PRÁCTICA 4

Introducción al WinMIPS64. Instrucciones. E/S.

Objetivos: Familiarizarse con el desarrollo de programas para procesadores con sets reducidos de instrucciones (RISC). Resolver problemas y verificarlos a través de simulaciones. Comprender el uso del puerto de E/S provisto en el WinMIPS64.

Parte 1: Introducción al set de instrucciones del WinMIPS64

1. Tipos de instrucciones en WinMIPS64 ⭐

Para cada una de las siguientes instrucciones del WinMIPS64, indicar si son instrucciones de salto incondicional (SI), salto condicional (SC), de lectura de memoria (LMEM), escritura de memoria (EMEM), o aritmético-lógicas (AL):

`and`	`bnez`	`halt`	`slt`	`ld`	`sb`
`andi`	`dadd`	`nop`	`slti`	`lb`	`sd`
`beq`	`daddi`	`or`	`j`	`lbu`	`sw`
`bne`	`dmul`	`ori`	`jal`	`lw`	`xor`
`beqz`	`ddiv`		`jr`	`lwu`	`xori`

2. VonSim vs WinMIPS64 ⭐

Indicar qué instrucciones se corresponden entre el simulador VonSim y el simulador WinMIPS64. Tener en cuenta que algunas de las instrucciones del VonSim pueden hacerse de distintas formas en WinMIPS64. y otras no pueden hacerse con una sola instrucción. Por ende, algunas instrucciones de la columna izquierda corresponden a varias de la derecha, y otras no corresponden a ninguna

Nota: para simplificar, usaremos los registros del WinMIPS64 (r0 a r31) como nombres de registro en las instrucciones del VonSim.

VonSim

`mov r1, r2`
`mov r1, 1`
`mov r1, 0`
`add r1, r2`
`add r1, 1`
`add r1, 0`
`inc r1`
`dec r1`
`or r1, r2`
`or r1, 1`
`or r1, 0`
`mov r1, variable`
`mov variable, r1`
`add r1, variable`
`add variable, r1`
`mov r1, offset variable`
`jump etiqueta`
`call etiqueta`
`hlt`

WinMIPS64

`ld r1, variable(r0)`
`sd r1, variable(r0)`
`halt`
`jal etiqueta`
`j etiqueta`
`or r1, r1, r0`
`or r1, r1, r2`
`ori r1, r1, 1`
`dadd r1, r1, r0`
`daddi r1, r0, 0`
`daddi r1, r0, 1`
`dadd r1, r0, r0`
`dadd r1, r0, r2`
`daddi r1, r1, 1`
`dadd r1, r1, r2`
`daddi r1, r0, variable`
`daddi r1, r2, 0`
`daddi r1, r1, -1`

3. Tipos de variables en WinMIPS64 ⭐

Indicar qué tamaño (en bytes) ocupan cada uno de estos tipos de datos del simulador WinMIPS64. En el caso en que dos tipos de datos tengan el mismo tamaño, explicar sus diferencias:

Tipo de dato	Tamaño en bytes	Uso
`space 1`		
`ascii`		
`asciiz`		
`byte`		
`word16`		
`word32`		
`word`		
`double`		

4. Registros del MIPS64 ⭐.

El procesador MIPS64 posee 32 registros, de 64 bits cada uno, llamados r0 a r31 (también conocidos como $0 a $31). Sin embargo, al programar resulta más conveniente darles nombres más significativos a esos registros.
La siguiente tabla muestra la convención empleada para nombrar a los 32 registros mencionados:

Registros	Nombres	¿Para que se los utiliza?	¿Preservado?
r0	$zero		
r1	$at		
r2-r3	$v0-$v1		
r4-r7	$a0-$a3		
r8-r15	$t0-$t7		
r16-r23	$s0-$s7		
r24-r25	$t8-$t9		
r26-r27	$k0-$k1		
r28	$gp		
r29	$sp		
r30	$fp		
r31	$ra		

Complete la tabla anterior explicando el uso que normalmente se le da a cada uno de los registros nombrados. Marque en la columna "*¿Preservado?*" si el valor de cada grupo de registros debe ser preservado luego de realizada una llamada a una subrutina. Puede encontrar información útil en el apunte "Programando sobre MIPS64".

