar de [Pulsación(3) = p] la Mancha de cuyo nombre ....
En [Pulsación(4) = q] un lugar de la Mancha de cuyo nombre ....
En un lugar de la Mancha [Pulsación(5) = w] de cuyo nombre ....
En un lugar d [Pulsación(6) = x] e la Mancha de cuyo nombre ....
En un [Pulsación(7) = z] lugar de [Pulsación(8) = z] la Mancha de cuyo nombre ....
En un l [Pulsación(9) = 2] ugar de la Mancha [Pulsación(10) = u] de cuyo nombre ....
```

**main:**

```
jal TInterEnable      # Saltamos a la activación de las interrupciones
```

**CargaFrase:**

```
la $s3, frase # Cargamos la frase para luego imprimirla
```

**ImprimeFrase:**

```
lb $s2, 0($s3)      # Cargamos un byte de la frase
```

```
beqz $s2, CargaFrase # Comprobamos que no es el final de la cadena
```

```
jal PrintCharacter   # Imprimimos el carácter
```

```
addi $s3, $s3, 1     # Avanzamos en la cadena
```

```
jal Delay            # Saltamos al retardo de la impresión
```

```
j ImprimeFrase       #Volvemos a imprimir el siguiente carácter
```

**PrintCharacter:**

```
la $t0, pControl     # Comprobamos el estado de la pantalla
```

```
lb $t1, 0($t0)
```

```
beqz $t1, PrintCharacter
```

```
sw $s2, pData        # Almacenamos el byte correspondiente en el registro
```

```
                    # de datos de la pantalla
```

```
j $ra
```

**Delay:** # Rutina de retardo

```
li $s5, 50000
```

Keep:

```

        addi $s5, $s5, -1
        bnez $s5, Keep

jr $ra

TInterEnable:      # Rutina de interrupción del teclado

        la $s0, tControl      #Cargamos la direccion del registro de control del
                                teclado
        li $s1, 0x2           #Cargamos un 0010 en hexadecimal

        sw $s1, 0($s0)

        mfc0 $t1, $12         #Movemos desde el coprocesador 0 hasta t1, el
                                contenido del registro status

        ori $t1, $t1, 0x801
        mtc0 $t1, $12         #Movemos al procesador los cambios que hemos
                                realizado

        jr $ra

CaseIntr:        # Rutina que identifica de donde procede la interrupción

        sw $ra, pila          # Almacenamos en la variable pila la dirección de retorno

        mfc0 $t1, $13         # Movemos desde el coprocesador 0 hasta t1, el contenido
                                del registro cause

        andi $s0, $t1, 0x800   # Si a0 != 0, la interrupción proviene del teclado
                                y saltamos a la rutina de servicio del teclado
        bnez $s0, TInter

        lw $ra, pila          # Recuperamos la dirección de retorno
        jr $ra

TInter:          # Rutina de servicio de interrupción del teclado

        sw $ra, pila          # Almacenamos en la variable pila la dirección de retorno

        li $v0, 4
        la $a0, pulsacion     # Impresión "[Pulsación("
        syscall

        lw $s1, num_pulsacion # Entero que representa el número de veces que se ha
                                generado una interrupción por teclado

        addi $s1, $s1, 1
        sw $s1, num_pulsacion

        li $v0, 1

```

```

lw $a0, num_pulsacion      # Imprimir entero
syscall

li $v0, 4
la $a0, parentesis        # Impresión ")" = "
syscall

lb $s2, tData              # Cargamos el carácter pulsado por teclado que será
mandado a la rutina PrintCharacter
jal PrintCharacter

li $v0, 4
la $a0, corchete           # Impresión "]" "
syscall

lw $ra, pila               # Recuperamos la dirección de retorno
jr $ra

```

## Práctica final

**Tarea 8. (Práctica final)** Implementar un reloj haciendo uso de los registros “count” y “compare” del procesador. Cuando la cuenta en "count" alcance el valor prefijado en el registro "compare" se generará una interrupción de timer (deberá ser una por segundo). La subrutina de servicio del timer será la encargada de actualizar la hora interna implementada a través de contadores almacenados en memoria (hora-minuto-segundo). Antes de salir la rutina de servicio mostrará en pantalla (en una nueva línea) la hora según el siguiente formato:

<nueva línea> “*Hora local → 13:23:55*” <nueva línea>.

La puesta en hora del reloj se activará al pulsar en el teclado CTRL+R cuyo código será detectado por la rutina de servicio del teclado para saltar o llamar a la subrutina de puesta en hora. Esta subrutina de puesta en hora pedirá al usuario el valor de cada uno de los campos correspondientes a hora-min-seg. Tiene completa libertad para diseñar los diversos aspectos del interfaz de usuario de esta tarea: se valorará positivamente la calidad de dicho interfaz.

