Práctica 4: Introducción al WinMIPS64

Parte 1: Introducción al set de instrucciones del WinMIPS64

Ejercicio 1

Instrucciones de salto incondicional (SI), salto condicional (SC), lectura de memoria (LMEM), escritura de memoria (EMEM) y aritmético-lógicas (AL) del WinMIPS64:

Instrucciones del WinMIPS64					
and (AL)	bnez (SC)	halt (XXX)	slt (AL)	ld (LMEM)	sb (EMEM)
andi (AL)	dadd (AL)	nop (XXX)	slti (AL)	lb (LMEM)	sd (EMEM)
beq (SC)	daddi (AL)	or (AL)	j (SI)	lbu (LMEM)	sw (EMEM)
bne (SC)	dmul (AL)	ori (AL)	jal (SI)	lw (LMEM)	xor (AL)
beqz (SC)	ddiv (AL)		jr (SI)	lwu (LMEM)	xori (AL)

Ejercicio 2

Equivalencias y correspondencias entre las instrucciones soportadas por los simuladores VonSim y WinMIPS64:

VonSim	**WinMIPS64**
`mov r1, r2`	`daddi r1, r2, 0` `dadd r1, r0, r2`
`mov r1, 1`	`daddi r1, r0, 1`
`mov r1, 0`	`daddi r1, r0, 0` `dadd r1, r0, r0`
`add r1, r2`	`dadd r1, r1, r2`
`add r1, 1` `inc r1`	`daddi r1, r1, 1`
`add r1, 0`	`dadd r1, r1, r0`
`dec r1`	`daddi r1, r1, -1`
`or r1, r2`	`or r1, r1, r2`
`or r1, 1`	`ori r1, r1, 1`
`or r1, 0`	`or r1, r1, r0`
`mov r1, variable`	`ld r1, variable(r0)`
`mov variable, r1`	`sd r1, variable(r0)`
`add r1, variable`	
`add variable, r1`	
`mov r1, offset variable`	`daddi r1, r0, variable`
`jump etiqueta`	`j etiqueta`
`call etiqueta`	`jal etiqueta`
`hlt`	`halt`

Ejercicio 3

Tipos de datos del simulador WinMIPS64:

Tipo de dato	Tamaño en bytes	Uso
`space 1`	1 (podría ser cualquier número N)	Deja un byte vacío (N bytes vacíos si se utilizara otro número N)
`ascii`	1 por carácter	Define una cadena de caracteres ASCII
`asciiz`	1 por carácter y 1 adicional por el 0 final	Define una cadena de caracteres ASCII terminada en 0 (no confundir con el carácter '0')
`byte`	1	Define uno o múltiples bytes (por ejemplo, número/s de 8 bits)
`word16`	2	Define uno o múltiples números de 16 bits
`word32`	4	Define uno o múltiples números de 32 bits
`word`	8	Define uno o múltiples números de 64 bits
`double`	8	Define números de 64 bits representados en punto flotante (IEEE 754)

Ejercicio 4

El procesador MIPS64 posee 32 registros enteros, de 64 bits cada uno, llamados r0 a r31 (también denominados $0 a $31). Sin embargo, resulta más conveniente para los programadores asignarles nombres más significativos a esos registros.

La siguiente tabla muestra la convención empleada para nombrar a los 32 registros previamente mencionados:

Registros	Nombres	Usos	Preservado
`r0`	`$zero`	Siempre tiene el valor 0 (cero) y no se puede cambiar	
`r1`	`$at`	*Assembler Temporary* – Reservado para ser usado por el ensamblador	
`r2-r3`	`$v0-$v1`	Valores de retorno de la subrutina llamada	
`r4-r7`	`$a0-$a3`	Argumentos pasados a la subrutina llamada	
`r8-r15`	`$t0-$t7`	Registros temporarios – No son conservados en el llamado a subrutinas	
`r16-r23`	`$s0-$s7`	Registros conservados durante el llamado a subrutinas	X
`r24-r25`	`$t8-$t9`	Registros temporarios – No son conservados en el llamado a subrutinas	
`r26-r27`	`$k0-$k1`	Para uso del kernel del sistema operativo	
`r28`	`$gp`	*Global Pointer* – Puntero a la zona de la memoria estática del programa	X
`r29`	`$sp`	*Stack Pointer* – Puntero al tope de la pila	X
`r30`	`$fp`	*Frame Pointer* – Puntero al marco actual de la pila	X
`r31`	`$ra`	*Return Address* – Dirección de retorno en un llamado a una subrutina	X