5. Programas simples ⭐⭐.

Escribir un programa en WinMIPS64 que:

a) Lee 2 números A y B de la memoria de datos, y calcula S, P y D, que luego se guardan en la memoria de datos.
b) Dadas dos variables A y B de la memoria, calcula y almacena C
c) Calcula el factorial de N, y lo guarda en F
d) Calcula el logaritmo (entero) en base 2 de N (N positivo) mediante divisiones sucesivas y lo guarda en L
e) Guarda en B el valor 1 si A es impar y 0 de lo contrario
Los nombres de variables como A, B, C, etc, deben implementarse usando variables de tipo **word** a las cuales se lee y escribe de memoria con **ld** y **sd**.

a	b	c	d	e

`S = A + B`
`P =  2 + (A*B)`
`D = A^2 / B`
	`if A == 0:`
	`  C = 0`
	`else:`
	`  if A > B:`
	`  C = A * 2`
	`  else:`
	`  C = B`
		`F = 1`
		`for i=1..N:`
		`  F = F * i`
			`L = 0`
			`while N > 0:`
			`  N = N / 2`
			`  L = L + 1`
				`if impar(A):`
				`  B = 1`
				`else:`
				`  B = 0`

6. Recorrido básico de vectores ⭐

Dado un vector definido como: `V: .word 5, 2, 6`, escribir programas para:

a) Calcular la suma de los 3 valores sin utilizar un loop o lazo
Pista: Usar tres instrucciones `ld $t1, V($t2)`, donde $t2 va aumentando de a 8 bytes el desplazamiento.
b) Calcular la suma de los 3 valores utilizando un lazo con la dirección base y un registro como desplazamiento. **Pista**: Idem anterior, pero ahora con una única instrucciones de lectura y $t2 se incrementa dentro de un loop
c) Calcular la suma de los 3 valores utilizando un lazo, con una dirección base de 0 y un registro como puntero.
Pista: Cargar la dirección en un registro con `daddi $t2, $zero, V` y luego cargar los valores con `ld $t1, 0($t2)`
d) ¿Qué cambios se deberían realizar al programa del inciso b) si los elementos fueran de 32 bits: `V: .word32 5, 2, 6`?

7. Operaciones con vectores ⭐⭐

a) Definir un vector con 10 valores , y escribir programas para:
- **Contar positivos.** Contar la cantidad de elementos positivos, y guardar la cantidad en una variable llamada POS.
- **Calcular máximo.** Calcular el máximo elemento del vector y guardarlo en una variable llamada MAX.
- **Modificar valores** Modificar los elementos del vector, de modo que cada elemento se multiplique por 2.
b) **Impares** Generar un vector con los primeros 10 números impares.
c) **Fibonacci** Generar un vector con los primeros 10 números de la secuencia de fibonacci
d) **Vector de impares a partir de vector** Definir un vector V con 10 números cualesquiera. Escribir un programa que genere un vector W con los números impares de V.

8. Codificación de Strings⭐

a) **Strings en memoria**: En WinMIPS los strings se almacenan como vectores de caracteres. Observar cómo se almacenan en memoria los códigos ASCII de los caracteres (código de la letra “a” es 61H). Además, los strings se suelen terminar con un carácter de fin con código 0, es decir el valor 0. Para definir un string en una variable y no tener que agregar a mano el valor 0, se puede utilizar el tipo de datos **asciiz**. Si en lugar de eso se utiliza el tipo **ascii** (sin la **z)** no se agrega el 0 de forma automática. Probar estas definiciones en el simulador y observar cómo se organizan en la memoria las variables.

```
.data
cadena:   .asciiz "ABCdef1"
cadena2: .ascii "ABCdef11"
cadena3: .asciiz "ABCdef1111111"
num:      .word 5
```

b) **Lectura de caracteres: lbu vs. lb vs. ld**: En un string, cada código ASCII ocupa 1 byte. No obstante, los registros del simulador tienen 8 bytes (64 bits) de capacidad. Por ende, al cargar un byte en un registro, se desperdician 7 bytes de espacio; esta ineficiencia es inevitable. Más allá de eso, esta diferencia nos trae otra dificultad. Cuando se carga un valor de memoria, no se puede utilizar **ld**, porque ello traería 8 caracteres (8 bytes) al registro, y sería muy difícil hacer operaciones sobre los caracteres individuales. Por ello, existen instrucciones para traer solo un byte desde la memoria: **lbu** y **lb**. ¿Cuál es la diferencia?
- **lbu** asume que el valor que se trae está codificado en BSS y entonces rellena los últimos **7 bytes** con **0**.