```
Console
En un lugar de la Mancha de cuyo nombre ....
En un lugar d [Pulsación(1) = m] e la Mancha de cuyo nombre ....
En un [Pulsación(2) = o] lugar de [Pulsación(3) = p] la Mancha de cuyo nombre ....
En [Pulsación(4) = q] un lugar de la Mancha de cuyo nombre ....
En un lugar de la Mancha [Pulsación(5) = w] de cuyo nombre ....
En un lugar d [Pulsación(6) = x] e la Mancha de cuyo nombre ....
En un [Pulsación(7) = z] lugar de [Pulsación(8) = z] la Mancha de cuyo nombre ....
En un l [Pulsación(9) = 2] ugar de la Mancha [Pulsación(10) = u] de cuyo nombre ....
Hora local: 13:54:00
```

##

## Nombre: Ana Isabel Santana Medina

##

## Grupo: 43

##

## Fecha: 04/02/18

#####

#####

# Definición de los segmentos de datos

.data 0xFFFF0000

# Registros de los dispositivos de entrada/salida

kControl: .space 4 # Estado del teclado

kData: .space 4 # Carácter pulsado

pControl: .space 4 # Estado de la pantalla

pData: .space 4 # Dato en pantalla - Carácter a imprimir

# Registros de los dispositivos de entrada/salida

.data 0x10000000

# Frase a imprimir por el programa principal

frase: .asciiz "En un lugar de la Mancha cuyo nombre ....\n"

# Definición del mensaje: "[Pulsación (n) = tecla]"

pulsacion: .asciiz " [Pulsacion("

pulsacionNum: .word 0

parentesis: .asciiz ") = "

letra: .space 1

```

corchete:                .asciiiz "]" "

# Implementación del reloj

horas:                    .word 0

minutos:                  .word 0

segundos:                 .word 0

dosPuntos:                .asciiiz ":"

saltoLinea:               .asciiiz "\n"


horaLocal:                .asciiiz "\nHora Local: "

introducirHora:           .asciiiz "\nIntroducir Hora: "

introducirMinutos:        .asciiiz "\nIntroducir Minutos: "

introducirSegundos:       .asciiiz "\nIntroducir Segundos: "


# Definición de los mensajes de errores

errorHora:                .asciiiz "\n\tEl valor es incorrecto. Debe ser un
numero entre 0 y 23\n"

errorMinutosSegundos:     .asciiiz "\n\tEl valor es incorrecto. Debe ser un
numero entre 0 y 59\n"


# Referencias auxiliares

lastAddress:              .word 0          # Variable para guardar la
                                         dirección de retorno


# Segmento de texto

.text 0x400000

.globl main


# Inicio programa principal

main:

    jal Kbd_Timer_IntrEnable

    li $t8, 10              # Si el timer pasa de 9 se pone este a 10

    la $t7, 0x12            # Código de Ctrl + R (Reloj)


    # Bucle que imprime la frase en pantalla de forma indefinida


CargaFrase:

    la $s4, frase           # Cargamos la dirección de la frase a imprimir

```

**ImprimeFrase:**

```
        lb $t1, 0($s4)                # Se lee un byte de la cadena

        beqz $t1, CargaFrase          # Comprobamos que no es el final de la
                                      frase

        jal PrintCharacter             # Saltamos a imprimir el caracter

        addi $s4, $s4, 1              # Avanzamos en la frase

        jal Delay                     # Saltamos al retardo

        j ImprimeFrase                # Volvemos a ImprimeFrase
```

#####

**# PrintCharacter(): Rutina que imprime un dato en pantalla**

#####

**PrintCharacter:**

```
        la $s1, pControl              # Cargamos el registro control de la pantalla

        lb $s2, 0($s1)

        beqz $s2, PrintCharacter      # Comprobación para ver si la pantalla está
                                      disponible

        sw $t1, pData                 # Si esta listo guardamos el contenido

        jr $ra
```

#####

**# Delay(): Rutina para realizar un retardo**

#####

**Delay:**

```
        li $s5, 50000                # Guardamos el tiempo de retardo
```

**LoopDelay:**

```
        addi $s5, $s5, -1             # Decrementamos hasta llegar a 0

        bnez $s5, LoopDelay           # Bucle que termina cuando el contador es 0

        jr $ra
```

#####

**# Kbd\_Timer\_IntrEnable(): Rutina que habilita las interrupciones de teclado y timer**

#####





```

andi $v0, $t1, 0x800          # Comprobamos si es una interrupción del teclado
bnez $v0, KbdIntr

```

```

andi $v0, $t1, 0x8000         # Comprobamos si es una interrupción del timer
bnez $v0, TimerIntr

```

```

lw $ra, lastAddress           # Cargamos la dirección de retorno
jr $ra

```

```

#####

```

```

# KbdIntr(): Rutina de servicio de interrupciones del teclado

```

```

#####

```

```

KbdIntr:

```

```

sw $ra, lastAddress           # Guardamos la dirección de retorno
lb $t1, kData                  # Traemos la tecla pulsada

```

```

beq $t1, $t7, CtrlR           # Comprobamos si es una interrupción por CTRL + R

```

