

## Trabajo Práctico N° 6: **Procesador RISC (Utilizando la E/S).**

### Ejercicio 1.

*El siguiente programa produce la salida de un mensaje predefinido en la ventana Terminal del simulador WinMIPS64. Teniendo en cuenta las condiciones de control del puerto de E/S (en el resumen anterior), modificar el programa de modo que el mensaje a mostrar sea ingresado por teclado en lugar de ser un mensaje fijo.*

```
.data
TEXTO:    .asciiz "Hola, Mundo!"      ; El mensaje a mostrar
CONTROL:   .word32 0x10000
DATA:     .word32 0x10008

.code
lwu $s0, DATA($zero)      ; $s0 = dirección de DATA
daddi $t0, $zero, TEXT0    ; $t0 = dirección del mensaje a mostrar
sd $t0, 0($$s0)           ; DATA recibe el puntero al comienzo del
mensaje
lwu $s1, CONTROL($zero)    ; $s1 = dirección de CONTROL
daddi $t0, $zero, 6         ; $t0 = 6 -> función 6: limpiar pantalla
alfanumérica
sd $t0, 0($$s1)            ; CONTROL recibe 6 y limpia la pantalla
alfanumérica
daddi $t0, $zero, 4         ; $t0 = 4 -> función 4: salida de una cadena
ASCII
sd $t0, 0($$s1)            ; CONTROL recibe 4 y produce la salida del
mensaje
halt

.data
CONTROL:   .word 0x10000
DATA:     .word 0x10008
MSJ:      .asciiz "INTRODUCIR Hola, Mundo!:\n"
STR:      .asciiz ""

.code
ld $s0, CONTROL($0)
ld $s1, DATA($0)
daddi $t1, $0, MSJ
sd $t1, 0($$s1)
daddi $t0, $0, 4
sd $t0, 0($$s0)
daddi $t0, $0, 9
dadd $t2, $0, $0
daddi $t3, $0, 12
LAZO:    sd $t0, 0($$s0)
```

```
lbu $t1, 0($$s1)
sb $t1, STR($t2)
daddi $t3, $t3, -1
daddi $t2, $t2, 1
bnez $t3, LAZO
daddi $t1, $0, STR
sd $t1, 0($$s1)
daddi $t0, $0, 4
sd $t0, 0($$s0)
halt
```

## Ejercicio 2.

Escribir un programa que utilice, sucesivamente, dos subrutinas: la primera, denominada INGRESO, debe solicitar el ingreso por teclado de un número entero (de un dígito), verificando que el valor ingresado, realmente, sea un dígito; la segunda, denominada MUESTRA, deberá mostrar, en la salida estándar del simulador (ventana Terminal), el valor del número ingresado expresado en letras (es decir, si se ingresa un "4", deberá mostrar "CUATRO"). Establecer el pasaje de parámetros entre subrutinas respetando las convenciones para el uso de los registros y minimizar las detenciones del cauce (ejercicio similar al Ejercicio 6 de la Práctica 2).

```
.data
CONTROL:    .word 0x10000
DATA:        .word 0x10008
MSJ:         .asciiz "INTRODUCIR UN NUMERO ENTERO DE UN
DIGITO:\n"
ERROR:        .asciiz "ERROR"
MENOSNUEVE:  .asciiz "MNUEVE"
MENOSOCHO:   .asciiz "MOCHO"
MENOSSIETE: .asciiz "MSIETE"
MENOSSEIS:   .asciiz "MSEIS"
MENOSCINCO: .asciiz "MCINCO"
MENOSCUATRO: .asciiz "MCUATRO"
MENOSTRES:   .asciiz "MTRES"
MENOSDOS:    .asciiz "MDOS"
MENOSUNO:    .asciiz "MUNO"
CERO:         .asciiz "CERO"
UNO:          .asciiz "UNO"
DOS:          .asciiz "DOS"
TRES:         .asciiz "TRES"
CUATRO:       .asciiz "CUATRO"
CINCO:        .asciiz "CINCO"
SEIS:         .asciiz "SEIS"
SIETE:        .asciiz "SIETE"
OCHO:         .asciiz "OCHO"
NUEVE:        .asciiz "NUEVE"

.code
ld $s0, CONTROL($0)
ld $s1, DATA($0)
daddi $t1, $0, MSJ
sd $t1, 0($s1)
daddi $t0, $0, 4
sd $t0, 0($s0)
jal INGRESO
dadd $a0, $v0, $0
jal MUESTRA
halt
```

INGRESO:

```
daddi $v0, $0, -10
daddi $t0, $0, 8
sd $t0, 0($s0)
ld $t1, 0($s1)
slti $t2, $t1, 10
bez $t2, FIN
dadd $v0, $t1, $0
jr $ra
```