La tabla anterior establece una convención a seguir al momento de escribir subrutinas, de forma que cualquier subrutina escrita por un programador (o generada por un compilador de algún lenguaje de más alto nivel) pueda usarse junto con otras escritas por otros. Los registros marcados con una “X” en la última columna de la tabla deben ser preservados durante la invocación a una subrutina. Esto quiere decir que, si la subrutina los va a utilizar, tiene que

asegurarse de salvar sus valores antes de alterarlos para así poder restaurarlos antes de retornar (de ahí los nombres $s0-$s7, la "s" es de ***S**aved Register*).
Los registros $s0 al $s7 son usados normalmente para cumplir el rol de variables locales a la subrutina: existen desde el principio al final de la misma y su valor es preservado durante invocaciones a otras subrutinas. Al contrario, los registros $t0 al $t9 son utilizados para almacenar valores temporarios y probablemente sean modificados al invocar a otras subrutinas. A pesar de esto, resultan útiles para contener valores auxiliares dentro de cálculos complicados.

Ejercicio 5

a)

```
            .data
A:          .word 4
B:          .word 7
S:          .word 0    ; aquí no es válido el valor desconocido "?"
P:          .word 0
D:          .word 0

            .code
            ld $t0, A($0)
            ld $t1, B($0)
            dadd $t2, $t0, $t1
            dmul $t3, $t0, $t1
            daddi $t3, $t3, 2
            dmul $t4, $t0, $t0
            ddiv $t4, $t4, $t1
            sd $t2, S($0)
            sd $t3, P($0)
            sd $t4, D($0)
            halt
```

b)

```
            .data
A:          .word 7
B:          .word 4
C:          .word 0

            .code
            ld $t0, A($0)
            ld $t1, B($0)
            bnez $t0, ALT_2_3
ALT_1:      daddi $t2, $0, 0
            j FIN_PROG
ALT_2_3:    slt $t2, $t1, $t0     ; $t2 será igual a uno o cero si
                                  ; se cumple o no respectivamente la
                                  ; condición de salto ($t1 < $t0)
            beqz $t2, ALT_3
ALT_2:      dadd $t2, $t0, $t0
            J FIN_PROG
ALT_3:      dadd $t2, $t1, $0
FIN_PROG:   sd $t2, C($0)
            halt
```

d)

```
            .data
N:          .word 8                 ; debe ser un valor positivo
L:          .word 0

            .code
            ld $t0, N($0)
            daddi $t1, $0, 0
            daddi $t2, $0, 2
            daddi $t3, $0, 1
BUCLE:      ddiv $t0, $t0, $t2
            daddi $t1, $t1, 1
            bne $t0, $t3, BUCLE     ; repetir el bucle mientras N
                                    ; sea mayor a 1 (uno)
            sd $t1, L($0)
            halt
```

e)

```
            .data
A:          .word 7
B:          .word 0

            .code
            ld $t0, A($0)
            andi $t0, $t0, 1
            beqz $t0, ES_PAR
ES_IMPAR:   daddi $t0, $0, 1
            j FIN_PROG
ES_PAR:     daddi $t0, $0, 0
FIN_PROG:   sd $t0, B($0)
            halt
```