```

li $v0, 4

```

```

la $a0, pulsacion              # Primera parte "[Pulsacion("
syscall

```

```

lw $s1, pulsacionNum

```

```

addi $s1, $s1, 1

```

```

sw $s1, pulsacionNum           # Contador

```

```

lw $a0, pulsacionNum

```

```

li $v0, 1

```

```

syscall

```

```

li $v0, 4

```

```

    la $a0, parenthesis          # Segunda parte ")" = "
    syscall

    lb $t1, kData                # Carácter
    jal PrintCharacter

    li $v0, 4

    la $a0, corchete             # Parte final "]"
    syscall

    lw $ra, lastAddress          # Cargamos la dirección de retorno
    jr $ra

#####

# TimerIntr(): Rutina de servicio de interrupciones del timer

#####

TimerIntr:

    sw $ra, lastAddress          # Guardamos la dirección de retorno

    ImprimeHoraLocal:           # Subrutina que imprime la hora local

        li $v0 4

        la $a0, saltoLinea      # Salto de línea
        syscall

        li $v0 4

        la $a0, horaLocal        # Imprimimos "Hora Local: "
        syscall

        li $v0, 1                # Imprimimos el entero

        la $t2, horas            # Lo guardamos en la variable para horas

        lw $s3, 0($t2)

```

```
bge $s3, $t8, ActualizaHoras # Comprobamos si el timer pasa de 10
```

```
li $a0, 0 # Si no, pone un 0 antes del número
```

```
syscall
```

#### **ActualizaHoras:**

```
lw $a0, 0($t2) # Actualizamos el valor de las horas
```

```
syscall
```

```
li $v0, 4
```

```
la $a0, dosPuntos # Introducimos dos puntos para la separación
```

```
syscall
```

```
li $v0, 1 # Imprimimos el entero
```

```
la $t2, minutos # Lo guardamos en la variable para minutos
```

```
lw $s3, 0($t2)
```

```
bge $s3, $t8, ActualizaMinutos # Si es mayor o igual que 10,  
                                salta a los min
```

```
li $a0, 0 # Si no pone un 0 antes del dígito único
```

```
syscall
```

#### **ActualizaMinutos:**

```
lw $a0, 0($t2) # Actualizamos el valor de los minutos
```

```
syscall
```

```
li $v0, 4
```

```
la $a0, dosPuntos # Introducimos dos puntos para la separación
```

```
syscall
```

```
li $v0, 1 # Imprimimos el entero
```

```
la $t2, segundos # Lo guardamos en la variable para segundos
```

```
lw $s3, 0($t2)
```

```
bge $s3, $t8, ActualizaSegundos    # Si es mayor o igual que 10,  
                                     salta a los segundos
```

```
li $a0, 0    # Si no, pone un 0 antes del número
```

```
syscall
```

#### **ActualizaSegundos:**

```
lw $a0, 0($t2)    # Actualizamos el valor de los segundos
```

```
syscall
```

```
li $v0, 4
```

```
la $a0, saltoLinea    # Salto de línea
```

```
syscall
```

```
addi $s3, $s3, 5    # Suma 5 segundos
```

```
bgt $s3, 59, IncrementarMinutos    # Si es mayor de 59 salta a  
                                     incrementar los minutos
```

```
sw $s3, 0($t2)
```

```
j FinalTimeIntr    # Salta al final del reloj
```

#### **IncrementarMinutos:**

```
li $s3, 0    # Pone a 0 los segundos
```

```
sw $s3, 0($t2)
```

```
la $t2, minutos
```

```
lw $s3, 0($t2)
```

```
addi $s3, $s3, 1    # Suma 1 minuto
```

```
bgt $s3, 59, IncrementarHoras    # Si es mayor que 59 minutos  
                                     reinicias
```

```
sw $s3, 0($t2)
```

```

        j FinalTimeIntr          # Salta al final del reloj

IncrementarHoras:

        li $s3, 0                # Pone a 0 los minutos

        sw $s3, 0($t2)

        la $t2, horas

        lw $s3, 0($t2)

        addi $s3, $s3, 1          # Suma 1 hora

        bgt $s3, 23, Clock_Reset # Si es mayor que 24 horas reinicias

        sw $s3, 0($t2)

        j FinalTimeIntr          # Salta al final del reloj

Clock_Reset:                      # Reseteamos el reloj

        la $t2, horas

        sw $0, 0($t2)            # Pone a 0 las horas. Los minutos y
                                # segundos ya están a 0

        j FinalTimeIntr          # Salta al final del reloj

FinalTimeIntr:

        mtc0 $0, $9              # Inicializamos el registro "count" a 0 y los
                                # cargamos en el coprocesador0

        lw $ra, lastAddress       # Cargamos la dirección de retorno

        jr $ra

#####

# CtrlR(): Rutina que se ejecuta al presionar Ctrl + R, y solicita la hora

#####

CtrlR:

        li $v0 4

```

```

        la $a0, saltoLinea           # Salto de línea

        syscall

PedirHoras:

        li $v0, 4

        la $a0, introducirHora       # Imprimimos "Introducir Hora: "

        syscall

        li $v0, 5                    # Solicitamos un valor

        syscall

        addi $t0, $0, 0              # Código para el error

        blt $v0, 0, Error_CtrlR      # Comprobamos que la hora no sea < 0

        bge $v0, 24, Error_CtrlR     # Comprobamos que la hora no sea >= 24

        la $t2, horas

        sw $v0, 0($t2)