MUESTRA:

```
daddi $t3, $0, 8
dmul $t4, $a0, $t3
daddi $t5, $t4, CERO
sd $t5, 0($s1)
daddi $t6, $0, 4
sd $t6, 0($s0)
jr $ra
```

### Ejercicio 3.

*Escribir un programa que realice la suma de dos números enteros (de un dígito cada uno) utilizando dos subrutinas: la denominada INGRESO del ejercicio anterior (ingreso por teclado de un dígito numérico) y otra denominada RESULTADO, que muestre, en la salida estándar del simulador (ventana Terminal), el resultado numérico de la suma de los dos números ingresados (ejercicio similar al Ejercicio 7 de la Práctica 2).*

```
.data
CONTROL: .word 0x10000
DATA: .word 0x10008
MSJ1: .asciiz "INTRODUCIR DOS NUMEROS ENTEROS:\n"
MSJ2: .asciiz "SU SUMA ES IGUAL A:\n"

.code
ld $s0, CONTROL($0)
ld $s1, DATA($0)
daddi $t1, $0, MSJ1
sd $t1, 0($s1)
daddi $t0, $0, 4
sd $t0, 0($s0)
daddi $t3, $0, -99
jal INGRESO
beq $v0, $t3, FIN1
dadd $a0, $v0, $0
jal INGRESO
beq $v0, $t3, FIN1
dadd $a1, $v0, $0
jal RESULTADO
halt

FIN1:
INGRESO: daddi $v0, $0, -99
          daddi $t0, $0, 8
          sd $t0, 0($s0)
          ld $t1, 0($s1)
          slti $t2, $t1, 10
          beqz $t2, FIN2
          dadd $v0, $t1, $0
FIN2:   jr $ra

RESULTADO: daddi $t1, $0, MSJ2
            sd $t1, 0($s1)
            daddi $t0, $0, 4
            sd $t0, 0($s0)
            dadd $t0, $a0, $a1
            sd $t0, 0($s1)
            daddi $t1, $0, 2
            sd $t1, 0($s0)
            jr $ra
```

## Ejercicio 4.

*Escribir un programa que solicite el ingreso por teclado de una clave (sucesión de cuatro caracteres) utilizando la subrutina CHAR de ingreso de un carácter. Luego, se debe comparar la secuencia ingresada con una cadena almacenada en la variable CLAVE. Si las dos cadenas son iguales entre sí, la subrutina llamada RESPUESTA mostrará el texto “Bienvenido” en la salida estándar del simulador (ventana Terminal). En cambio, si las cadenas no son iguales, la subrutina deberá mostrar “ERROR” y solicitar, nuevamente, el ingreso de la clave.*

```
.data
CONTROL: .word 0x10000
DATA: .word 0x10008
CLAVE: .asciiz "JUAN"
MSJ_BIENV: .asciiz "BIENVENIDO"
MSJ_ERROR: .asciiz "ERROR. INTRODUCIR, NUEVAMENTE, LA
CLAVE:\n"
MSJ: .asciiz "INTRODUCIR UNA CLAVE DE CUATRO
CARACTERES:\n"

.code
daddi $sp, $0, 0x400
ld $s0, CONTROL($0)
ld $s1, DATA($0)
daddi $t1, $0, MSJ
sd $t1, 0($s1)
daddi $t0, $0, 4
sd $t0, 0($s0)
daddi $a2, $0, CLAVE
jal CHAR
halt

CHAR: daddi $sp, $sp, -16
sd $ra, 0($sp)
sd $s2, 8($sp)
daddi $t0, $0, 9
dadd $s2, $a2, $0
daddi $t3, $0, 4
dadd $t4, $0, $0
sd $t0, 0($s0)
lbu $t1, 0($s1)
lbu $t5, 0($s2)
beq $t1, $t5, NO SON IGUALES
daddi $t4, $t4, 1
daddi $t3, $t3, -1
daddi $s2, $s2, 1
beqz $t3, FIN1
j LAZO
beq $t4, $t3, FIN2

LAZO: daddi $sp, $sp, 16
sd $ra, 0($sp)
sd $s2, 8($sp)
daddi $t0, $0, 9
dadd $s2, $a2, $0
daddi $t3, $0, 4
dadd $t4, $0, $0
sd $t0, 0($s0)
lbu $t1, 0($s1)
lbu $t5, 0($s2)
beq $t1, $t5, NO SON IGUALES
daddi $t4, $t4, 1
daddi $t3, $t3, -1
daddi $s2, $s2, 1
beqz $t3, FIN1
j LAZO
beq $t4, $t3, FIN2

NO SON IGUALES: daddi $sp, $sp, 16
sd $ra, 0($sp)
sd $s2, 8($sp)
daddi $t0, $0, 9
dadd $s2, $a2, $0
daddi $t3, $0, 4
dadd $t4, $0, $0
sd $t0, 0($s0)
lbu $t1, 0($s1)
lbu $t5, 0($s2)
beq $t1, $t5, NO SON IGUALES
daddi $t4, $t4, 1
daddi $t3, $t3, -1
daddi $s2, $s2, 1
beqz $t3, FIN1
j LAZO
beq $t4, $t3, FIN2

FIN1:
```

```
daddi $t6, $0, MSJ_ERROR
sd $t6, 0($s1)
jal RESPUESTA
daddi $t3, $0, 4
dadd $t4, $0, $0
dadd $s2, $a2, $0
j LAZO
FIN2:
daddi $t6, $0, MSJ_BIENV
sd $t6, 0($s1)
jal RESPUESTA
ld $ra, 0($sp)
ld $s2, 8($sp)
daddi $sp, $sp, 16
jr $ra

RESPUESTA:
daddi $t7, $0, 4
sd $t7, 0($s0)
jr $ra
```