Ejercicio 6

a)

```
            .data
V:          .word 5, 2, 6
S:          .word 0

            .code
            daddi $t0, $0, 0
            daddi $t2, $0, 0
            ld $t1, V($t0)
            dadd $t2, $t2, $t1
            daddi $t0, $t0, 8
            ld $t1, V($t0)
            dadd $t2, $t2, $t1
            daddi $t0, $t0, 8
            ld $t1, V($t0)
            dadd $t2, $t2, $t1
            sd $t2, S($0)
            halt
```

b)

```
            .data
V:          .word 5, 2, 6
S:          .word 0

            .code
            daddi $t0, $0, 0
            daddi $t1, $0, 0
            daddi $t2, $0, 3
BUCLE:      ld $t3, V($t1)
            dadd $t0, $t0, $t3
            daddi $t1, $t1, 8
            daddi $t2, $t2, -1
            bnez $t2, BUCLE
            sd $t0, S($0)
            halt
```

c)

```
            .data
V:          .word 5, 2, 6
S:          .word 0

            .code
            daddi $t0, $0, 0
            daddi $t1, $0, V
            daddi $t2, $0, 3
BUCLE:      ld $t3, 0($t1)
            dadd $t0, $t0, $t3
            daddi $t1, $t1, 8
            daddi $t2, $t2, -1
            bnez $t2, BUCLE
            sd $t0, S($0)
            halt
```

d)

```
            .data
V:          .word32 5, 2, 6
S:          .word32 0

            .code
            daddi $t0, $0, 0
            daddi $t1, $0, 0
            daddi $t2, $0, 3
BUCLE:      lw $t3, V($t1)
            dadd $t0, $t0, $t3
            daddi $t1, $t1, 4
            daddi $t2, $t2, -1
            bnez $t2, BUCLE
            sw $t0, S($0)
            halt
```

Ejercicio 7

a)

Contar positivos:

```
            .data
V:          .word 5, -2, -6, 4, -7, 24, -25, 13, 12, 18
POS:        .word 0

            .code
            daddi $t0, $0, 0
            daddi $t1, $0, 0
            daddi $t2, $0, 10
BUCLE:      ld $t3, V($t1)
            slti $t3, $t3, 1      ; positivos: mayores a cero
            bnez $t3, SIG_NUM
            daddi $t0, $t0, 1
SIG_NUM:    daddi $t1, $t1, 8
            daddi $t2, $t2, -1
            bnez $t2, BUCLE
            sd $t0, POS($0)
            halt
```

Calcular máximo:

```
            .data
V:          .word 5, -2, -6, 4, -7, 24, -25, 13, 12, 18
MAX:        .word 0

            .code
            ld $t0, V($0)
            daddi $t1, $0, 8
            daddi $t2, $0, 9
BUCLE:      ld $t3, V($t1)
            slt $t4, $t0, $t3
            beqz $t4, SIG_NUM
            daddi $t0, $t3, 0
SIG_NUM:    daddi $t1, $t1, 8
            daddi $t2, $t2, -1
            bnez $t2, BUCLE
            sd $t0, MAX($0)
            halt
```

Modificar valores:

```
            .data
V:          .word 5, -2, -6, 4, -7, 24, -25, 13, 12, 18

            .code
            daddi $t1, $0, 0
            daddi $t2, $0, 10
BUCLE:      ld $t0, V($t1)
            dadd $t0, $t0, $t0
            sd $t0, V($t1)
            daddi $t1, $t1, 8
            daddi $t2, $t2, -1
            bnez $t2, BUCLE
            halt
```

b)

```
            .data
V:          .word 0

            .code
            daddi $t0, $0, 1
            daddi $t1, $0, 0
            daddi $t2, $0, 10
BUCLE:      sd $t0, V($t1)
            daddi $t0, $t0, 2
            daddi $t1, $t1, 8
            daddi $t2, $t2, -1
            bnez $t2, BUCLE
            halt
```

d)

```
            .data
V:          .word 5, -2, -6, 4, -7, 24, -25, 13, 12, 18
W:          .word 0

            .code
            daddi $t1, $0, 0
            daddi $t2, $0, 10
            daddi $t3, $0, 0
BUCLE:      ld $t0, V($t1)
            andi $t4, $t0, 1
            beqz $t4, SIG_NUM
            sd $t0, W($t3)
            daddi $t3, $t3, 8
SIG_NUM:    daddi $t1, $t1, 8
            daddi $t2, $t2, -1
            bnez $t2, BUCLE
            halt
```

Ejercicio 8

a)