PedirMinutos:

        li $v0, 4

        la $a0, introducirMinutos    # Imprimimos "Introducir Minutos: "

        syscall

        li $v0, 5                    # Solicitamos un valor

        syscall

        addi $t0, $0, 1              # Código para el error

        blt $v0, 0, Error_CtrlR      # Comprobamos que los minutos no
                                     sean < 60

        bge $v0, 60, Error_CtrlR     # Comprobamos que los minutos no
                                     sean >= 60

        la $t2, minutos

        sw $v0, 0($t2)

```

#### **PedirSegundos:**

```
li $v0, 4

la $a0, introducirSegundos    # Imprimimos "Introducir Segundos: "

syscall

li $v0, 5                      # Solicitamos un valor

syscall

addi $t0, $0, 2                # Código para el error

blt $v0, 0, Error_CtrlR        # Comprobamos que los segundos no
                                sean < 60

bge $v0, 60, Error_CtrlR       # Comprobamos que los segundos no
                                sean >= 60

la $t2, segundos

sw $v0, 0($t2)

li $v0 4

la $a0, saltoLinea            # Salto de línea

syscall

lw $ra lastAddress             # Cargamos la dirección de retorno

jr $ra
```

#####

**# Error\_CtrlR(): Rutina que se ejecuta al ocurrir un error en la hora introducida**

#####

#### **Error\_CtrlR:**

```
# Saltar dependiendo del código de error al introducir la hora local

# 0 -> hora

# 1 -> minutos

# 2 -> segundos

beqz $t0, ErrorHoras           # Si es igual a cero es que fue un error
                                al introducir la hora
```



```
la $a0, errorMinutosSegundos      # Si no, fue un error de minutos o
                                   segundos, cargamos mensaje minutos y
                                   segundos
```

```
j PrintError
```

#### **ErrorHoras:**

```
la $a0, errorHora                 # Mensaje error horas
```

#### **PrintError:**

```
li $v0 4                         # Imprimimos la cadena previamente cargada
```

```
syscall
```

```
beqz $t0, PedirHoras             # Si es igual a cero, retornamos a pedir la hora
```

```
addi $s1, $0, 1                  # Si es igual a uno
```

```
beq $t0, $s1, PedirMinutos       # Retornamos a pedir los minutos
```

```
j PedirSegundos                  # Si no, retornamos a pedir los segundos
```

## Modificaciones en el exception.s

```
# SPIM S20 MIPS simulator.
# The default exception handler for spim.
#
# Copyright (c) 1990-2010, James R. Larus.
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification,
# are permitted provided that the following conditions are met:
#
# Redistributions of source code must retain the above copyright notice,
# this list of conditions and the following disclaimer.
#
# Redistributions in binary form must reproduce the above copyright notice,
# this list of conditions and the following disclaimer in the documentation and/or
# other materials provided with the distribution.
#
# Neither the name of the James R. Larus nor the names of its contributors may be
# used to endorse or promote products derived from this software without specific
# prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
# AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
# ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
# LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
# CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
# GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
# OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#

# Define the exception handling code.  This must go first!

        .kdata
__m1_: .asciiz "  Exception "
__m2_: .asciiz " occurred and ignored\n"
__e0_: .asciiz "  [Interrupt] "
__e1_: .asciiz " [TLB]"
__e2_: .asciiz " [TLB]"
__e3_: .asciiz " [TLB]"
__e4_: .asciiz " [Address error in inst/data fetch] "
__e5_: .asciiz " [Address error in store] "
__e6_: .asciiz " [Bad instruction address] "
__e7_: .asciiz " [Bad data address] "
__e8_: .asciiz " [Error in syscall] "
__e9_: .asciiz " [Breakpoint] "
__e10_: .asciiz " [Reserved instruction] "
```

```

__e11_:.asciiz ""
__e12_:.asciiz " [Arithmetic overflow] "
__e13_:.asciiz " [Trap] "
__e14_:.asciiz ""
__e15_:.asciiz " [Floating point] "
__e16_:.asciiz ""
__e17_:.asciiz ""
__e18_:.asciiz " [Coprocc 2]"
__e19_:.asciiz ""
__e20_:.asciiz ""
__e21_:.asciiz ""
__e22_:.asciiz " [MDMX]"
__e23_:.asciiz " [Watch]"
__e24_:.asciiz " [Machine check]"
__e25_:.asciiz ""
__e26_:.asciiz ""
__e27_:.asciiz ""
__e28_:.asciiz ""
__e29_:.asciiz ""
__e30_:.asciiz " [Cache]"
__e31_:.asciiz ""
__excp:.word __e0_, __e1_, __e2_, __e3_, __e4_, __e5_, __e6_, __e7_, __e8_, __e9_
        .word __e10_, __e11_, __e12_, __e13_, __e14_, __e15_, __e16_, __e17_, __e18_,
        .word __e19_, __e20_, __e21_, __e22_, __e23_, __e24_, __e25_, __e26_, __e27_,
        .word __e28_, __e29_, __e30_, __e31_
s1:    .word 0
s2:    .word 0

```

```
#####
```

#### # Modificación

```
#####
```

```
    save0:    .word 0    # Registro para guardar la dirección $ra
```

```
#####
```

#### # Final de la modificación

```
#####
```

```

# This is the exception handler code that the processor runs when
# an exception occurs. It only prints some information about the
# exception, but can server as a model of how to write a handler.
#
# Because we are running in the kernel, we can use $k0/$k1 without
# saving their old values.

```

```
# This is the exception vector address for MIPS-1 (R2000):
```

```
#    .ktext 0x80000080
```

```
# This is the exception vector address for MIPS32:
```

```
    .ktext 0x80000180
```

```
# Select the appropriate one for the mode in which SPIM is compiled.
```

```
    .set noat
```

```
    move $k1 $at    # Save $at
```

```

.set at
sw $v0 s1          # Not re-entrant and we can't trust $sp
sw $a0 s2          # But we need to use these registers

mfc0 $k0 $13       # Cause register
srl $a0 $k0 2      # Extract ExcCode Field
andi $a0 $a0 0x1f