- **lb** asume que el valor que se trae de memoria está codificado en CA2, y entonces si el número es negativo realiza la *expansión de signo*. ¿De qué trata esto? Para que el número siga valiendo lo mismo en CA2 de 8 **bytes**, se rellenan los últimos **7 bytes** con **1**.

Para probar la diferencia entre estas 3 instrucciones, ejecutar el siguiente programa en el simulador que intenta cargar el primer valor del vector de números **datos** y observar los valores finales de $t1, $t2 y $t3.

```
.data
datos: .byte -2, 2, 2, 2, 2, 2
.code
ld $t1, datos($zero)
lb $t2, datos($zero)
lbu $t3, datos($zero)
halt
```

Responde:
¿Qué registro tiene el valor "correcto" del primer valor?
¿Qué instrucción deberías utilizar de las 3 para cargar un código ASCII que siempre es positivo? Tené en cuenta que, por ejemplo, el código ASCII de la **Á** es 181, que en BSS se escribe como **10110101**.

9. Operaciones con strings ⭐⭐

a) **Longitud de un string** Escribir un programa que cuente la longitud de un string iterando el mismo hasta llegar al valor 0 y guarde el resultado en una variable llamada **LONGITUD.** Probarlo con el string "ArquiTectuRa de ComPutaDoras".
b) **Contar apariciones de carácter** Escribir un programa que cuente la cantidad de veces que un determinado carácter aparece en una cadena de texto.

```
.data
cadena: .asciiz "adbdcdedfdgdhdid"  # cadena a analizar
car:     .ascii "d"                 # carácter buscado
cant:    .word 0  # cantidad de veces que se repite el carácter car en cadena.
```

c) **Contar mayúsculas** Escribir un programa que cuente la cantidad de letras mayúsculas de un string. Probarlo con el string "ArquiTectuRa de ComPutaDoras". **Pista**: El código ASCII de la "A" es 65, y el de la "Z" es 90.
d) **Generar string** Escribir un programa que genere un string de la siguiente forma: "a**bb**ccc**dddd**eeeee…", así hasta la letra "h". Para ello debe utilizar un loop e ir guardando los códigos ascii en la memoria. El string debe finalizar con el valor ascii 0 para que esté bien formado (debe agregar un elemento más, que valga 0, al final del string).

Parte 2: Entrada/Salida

Puerto de E/S mapeado en memoria de datos En el WinMIPS, para hacer operaciones de E/S con el teclado y la pantalla se utiliza la técnica de mapeado en memoria de datos. Para ello, se cuentan con los registros CONTROL y DATA que se encuentran en la memoria de datos y tienen direcciones de memoria fijas. Aplicando distintos comandos a través de CONTROL, es posible producir salidas o ingresar datos a través de la dirección DATA.

Las direcciones de memoria de CONTROL y DATA son 0x10000 y 0x10008 respectivamente. Como el set de instrucciones del procesador MIPS64 no cuenta con instrucciones que permitan cargar un valor inmediato de más de 16 bits (como es el caso de las direcciones mencionadas), no se puede utilizar una instrucción como sd r1, 0x10000(r0) para escribir en CONTROL. Esta limitación no impide usar los registros, pero si hace engorroso su uso. Para usarlos es necesario primero definir variables en la memoria de datos que contengan las direcciones, y luego cargarlas en registros.

```
.data
DIR_CONTROL: .word 0×10000  # DIR_CONTROL tiene la DIRECCIÓN del registro CONTROL
DIR_DATA:    .word 0×10008  # IDEM para DATA
.code
ld  r2, DIR_CONTROL($zero)  # Carga el valor 0×10000 en r2
ld  r3, DIR_DATA($zero)     ;     # Carga el valor 0×10008 en r3
```

Para realizar operaciones de E/S, se encuentran disponibles los siguientes códigos de CONTROL:

- **CONTROL = 1** Si se escribe en `DATA` un número entero y se escribe un 1 en `CONTROL`, se interpretará el valor escrito en `DATA` como un entero sin signo y se lo imprimirá en la pantalla alfanumérica de la terminal.
- **CONTROL = 2** Si se escribe en `DATA` un número entero y se escribe un 2 en `CONTROL`, se interpretará el valor escrito en `DATA` como un entero con signo y se lo imprimirá en la pantalla alfanumérica de la terminal.
- **CONTROL = 3** Si se escribe en `DATA` un número en punto flotante y se escribe un 3 en `CONTROL`, se imprimirá en la pantalla alfanumérica de la terminal el número en punto flotante.
- **CONTROL = 4** Si se escribe en `DATA` la dirección de memoria del comienzo de una cadena terminada en 0 y se escribe un 4 en `CONTROL`, se imprimirá la cadena en la pantalla alfanumérica de la terminal.
- **CONTROL = 5** Si se escribe en `DATA` un color expresado en RGB (usando 4 bytes para representarlo: un byte para cada componente de color e ignorando el valor del cuarto byte), en `DATA+4` la coordenada Y, en `DATA+5` la coordenada X y se escribe un 5 en `CONTROL`, se pintará con el color indicado un punto de la pantalla gráfica de la terminal, cuyas coordenadas están indicadas por X e Y. La pantalla gráfica cuenta con una resolución de 50x50 puntos, siendo (0, 0) las coordenadas del punto en la esquina inferior izquierda y (49, 49) las del punto en la esquina superior derecha.
- **CONTROL = 6** Si se escribe un 6 en `CONTROL`, se limpia la pantalla alfanumérica de la terminal.
- **CONTROL = 7** Si se escribe un 7 en `CONTROL`, se limpia la pantalla gráfica de la terminal.
- **CONTROL = 8** Si se escribe un 8 en `CONTROL`, se permitirá ingresar en la terminal un número (entero o en punto flotante) y el valor del número ingresado estará disponible para ser leído en `DATA`.
- **CONTROL = 9** Si se escribe un 9 en `CONTROL`, se esperará a que se presione una tecla en la terminal (la cuál no se mostrará al ser presionada) y el código ASCII de dicha tecla estará disponible para ser leído en `DATA`.

1. Lectura e impresión de strings. ⭐

El siguiente programa produce la salida de un mensaje predefinido en la ventana Terminal del simulador WinMIPS64.

```
.data
texto:     .asciiz    "Hola, Mundo!" ; El mensaje a mostrar
CONTROL: .word        0×10000
DATA:      .word      0×10008
.code
  ld $t0, CONTROL($zero)    ; $t0 = dirección de CONTROL
  ld $t1, DATA($zero)       ; $t1 = dirección de DATA
  daddi $t2, $zero, texto   ; $t2 = dirección del mensaje a mostrar
  sd $t2, 0($t1)            ; DATA recibe el puntero al comienzo del mensaje

  daddi $t2, $zero, 4       ; $t2 = 4 → función 4: salida de una cadena ASCII
  sd     $t2, 0($t0)        ; CONTROL recibe 4 y produce la salida del mensaje
  halt
```

a) ¿Qué valor tienen los registros $t0 y $t1?
b) Modifique el programa (a) para que el mensaje se muestre 10 veces.

2. Comprobación de clave ⭐⭐

a) **Comprobación simple** Escriba un programa que solicite el ingreso por teclado de una clave, representada por un string de 4 carácteres. Para indicar al usuario que debe ingresar un valor, imprimir en pantalla "Ingrese una clave de 4 carácteres". Luego, debe comparar la secuencia ingresada con una cadena almacenada en la variable `clave`. Si las dos cadenas son iguales entre sí, mostrar el texto "Clave correcta: acceso permitido" en la salida estándar del simulador (ventana Terminal). En cambio, si las cadenas no son iguales, mostrar "Clave incorrecta."
b) **Comprobación infinita** Modificar el programa anterior para solicitar nuevamente el ingreso de la clave cuando es incorrecta. El programa solo termina cuando la clave ingresada es la correcta
b) **Comprobación con intentos** ⭐⭐⭐ Modificar el programa anterior para que el programa termine luego de 5 intentos fallidos. Indicar la cantidad de intentos restantes con el mensaje: "Ingrese una clave de 4 carácteres (X intentos restantes)". **Pista:** Para imprimir este mensaje, utilice 3 impresiones: primero con el comienzo del string, luego el valor X y luego el resto del string.