## Ejercicio 5.

*Escribir un programa que calcule el resultado de elevar un valor en punto flotante a la potencia indicada por un exponente que es un número entero positivo. Para ello, en el programa principal, se solicitará el ingreso de la base (un número en punto flotante) y del exponente (un número entero sin signo) y se deberá utilizar la subrutina A\_LA\_POTENCIA para calcular el resultado pedido (que será un valor en punto flotante). Tener en cuenta que cualquier base elevada a la 0 da como resultado 1. Mostrar el resultado numérico de la operación en la salida estándar del simulador (ventana Terminal).*

```
.data
CONTROL: .word 0x10000
DATA: .word 0x10008
BASE: .word 0.0
EXP: .word 0
RES: .double 0
UNO: .double 1.0
MSJ1: .asciiz "BASE en Flotante: \n"
MSJ2: .asciiz "EXPONENTE en BSS: \n"
MSJ3: .asciiz "RESULTADO:\n"

.code
ld $s0, CONTROL($0)
ld $s1, DATA($0)
daddi $t1, $0, MSJ1
sd $t1, 0($s1)
daddi $t0, $0, 4
sd $t0, 0($s0)
daddi $t0, $0, 8
sd $t0, 0($s0)
l.d f1, 0($s1)
s.d f1, BASE($0)
daddi $t1, $0, MSJ2
sd $t1, 0($s1)
daddi $t0, $0, 4
sd $t0, 0($s0)
daddi $t0, $0, 8
sd $t0, 0($s0)
ld $a0, 0($s1)
sd $a0, EXP($0)
jal A_LA_POTENCIA
s.d f2, RES($0)
daddi $t1, $0, MSJ3
sd $t1, 0($s1)
daddi $t0, $0, 4
sd $t0, 0($s0)
s.d f2, 0($s1)
daddi $t0, $0, 3
```

sd \$t0, 0(\$s0)  
halt

A\_LA\_POTENCIA: l.d f2, UNO(\$0)  
dadd \$t0, \$a0, \$0  
LAZO: beqz \$t0, FIN  
mul.d f2, f2, f1  
daddi \$t0, \$t0, -1  
j LAZO  
FIN: jr \$ra

## Ejercicio 6.

El siguiente programa produce una salida estableciendo el color de un punto de la pantalla gráfica (en la ventana Terminal del simulador WinMIPS64). Modificar el programa de modo que las coordenadas y color del punto sean ingresados por teclado.

```

.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: .word32 0x10000
DATA: .word32 0x10008

.code
lwu $s6, CONTROL($zero) ; $s6 = dirección de CONTROL
lwu $s7, DATA($zero) ; $s7 = dirección de DATA
daddi $t0, $zero, 7 ; $t0 = 7 -> función 7: limpiar pantalla
gráfica
sd $t0, 0($s6) ; CONTROL recibe 7 y limpia la pantalla
gráfica
lbu $s0, COORX($zero) ; $s0 = valor de coordenada X
sb $s0, 5($s7) ; DATA+5 recibe el valor de coordenada X
lbu $s1, COORY($zero) ; $s1 = valor de coordenada Y
sb $s1, 4($s7) ; DATA+4 recibe el valor de coordenada Y
lwu $s2, COLOR($zero) ; $s2 = valor de color a pintar
sw $s2, 0($s7) ; DATA recibe el valor del color a pintar
daddi $t0, $zero, 5 ; $t0 = 5 -> función 5: salida gráfica
sd $t0, 0($s6) ; CONTROL recibe 5 y produce el dibujo del
punto
halt