#####
# Modificación
#####
GET_INT:
    beqz $a0, HANDLE_INT # Salto que nos lleva a nuestro programa si hay una
                           interrupción

    nop

#####
# Final de la modificación
#####

    # Print information about exception.
    #
    li $v0 4          # syscall 4 (print_str)
    la $a0 __m1_
    syscall

    li $v0 1          # syscall 1 (print_int)
    srl $a0 $k0 2      # Extract ExcCode Field
    andi $a0 $a0 0x1f
    syscall

    li $v0 4          # syscall 4 (print_str)
    andi $a0 $k0 0x3c
    lw $a0 __excp($a0)
    nop
    syscall

    bne $k0 0x18 ok_pc # Bad PC exception requires special checks
    nop

    mfc0 $a0 $14       # EPC
    andi $a0 $a0 0x3    # Is EPC word-aligned?
    beq $a0 0 ok_pc
    nop

    li $v0 10          # Exit on really bad PC
    syscall

ok_pc:
    li $v0 4          # syscall 4 (print_str)
    la $a0 __m2_

```

```

        syscall

        srl $a0 $k0 2          # Extract ExcCode Field
        andi $a0 $a0 0x1f
        bne $a0 0 ret          # 0 means exception was an interrupt
        nop

# Interrupt-specific code goes here!
# Don't skip instruction at EPC since it has not executed.

#####
# Modificación
#####
HANDLE_INT:                      # Salta hasta CaseIntr
    sw $ra save0

    la $v0 CaseIntr             # Guardamos la direccion en $v0 para poder saltar
    jalr $v0                    # Saltamos a la subrutina CaseIntr

    lw $ra save0

    j JUM_INT                   # SALTO PARA EVITAR LA SUMA DE PC
#####
# Final de la modificación
#####

ret:
# Return from (non-interrupt) exception. Skip offending instruction
# at EPC to avoid infinite loop.
#
    mfc0 $k0 $14                # Bump EPC register
    addiu $k0 $k0 4              # Skip faulting instruction
                                   # (Need to handle delayed branch case here)
    mtc0 $k0 $14

#####
# Modificación
#####
JUM_INT:
#####
# Final de la modificación
#####

# Restore registers and reset procesor state
#
    lw $v0 s1                   # Restore other registers
    lw $a0 s2

    .set noat
    move $at $k1                # Restore $at

```

```

        .set at

        mtc0 $0 $13          # Clear Cause register

        mfc0 $k0 $12         # Set Status register
        ori  $k0 0x1         # Interrupts enabled
        mtc0 $k0 $12

# Return from exception on MIPS32:
        eret

# Return sequence for MIPS-I (R2000):
#        rfe                  # Return from exception handler
#                               # Should be in jr's delay slot
#        jr $k0
#        nop

# Standard startup code.  Invoke the routine "main" with arguments:
#        main(argc, argv, envp)
#
        .text
        .globl __start
__start:
        lw $a0 0($sp)        # argc
        addiu $a1 $sp 4      # argv
        addiu $a2 $a1 4      # envp
        sll $v0 $a0 2
        addu $a2 $a2 $v0
        jal main
        nop

        li $v0 10
        syscall              # syscall 10 (exit)

        .globl __eoth
__eoth:

```

### 3. Mejora de la práctica final

### Descripción

Aquí adjuntaré la mejora de la práctica que he realizado. Se trata de una mejora en la que incluyo un juego y una calculadora. Para la realización de esta mejora no he necesitado de la modificación del exception.s que ya había modificado para la realización de la práctica final. Cabe añadir que en este programa solo intervienen las interrupciones por teclado y no las del timer.

Aquí podemos ver dos fotos de ejemplo del funcionamiento de la misma:

```

* Console
* ~~~~~
* Bienvenido a MathQuiz
* ~~~~~

Introduce tu nombre: Ana

Bienvenid@, Ana

* ~~~~~
* Que quieres hacer?
* ~~~~~
* CTRL + S -> Calculadora
* CTRL + T -> Jugar
* CTRL + X -> Salir
* ~~~~~

* ~~~~~
* Que quieres calcular?
* ~~~~~

Ten en cuenta que si introduces una letra,
sera como introducir un 0

Introducir primer numero: 5

Introducir segundo numero: 2

* ~~~~~
* Seleccione la operacion quiera realizar:
* ~~~~~
* Sumar = 0
* Restar = 1
* Multiplicar = 2
* Dividir = 3
* ~~~~~

--> Ha seleccionado = 0

--> Resultado = 7

* ~~~~~
* Desea realizar otra operacion:
* ~~~~~
* SI = 0
* NO = 1
* ~~~~~

--> Ha seleccionado = 1