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

Memoria de Datos

```
0000  0031666564434241 cadena: .asciiz "ABCdef1"
0008  3131666564434241 cadena2: .ascii "ABCdef11"
0010  3131666564434241 cadena3: .asciiz "ABCdef1111111"
0018  0000003131313131
0020  0000000000000005 num: .word 5
0028  0000000000000000
0030  0000000000000000
0038  0000000000000000
0040  0000000000000000
0048  0000000000000000
0050  0000000000000000
```

b)

```
            .data
datos:      .byte -2, 2, 2, 2, 2, 2

            .code
            ld $t1, datos($zero)
            lb $t2, datos($zero)
            lbu $t3, datos($zero)
            halt
```

Memoria de Datos

```
0000  00000202020202fe datos: .byte -2, 2, 2, 2, 2, 2
0008  0000000000000000
0010  0000000000000000
```

Registros

```
$0=  0000000000000000
$at= 0000000000000000
$v0= 0000000000000000
$v1= 0000000000000000
$a0= 0000000000000000
$a1= 0000000000000000
$a2= 0000000000000000
$a3= 0000000000000000
$t0= 0000000000000000
$t1= 00000202020202fe
$t2= fffffffffffffffe
$t3= 00000000000000fe
$t4= 0000000000000000
$t5= 0000000000000000
```

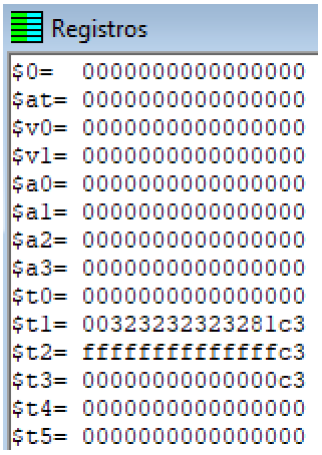
datos($zero) = -2_{10} = FE_{16}
$t1 = $00000202020202FE_{16}$ = $2.207.646.876.414_{10}$ → Valor incorrecto
$t2 = $FFFFFFFFFFFFFFFE_{16}$ = -2_{10} → Valor correcto
$t3 = $00000000000000FE_{16}$ = 254_{10} → Valor incorrecto

```
            .data
datos:      .ascii "Ã22222"       ; para probar en el simulador,
                                  ; escribir la cadena "Á22222"

            .code
            ld $t1, datos($zero)
            lb $t2, datos($zero)
            lbu $t3, datos($zero)
            halt
```

Memoria de Datos

```
0000  00323232323281c3 datos:   .ascii "Ã22222"
0008  0000000000000000
0010  0000000000000000
```



datos($zero) = 'Ã' = $C3_{16}$
$t1 = 00323232323281$C3_{16}$ = "ÃX22222" → Valor incorrecto
$t2 = FFFFFFFFFFFFFFC3$_{16}$ = "Ãÿÿÿÿÿÿÿ" → Valor incorrecto
$t3 = 00000000000000$C3_{16}$ = 'Ã' → Valor correcto

Ejercicio 9

a)

```
            .data
CADENA:     .asciiz "ArquiTectuRa de ComPutaDoras"
LONGITUD:   .word 0

            .code
            daddi $t0, $0, 0
BUCLE:      lbu $t1, CADENA($t0)
            beqz $t1, FIN_PROG
            daddi $t0, $t0, 1
            j BUCLE
FIN_PROG:   sd $t0, LONGITUD($0)
            halt
```

c)

```
            .data
CADENA:     .asciiz "ArquiTectuRa de ComPutaDoras"
C_MAYUS:    .word 0

            .code
            daddi $t0, $0, 0
            daddi $t1, $0, 0
BUCLE:      lbu $t2, CADENA($t1)
            beqz $t2, FIN_PROG
            slti $t3, $t2, 0x41   ; 41₁₆ = 65₁₀ = 'A'
            bnez $t3, SIG_CAR
            slti $t3, $t2, 0x5B   ; 5A₁₆ = 90₁₀ = 'Z'
            beqz $t3, SIG_CAR
ES_MAYUS:   daddi $t0, $t0, 1
SIG_CAR:    daddi $t1, $t1, 1
            j BUCLE
FIN_PROG:   sd $t0, C_MAYUS($0)
```

```
            halt
```