```

```

.data
CONTROL: .word 0x10000
DATA: .word 0x10008
COORX: .byte 24
COORY: .byte 24
COLOR: .byte 255, 0, 255, 0

.code
ld $s0, CONTROL($0)
ld $s1, DATA($0)
daddi $t0, $0, 7
sd $t0, 0($s0)
lbu $t0, COORX($0)
sb $t0, 5($s1)
lbu $t1, COORY($0)
sb $t1, 4($s1)
lwu $t2, COLOR($0)

```

```
sw $t2, 0($s1)
daddi $t0, $0, 5
sd $t0, 0($s0)
halt

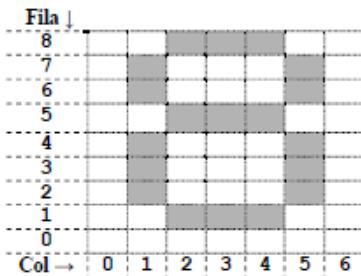
.data
CONTROL: .word 0x10000
DATA: .word 0x10008
COORX: .byte 0
COORY: .byte 0
COLOR: .byte 0, 0, 0, 0
MSJ1: .asciiz "INTRODUCIR COORDENADA X:\n"
MSJ2: .asciiz "INTRODUCIR COORDENADA Y:\n"
MSJ3: .asciiz "INTRODUCIR RGBA:\n"

.code
ld $s0, CONTROL($0)
ld $s1, DATA($0)
daddi $t1, $0, MSJ1
sd $t1, 0($s1)
daddi $t0, $0, 4
sd $t0, 0($s0)
daddi $t0, $0, 8
sb $t0, 0($s0)
lbu $t2, 0($s1)
sb $t2, COORX($0)
daddi $t1, $0, MSJ2
sd $t1, 0($s1)
daddi $t0, $0, 4
sd $t0, 0($s0)
daddi $t0, $0, 8
sb $t0, 0($s0)
lbu $t2, 0($s1)
sb $t2, COORY($0)
daddi $t1, $0, MSJ3
sd $t1, 0($s1)
daddi $t0, $0, 4
sd $t0, 0($s0)
daddi $a0, $0, 0
daddi $a1, $0, 4
LAZO: daddi $t0, $0, 8
sb $t0, 0($s0)
lbu $t2, 0($s1)
sb $t2, COLOR($a0)
daddi $a1, $a1, -1
daddi $a0, $a0, 1
bnez a1, LAZO
daddi $t0, $0, 7
sd $t0, 0($s0)
lbu $t0, COORX($0)
```

```
sb $t0, 5($s1)
lbu $t1, COORY($0)
sb $t1, 4($s1)
lwu $t2, COLOR($0)
sw $t2, 0($s1)
daddi $t0, $0, 5
sd $t0, 0($s0)
halt
```

### Ejercicio 7.

Se desea realizar la demostración de la transformación de un carácter codificado en ASCII a su visualización en una matriz de puntos con 7 columnas y 9 filas. Escribir un programa que realice tal demostración, solicitando el ingreso por teclado de un carácter para, luego, mostrarlo en la pantalla gráfica de la terminal. El carácter “8” es representado como:



### **Ejercicio 8.**

*El siguiente programa implementa una animación de una pelotita rebotando por la pantalla. Modificarlo para que, en lugar de una pelotita, se muestren, simultáneamente, varias pelotitas (cinco, por ejemplo), cada una con su posición, dirección y color particular.*

## **Ejercicio 9.**

*Escribir un programa que le permita dibujar en la pantalla gráfica de la terminal. Deberá mostrar un cursor (representado por un punto de un color particular) que pueda desplazarse por la pantalla usando las teclas “a”, “s”, “d” y “w” para ir a la izquierda, abajo, a la derecha y arriba, respectivamente. Usando la barra espaciadora, se alternará entre modo desplazamiento (el cursor pasa por arriba de lo dibujado sin alterarlo) y modo dibujo (cada punto por el que el cursor pasa quedará pintado del color seleccionado). Las teclas del “1” al “8” se usarán para elegir uno entre los ocho colores disponibles para pintar.*

*Observaciones: Para poder implementar este programa, se necesitará almacenar, en la memoria, la imagen completa de la pantalla gráfica. Si cada punto está representado por un byte, se necesitarán  $50 \times 50 \times 1 = 2500$  bytes. El simulador WinMIPS64 viene configurado para usar un bus de datos de 10 bits, por lo que la memoria disponible estará acotada a  $2^{10} = 1024$  bytes. Para poder almacenar la imagen, será necesario configurar el simulador para usar un bus de datos de 12 bits, ya que  $2^{12} = 4096$  bytes, los que si resultarán suficientes. La configuración se logra yendo al menú “Configure → Architecture” y poniendo “Data Address Bus” en 12 bits, en lugar de los 10 bits que trae por defecto.*