* ~~~~~
* Introduce otro comando si quieres continuar
* ~~~~~

```

```

*~::~::~::~::~::~::~::~::~::~::~::~::~::~::~::~*
*          Vamos a jugar!!!                      *
*~::~::~::~::~::~::~::~::~::~::~::~::~::~::~::~*

Luca debe escribir un texto con 120 palabras y ya ha
escrito 47. Indica cuantas le faltan: 73

~ Muy bien! Respuesta correcta :)

En un tren habian 200 personas. Al llegar a la
estacion bajaron 95 y subieron al tren 30.
Cuantas personas iban en el tren al salir de la estacion? 100

~ Oh, prueba suerte con la siguiente! Respuesta incorrecta :(

Que numero es menor de 50 y es divisible por 2,
3 y 5? 30

~ Muy bien! Respuesta correcta :)

Hay gatos en un cajon, cada gato en un rincon,
cada gato ve tres gatos, Sabes cuantos son? 5

~ Oh, prueba suerte con la siguiente! Respuesta incorrecta :(

*~::~::~::~::~::~::~::~::~::~::~::~::~::~::~::~*
*          OH! Se me han acabado las preguntas!  *
*          Espero que hayas disfrutado            *
*~::~::~::~::~::~::~::~::~::~::~::~::~::~::~::~*

--> El numero de preguntas correctas ha sido = 2
--> El numero de preguntas incorrectas ha sido = 2

*~::~::~::~::~::~::~::~::~::~::~::~::~::~::~::~*
*          Vuelve pronto!!!                      *
*~::~::~::~::~::~::~::~::~::~::~::~::~::~::~::~*
*          *                                     *
* Autora: Ana Isabel Santana Medina              *
* Grupo: 43                                       *
*~::~::~::~::~::~::~::~::~::~::~::~::~::~::~::~*

```

## Desarrollo de la mejora

```

##
## Nombre: Ana Isabel Santana Medina
##
## Grupo: 43
##
## Fecha: 04/02/18
#####
#####

# Definición de los segmentos de datos

.data 0xFFFF0000

#Registros de los dispositivos de entrada/salida

tControl:      .space 4  #Dirección del registro de control del teclado
                  (0xFFFF0000)

tData:         .space 4  #Dirección del registro de datos del teclado
                  (0xFFFF0004)

pControl:      .space 4  #Dirección del registro de control de pantalla
                  (0xFFFF0008)

pData:         .space 4  #Dirección del registro de datos de pantalla
                  (0xFFFF000C)

```



```
# Mensajes interfaz
```

```
saltoLinea:      .asciiiz "\n\n"
nombre:          .asciiiz "Introduce tu nombre: "
eNombre:         .space 10
bienvenidaN:     .asciiiz "Bienvenid@, "
```

```
pruebal:      .asciiz "Luca debe escribir un texto con 120 palabras y ya
ha
```

```

escrito 47. Indica cuantas le faltan: "
prueba2: .asciiiz "En un tren habian 200 personas. Al llegar a la
estacion bajaron 95 y subieron al tren 30.
Cuantas personas iban en el tren al salir de la estacion? "
prueba3: .asciiiz "Que numero es menor de 50 y es divisible por 2,
3 y 5? "
prueba4: .asciiiz "Hay gatos en un cajon, cada gato en un rincon,
cada gato ve tres gatos, Sabes cuantos son? "

# Respuestas

respuesta: .word 0

correct: .asciiiz "\n\t~ Muy bien! Respuesta correcta :)\n\n"
incorrect: .asciiiz "\n\t~ Oh, prueba suerte con la siguiente! Respuesta
incorrecta :(\n\n"
finpregunt: .asciiiz
"*~~~~~*
* OH! Se me han acabado las preguntas! *
* Espero que hayas disfrutado *
*~~~~~*\n"

ncorrectas: .asciiiz "\n\t--> El numero de preguntas correctas ha sido =
"
nincorrectas: .asciiiz "\n\t--> El numero de preguntas incorrectas ha sido = "

#Espacio calculadora

nota: .asciiiz "\nTen en cuenta que si introduces una letra,
sera como introducir un 0\n"
primerNumero: .asciiiz "\nIntroducir primer numero: "
segundoNumero: .asciiiz "\nIntroducir segundo numero: "

operacion: .asciiiz
"\n*~~~~~*
* Seleccione la operacion quiera realizar: *
* *
* Sumar = 0 *
* Restar = 1 *
* Multiplicar = 2 *
* Dividir = 3 *
*~~~~~*\n"

seleccionada: .asciiiz "\n\t--> Ha seleccionado = "
resultado: .asciiiz "\n\t--> Resultado = "
resto: .asciiiz "\n\t--> Resto de la division = "
firstNum: .word -1
secondNum: .word 16

otraVez: .asciiiz
"\n*~~~~~*

```

```

* Desea realizar otra operacion:
*
* SI = 0
* NO = 1
*~~~~~*\n"

fraseFinalCal: .asciiz
"\n*~~~~~*

* Introduce otro comando si quieres continuar
*~~~~~*\n"

#Variable para guardar dirección de retorno
lastAddress: .word 0

# Segmento de texto

.text 0x00400000
.globl main

# Programa principal

main:

jal KbdIntrEnable

jal PrintWelcome

ReadChar: # Bucle que espera una interrupción

j ReadChar

KbdIntrEnable:

lw $s3, tControl # Carga el registro de control del teclado
ori $s3, $s3, 2 # Habilitacion de interrupciones por teclado
sw $s3, tControl

mfc0 $t3, $12 # Cargamos en t3 el registro status
ori $t3, $t3, 0x801 # Ponemos a 1 los bits que nos interesan del "status"
mtc0 $t3, $12 # Cargamos en el coprocesador t3 habilitando
las interrupciones

jr $ra

CaseIntr:

sw $ra, lastAddress # Guardamos la dirección de retorno

mfc0 $t2, $13 # Obtenemos registro cause del coprocesador0
para ver de donde procede la interrupción