3. Lectura e impresión de números enteros

a) **Suma de números** ⭐ Escribir un programa que lea dos números enteros y muestre su suma en la salida estándar del simulador (ventana Terminal) el resultado numérico.
b) **Operaciones con números** ⭐⭐ Modificar a) para que además de leer los números, se lea un carácter que corresponda a distintas operaciones: **+**, **-**, ***** y **/**. Calcule la operación correspondiente (suma, resta, multiplicación o división). Asumir que el usuario siempre ingresa un carácter correspondiente a una operación válida.
c) **Mini Calculadora** ⭐⭐⭐ Modificar b) para que el programa funcione como una calculadora común. Para ello, lea primero un número, y luego lea pares de (operación, número). Cada vez que lee un par (operación, número), aplique la misma al resultado anterior y lo muestra en la pantalla.

d) **Superficie de un triángulo rectángulo** ⭐⭐ Escribir un programa que calcule la superficie de un triángulo rectángulo de base **B** y altura **A**, y almacene el resultado en una variable llamada **superficie**. Debe leer el valor de A y B por teclado, y mostrar el resultado.
Pista: la superficie de un triángulo se calcula como: `Superficie = (base x altura) / 2`

4. Pantalla gráfica: puntos y líneas

El siguiente programa pinta el pixel con coordenadas (24,24) de la pantalla gráfica de color de magenta. Ir a **Ventana → Terminal** en el simulador WinMIPS64 para ver el resultado luego de ejecutar el programa.

```
.data
    coorX:  .byte   24  ; coordenada X de un punto
    coorY:  .byte   24  ; coordenada Y de un punto
    color:  .byte   255, 0, 255, 0  ; color: máximo rojo + máximo azul ⇒ magenta
    CONTROL:.word 0×10000
    DATA:   .word 0×10008
```

```
.code
      ld    $t0, CONTROL($zero)     ; $t0 = dirección de CONTROL
      ld    $t1, DATA($zero)        ; $t1 = dirección de DATA

      lwu   $t2, color($zero)       ; $t2 = valor de color a pintar (4 bytes)
      sw    $t2, 0($t1)             ; DATA recibe el valor del color a pintar
      lbu   $t2, coorX($zero)       ; $t2 = valor de coordenada X
      sb    $t2, 5($t1)             ; DATA+5 recibe el valor de coordenada X
      lbu   $t2, coorY($zero)       ; $t2 = valor de coordenada Y
      sb    $t2, 4($t1)             ; DATA+4 recibe el valor de coordenada Y

      daddi $t2, $zero, 5           ; $t2 = 5 → función 5: salida gráfica
      sd    $t2, 0($t0)             ; CONTROL recibe 5 y produce el dibujo del punto
      halt
```

a) **Desde teclado**⭐ Modifique el programa anterior para que las coordenadas X e Y del punto a pintar se lean de teclado.

b) **Cambio de color** ⭐ Modifique el programa original para que el punto vaya cambiando de color, desde negro (255,0,0) hasta rojo puro (255,0,0).

c) **Línea azul** ⭐⭐ Modifique el programa original para pintar una línea horizontal de 50 pixeles desde el **0,0** hasta el **0,49** de azul puro.

5. Pantalla Gráfica: Cuadrados

```
.data
X:          .byte 45
Y:          .byte 0
color:      .byte 255, 0, 0, 0
CONTROL:    .word32 0×10000
DATA:       .word32 0×10008
.code
lwu $s0, CONTROL($zero)
lwu $s1, DATA($zero)
lwu $t0, color($zero)
sw $t0, 0($s1)
_____________________
lbu $t2,X($zero)
```

```
      daddi $t4,$zero,50
      daddi $t5,$zero,5
loop: sb $t1,4($s1)
      ____________________
      daddi $t3, $zero, 5
      sd $t3,0($s0)
      daddi $t2, $t2, 1
      bne $t4,$t2,loop
      ____________________
      daddi $t1,$t1,1
      bne $t5, $t1, loop
      halt
```

a) **Cuadrado rojo de 5x5** Completa las líneas faltantes para que el programa anterior pinte un cuadrado rojo de 5x5 en la esquina superior izquierda.

b) **Pantalla verde** ⭐⭐ Modifique el programa anterior para pintar **toda** la pantalla de verde

c) **Líneas de colores** ⭐⭐⭐ Modifique el programa anterior para pintar toda la pantalla, pero cada línea de un color distinto (idear una forma de variar los colores)