d)

```
            .data
CADENA:     .ascii ""

            .code
            daddi $t0, $0, 0x61   ; 61₁₆ = 'a'
            daddi $t1, $0, 0
            daddi $t2, $0, 1
            daddi $t3, $t2, 0
BUCLE:      sb $t0, CADENA($t1)
            daddi $t1, $t1, 1
            daddi $t3, $t3, -1
            bnez $t3, BUCLE
            daddi $t0, $t0, 1
            slti $t3, $t0, 0x69   ; 68₁₆ = 'h'
            beqz $t3, FIN_PROG
            daddi $t2, $t2, 1
            daddi $t3, $t2, 0
            j BUCLE
FIN_PROG:   sb $0, CADENA($t1)
            halt
```

Parte 2: Entrada/Salida

Ejercicio 1

a)

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

            .code
            ld $t0, CONTROL($0)   ; $t0 = dirección de CONTROL
            ld $t1, DATA($0)      ; $t1 = dirección de DATA
            daddi $t2, $0, TEXTO  ; $t2 = dirección del mensaje a
                                  ; mostrar
            sd $t2, 0($t1)        ; DATA recibe el puntero al comienzo
                                  ; del mensaje
            daddi $t2, $0, 4      ; $t2 = 4 → función 4: salida de una
                                  ; cadena ASCII
            sd $t2, 0($t0)        ; CONTROL recibe 4 y produce la salida
                                  ; del mensaje
            halt
```

Memoria de Datos

```
0000  754d202c616c6f48 TEXTO:   .asciiz "Hola, Mundo!"
0008  000000006f6f646e
0010  0000000000010000 CONTROL:   .word 0x10000
0018  0000000000010008 DATA:   .word 0x10008
0020  0000000000000000
0028  0000000000000000
0030  0000000000000000
```

Registros

```
$0=  0000000000000000
$at= 0000000000000000
$v0= 0000000000000000
$v1= 0000000000000000
$a0= 0000000000000000
$a1= 0000000000000000
$a2= 0000000000000000
$a3= 0000000000000000
$t0= 0000000000010000
$t1= 0000000000010008
$t2= 0000000000000004
$t3= 0000000000000000
$t4= 0000000000000000
```

$t0 = 0x10000 = 10000_{16} (dirección de memoria del registro CONTROL)
$t1 = 0x10008 = 10008_{16} (dirección de memoria del registro DATA)

b)

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

            .code
            ld $t0, CONTROL($0)   ; $t0 = dirección de CONTROL
            ld $t1, DATA($0)      ; $t1 = dirección de DATA
            daddi $t2, $0, TEXTO  ; $t2 = dirección del mensaje a
                                  ; mostrar
            sd $t2, 0($t1)        ; DATA recibe el puntero al comienzo
                                  ; del mensaje
            daddi $t2, $0, 4      ; $t2 = 4 → función 4: salida de una
                                  ; cadena ASCII
            daddi $t3, $0, 10     ; $t3 = cantidad de repeticiones de
                                  ; la impresión del mensaje en pantalla
BUCLE:      sd $t2, 0($t0)        ; CONTROL recibe 4 y produce la salida
                                  ; del mensaje
            daddi $t3, $t3, -1
            bnez $t3, BUCLE
            halt
```