```

```

andi $v0, $t2, 0x800          #Comprobamos si es una interrupcion del teclado
bnez $v0, KbdIntr

lw $ra, lastAddress
jr $ra

```

#### **KbdIntr:**

```

sw $ra, lastAddress

lb $t1, tData

beq $t1, 0x13, Calculadora    #Comprobamos si es igual a CTRL+S
beq $t1, 0x14, Juego          #Comprobamos si es igual a CTRL+T
beq $t1, 0x18, Exit            #Comprobamos si es igual a CTRL+X

lw $ra, lastAddress
jr $ra

```

#### **PrintWelcome:**

```

sw $ra, lastAddress

li $v0, 4
la $a0, string1                # Imprimimos frase de MathQuiz
syscall

li $v0, 4
la $a0, nombre                 # Pedimos nombre
syscall

li $v0, 8
la $a0, eNombre                # Guardamos nombre
li $a1, 10
syscall

li $v0, 4
la $a0, saltoLinea            # Salto de línea
syscall

li $v0, 4
la $a0, bienvenidaN           # Imprimimos la bienvenida con el nombre
syscall

li $v0, 4
la $a0, eNombre
syscall

li $v0, 4
la $a0, saltoLinea            # Salto de línea
syscall

```

```

li $v0, 4
la $a0, string2          # Preguntamos que quiere hacer el usuario
syscall

lw $ra, lastAddress

jr $ra

```

#### **Juego:**

```

la $t2, respuesta

li $v0, 4
la $a0, string4          # Imprime "Vamos a jugar!!"
syscall

```

#### **Pregunta1:**

```

li $v0, 4
la $a0, pruebal          # Imprime la pregunta 1
syscall

li $v0, 5
syscall                  # Lee el entero que se introduce por teclado

move $t4, $v0            # Guardamos el entero en una variable para compararlo

addi $t2, $0, 73         # Guardamos la respuesta en una variable

beq $t2, $t4, Correcto1  # Si ambos son iguales saltamos a mostrar el mensaje
                        de correcto

li $v0, 4
la $a0, incorrect        # Mostramos mensaje de incorrecto
syscall

addi $t6, $t6, 1         # Contador respuestas incorrectas

```

#### **Pregunta2:**

```

and $t2, $t2, $0         # Reiniciamos los registros, el resto funciona
and $t4, $t4, $0         # de la misma manera que el método anterior

li $v0, 4
la $a0, prueba2
syscall

li $v0, 5
syscall

```

```

move $t4, $v0

addi $t2, $0, 135

beq $t4, $t2, Correcto2

li $v0, 4
la $a0, incorrect
syscall

addi $t6, $t6, 1

```

#### **Pregunta3:**

```

and $t2, $t2, $0      # Reiniciamos los registros, el resto funciona
and $t4, $t4, $0      # de la misma manera que el método anterior

li $v0, 4
la $a0, prueba3        #Pregunta 3
syscall

li $v0, 5
syscall

move $t4, $v0

addi $t2, $0, 30

beq $t4, $t2, Correcto3

li $v0, 4
la $a0, incorrect
syscall

addi $t6, $t6, 1

```

#### **Pregunta4:**

```

and $t2, $t2, $0      # Reiniciamos los registros, el resto funciona
and $t4, $t4, $0      # de la misma manera que el método anterior

li $v0, 4
la $a0, prueba4        #Pregunta 4
syscall

li $v0, 5
syscall

move $t4, $v0

addi $t2, $0, 4

```

```
beq $t4, $t2, Correcto4
```

```
li $v0, 4  
la $a0, incorrect  
syscall
```

```
addi $t6, $t6, 1
```

#### **FinPreguntas:**

```
li $v0, 4  
la $a0, finpregunt      # Mensaje fin de las preguntas  
syscall
```

```
li $v0, 4  
la $a0, ncorrectas      # Preguntas correctas  
syscall
```

```
li $v0, 1  
add $a0, $0, $t5  
syscall
```

```
li $v0, 4  
la $a0, nincorrectas    # Preguntas incorrectas  
syscall
```

```
li $v0, 1  
add $a0, $0, $t6  
syscall
```

```
li $v0 4  
la $a0, saltoLinea      # Salto de línea  
syscall
```

```
jal Exit
```

#### **Correcto1:**

```
li $v0, 4  
la $a0, correct          # Mostramos mensaje de correcto  
syscall
```

```
addi $t5, $t5, 1        # Contador preguntas correctas
```

```
jal Pregunta2
```

#### **Correcto2:**

```
li $v0, 4  
la $a0, correct          # Mostramos mensaje de correcto
```

```

syscall

addi $t5, $t5, 1          # Contador preguntas correctas

jal Pregunta3

```

#### **Correcto3:**

```

li $v0, 4
la $a0, correct           # Mostramos mensaje de correcto
syscall

addi $t5, $t5, 1          # Contador preguntas correctas

jal Pregunta4

```

#### **Correcto4:**

```

li $v0, 4
la $a0, correct           # Mostramos mensaje de correcto
syscall

addi $t5, $t5, 1          # Contador preguntas correctas

jal FinPreguntas