Ejercicio 2

b)

```
            .data
control:    .word32 0x10000
data:       .word32 0x10008
msj_ing:    .asciiz "Ingrese la clave de cuatro caracteres: "
msj_ast:    .asciiz "*"
msj_lf:     .asciiz "\n"
clave:      .asciiz "T!9;"
usu_clave: .asciiz "    "
msj_exito: .asciiz "Clave correcta: acceso permitido. Bienvenido!"
msj_error: .asciiz "Error. Por favor, vuelva a intentarlo.\n"

            .code
            lwu $t0, control($0)
            lwu $t1, data($0)
            daddi $t2, $0, 6
            sd $t2, 0($t0)              ; limpiar la pantalla
lazo_main: daddi $t2, $0, 4             ; bucle principal del programa
            daddi $t3, $0, msj_ing
            sd $t3, 0($t1)
            sd $t2, 0($t0)              ; imprimir "msj_ing" en pantalla
            daddi $t2, $0, 0
ing_car:    daddi $t3, $0, 9
            sd $t3, 0($t0)              ; esperar el ingreso de un nuevo
                                        ; carácter en la pantalla
            lbu $t3, 0($t1)             ; cargar el carácter en $t3
            sb $t3, usu_clave($t2)      ; actualizar "usu_clave" con el
                                        ; carácter leído
            daddi $t3, $0, 4
            daddi $t4, $0, msj_ast
            sd $t4, 0($t1)
            sd $t3, 0($t0)              ; imprimir "msj_ast" en pantalla
            daddi $t2, $t2, 1
            slti $t3, $t2, 4
            bnez $t3, ing_car           ; cuatro iteraciones en total:
                                        ; una por cada carácter a leer
            daddi $t2, $0, 4
            daddi $t3, $0, msj_lf
            sd $t3, 0($t1)
            sd $t2, 0($t0)              ; imprimir "msj_lf" en pantalla
                                        ; (salto de línea)
            daddi $t2, $0, 0
comparar:  lbu $t3, clave($t2)          ; comparar "usu_clave" con
                                        ; "clave" carácter por carácter
            beqz $t3, es_exito          ; comprobar si ya ha terminado
                                        ; la comparación
            lbu $t4, usu_clave($t2)
            bne $t3, $t4, es_error
            daddi $t2, $t2, 1
            j comparar
es_error:  daddi $t2, $0, 4             ; las claves son distintas
            daddi $t3, $0, msj_error
            sd $t3, 0($t1)
```

```
            sd $t2, 0($t0)              ; imprimir "msj_error" en
                                        ; pantalla
            j lazo_main                 ; repetir el bucle principal
es_exito:   daddi $t2, $0, 4            ; las claves son iguales
            daddi $t3, $0, msj_exito
            sd $t3, 0($t1)
            sd $t2, 0($t0)              ; imprimir "msj_exito" en
                                        ; pantalla
            halt
```

c)

```
            .data
control:    .word32 0x10000
data:       .word32 0x10008
msj_ing_1: .asciiz "Ingrese la clave de cuatro caracteres ("
msj_ing_2: .asciiz "X intento/s restante/s): "
msj_ast:    .asciiz "*"
msj_lf:     .asciiz "\n"
clave:      .asciiz "T!9;"
usu_clave: .asciiz "    "
msj_exito: .asciiz "Clave correcta: acceso permitido. Bienvenido!"
msj_fallo: .asciiz "Clave incorrecta: acceso denegado."
msj_error: .asciiz "Error. Por favor, vuelva a intentarlo.\n"

            .code
            lwu $t0, control($0)
            lwu $t1, data($0)
            daddi $t5, $0, 0x35         ; cantidad de intentos restantes
                                        ; 0x35 = 35_16 = '5'
            daddi $t2, $0, 6
            sd $t2, 0($t0)              ; limpiar la pantalla
lazo_main: daddi $t2, $0, 4             ; bucle principal del programa
            daddi $t3, $0, msj_ing_1
            sd $t3, 0($t1)
            sd $t2, 0($t0)              ; imprimir "msj_ing_1" en
                                        ; pantalla
            sb $t5, msj_ing_2($0)       ; actualizar en "msj_ing_2" la
                                        ; cantidad de intentos restantes
            daddi $t3, $0, msj_ing_2
            sd $t3, 0($t1)
            sd $t2, 0($t0)              ; imprimir "msj_ing_2" en
                                        ; pantalla
            daddi $t2, $0, 0
ing_car:    daddi $t3, $0, 9
            sd $t3, 0($t0)              ; esperar el ingreso de un nuevo
                                        ; carácter en la pantalla
            lbu $t3, 0($t1)             ; cargar el carácter en $t3
            sb $t3, usu_clave($t2)      ; actualizar "usu_clave" con el
                                        ; carácter leído
            daddi $t3, $0, 4
            daddi $t4, $0, msj_ast
            sd $t4, 0($t1)
            sd $t3, 0($t0)              ; imprimir "msj_ast" en pantalla
            daddi $t2, $t2, 1
            slti $t3, $t2, 4
```

```
            bnez $t3, ing_car           ; cuatro iteraciones en total:
                                        ; una por cada carácter a leer
            daddi $t2, $0, 4
            daddi $t3, $0, msj_lf
            sd $t3, 0($t1)
            sd $t2, 0($t0)              ; imprimir "msj_lf" en pantalla
                                        ; (salto de línea)
            daddi $t2, $0, 0
comparar:   lbu $t3, clave($t2)         ; comparar "usu_clave" con
                                        ; "clave" carácter por carácter
            beqz $t3, es_exito          ; comprobar si ya ha terminado
                                        ; la comparación
            lbu $t4, usu_clave($t2)
            bne $t3, $t4, es_error
            daddi $t2, $t2, 1
            j comparar
es_error:   daddi $t5, $t5, -1          ; las claves son distintas
                                        ; (queda un intento menos)
            slti $t2, $t5, 0x31         ; 0x30 = 30₁₆ = '0'
            bnez $t2, es_fallo
            daddi $t2, $0, 4
            daddi $t3, $0, msj_error
            sd $t3, 0($t1)
            sd $t2, 0($t0)              ; imprimir "msj_error" en
                                        ; pantalla
            j lazo_main                 ; repetir el bucle principal
es_fallo:   daddi $t3, $0, msj_fallo    ; clave incorrecta (no quedan
                                        ; más intentos)
            j imp_fin
es_exito:   daddi $t3, $0, msj_exito    ; las claves son iguales
imp_fin:    daddi $t2, $0, 4
            sd $t3, 0($t1)
            sd $t2, 0($t0)              ; imprimir "msj_exito" o
                                        ; "msj_fallo" en pantalla
            halt
```