```

#### **Calculadora:**

```

sw $ra, lastAddress

li $v0, 4
la $a0, string5
syscall

```

#### **getPrimerNumero:**

```

li $v0 4
la $a0, nota
syscall

li $v0 4
la $a0, primerNumero    # Pedimos el primer número
syscall

li $v0, 5                # Guardamos el número y lo almacenamos
syscall

la $s1, firstNum
sw $v0, 0($s1)

```

#### **getSegundoNumero:**



```

li $v0 4
la $a0, segundoNumero # Pedimos el segundo número
syscall

li $v0, 5 # Guardamos el número y lo almacenamos

syscall

la $s2, secondNum
sw $v0, 0($s2)

getOperacion:

li $v0 4
la $a0, operacion # Pedimos la operación que se desea realizar
syscall

li $v0 4
la $a0, seleccionada # Operacion a realizar
syscall

li $v0, 5 # Opción
syscall

hacerOperacion:

la $s1, firstNum # Primer número
lw $s2, 0($s1)

la $s1, secondNum #Segundo número
lw $s3, 0($s1)

# Comparamos con la opción seleccionada

beq $v0, 0, sumar
beq $v0, 1, restar
beq $v0, 2, multiplicar
beq $v0, 3, dividir

sumar:

jal ImprimirFraseResultado
add $a0, $s2, $s3 # Sumar
jal ImprimirResultado
j OperacionFinal

restar:

jal ImprimirFraseResultado
sub $a0, $s2, $s3 # Restar

```

```
jal ImprimirResultado
j OperacionFinal
```

#### **multiplicar:**

```
jal ImprimirFraseResultado

mult $s2, $s3          # Multiplicar

mfhi $a0               # Guarda el valor más significativo
bnez $t0 ImprimirResultado

mflo $a0               # Guarda el valor menos significativo
jal ImprimirResultado

j OperacionFinal
```

#### **dividir:**

```
jal ImprimirFraseResultado

div $s2, $s3

mflo $a0               # Cociente
jal ImprimirResultado

li $v0, 4
la $a0, resto          # Resto
syscall

mfhi $a0
jal ImprimirResultado

j OperacionFinal
```

#### **ImprimirFraseResultado:**

```
li $v0, 4
la $a0, resultado      # Imprime frase "Resultado = "
syscall

jr $ra
```

#### **ImprimirResultado:**

```
li $v0, 1              # Imprime el resultado
syscall

jr $ra
```

#### **OperacionFinal:**

```

li $v0 4
la $a0, saltoLinea      # Salto de línea
syscall

li $v0 4
la $a0, otraVez          # Preguntamos si se quiere realizar otra operacion
syscall

li $v0 4
la $a0, seleccionada     # Opción seleccionada
syscall

li $v0, 5                 # Opción
syscall

beqz $v0, Calculadora    # Si es cero es que se quiere realizar otra operación

```

#### **final\_Calculadora:**

```

li $v0 4
la $a0, fraseFinalCal    # Si es un 1, saltamos aquí, mostramos frase de
syscall                  # final de calculadora

li $v0 4
la $a0, saltoLinea      # Salto de línea
syscall

lw $ra lastAddress       # Volveremos al bucle de pulsar un comando
jr $ra

```

#### **Exit:**

```

li $v0, 4
la $a0, string3          # Imprimimos string de despedida
syscall

li $v0, 10
syscall                  # Fin de programa

```

## 4. Conclusiones

Las conclusiones que he obtenido de la realización de esta primera práctica han sido que, en primer lugar, las competencias que se habían planteado han sido logradas gracias al estudio y trabajo de la misma. En segundo lugar, los conocimientos del MIPS-32 y el lenguaje ensamblador han sido reforzados; la resolución de esta práctica ha supuesto un reto, pues he tenido que entender el funcionamiento de las excepciones e interrupciones, los registros que resultaban implicados, etc. Por último, me gustaría resaltar, que ha sido un proceso grato de aprender, pues, aunque fue complicado al principio, fue gratificante el hecho de que al final conseguí resolver la última práctica y proponerme mi propio reto al realizar la mejora.

Para la mejora de la misma pensé en un juego que no resultase muy complejo, que fuera de matemáticas y que tuviera una calculadora para realizar cálculos; y que, además, estuvieran algunos de los conceptos aprendidos en la práctica. Finalmente ha resultado bastante divertida la realización de la misma.

## 5. Comentarios finales

Como comentarios finales, he de añadir que, en mi opinión, han resultado suficiente los recursos que nos han facilitado para la realización de la práctica, puesto que aparte de estos, teníamos la capacidad de acceder a internet, y, además, podíamos probar todos los programas y tareas que realizábamos en cualquier momento gracias al PCSpim, y comprobar el funcionamiento de las mismas. Además, la realización del control ha resultado un reto pues, aunque son conceptos que habíamos aprendido en la práctica, había que tener en cuenta la modificación del `exception.s`.

## 6. Recursos empleados

Para la realización de esta práctica hemos hecho uso de los siguientes recursos:

- **PCSpim (ver. 8):**

Como nombramos anteriormente, hemos hecho uso de este recurso software para simular nuestros programas

- **Documentación de la Práctica 1 – Excepciones e Interrupciones**

De este recurso es de donde más información he obtenido, ya que estaba ahí la mayoría de información que me interesaba para el desarrollo de la práctica

- **Guía de las prácticas de la asignatura Introducción a la informática**

Este recurso lo he utilizado más que nada para recordar conceptos a la hora de comenzar la práctica.