Ejercicio 3

b)

El presente programa realiza una tarea ligeramente diferente a aquélla solicitada en la consigna ya que lee en primer lugar un operando, luego el operador ('+', '-', '*' o '/') y finalmente el segundo operando, imprimiendo cada uno de ellos en pantalla conforme es ingresado. A continuación, realiza la operación aritmética y la informa en forma completa junto con el resultado obtenido.

```
            .data
control:    .word32 0x10000
data:       .word32 0x10008
msj_ing_1: .asciiz "Ingrese el primer operando (entero): "
msj_ing_2: .asciiz "Ingrese el operador: "
msj_ing_3: .asciiz "Ingrese el segundo operando (entero): "
msj_oper:  .asciiz "X"
msj_igu:   .asciiz "----"
msj_lf:    .asciiz "\n"
```

```
            .code
            lwu $t0, control($0)
            lwu $t1, data($0)
            daddi $t2, $0, 6
            sd $t2, 0($t0)
            daddi $t2, $0, 4
            daddi $t3, $0, msj_ing_1
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t2, $0, 8
            sd $t2, 0($t0)
            ld $t4, 0($t1)
            daddi $t2, $0, 4
            daddi $t3, $0, msj_ing_2
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t2, $0, 9
            sd $t2, 0($t0)
            lbu $t5, 0($t1)
            sb $t5, msj_oper($0)
            daddi $t2, $0, 4
            daddi $t3, $0, msj_oper
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t3, $0, msj_lf
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t3, $0, msj_ing_3
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t2, $0, 8
            ; precaución: el simulador intenta leer nuevamente el
            ; operador previamente ingresado, así que se lo debe borrar
            ; en la pantalla antes de escribir el segundo operando
            sd $t2, 0($t0)
            ld $t6, 0($t1)
            daddi $t7, $0, 0x2B        ; 0x2B = 2B16 = '+'
            bne $t5, $t7, no_suma
suma:       dadd $t7, $t4, $t6
            j imp_fin
no_suma:    daddi $t7, $0, 0x2D        ; 0x2D = 2D16 = '-'
            bne $t5, $t7, no_resta
resta:      dsub $t7, $t4, $t6
            j imp_fin
no_resta:   daddi $t7, $0, 0x2A        ; 0x2A = 2A16 = '*'
            bne $t5, $t7, divid
multip:     dmul $t7, $t4, $t6
            j imp_fin
divid:      ddiv $t7, $t4, $t6
imp_fin:    daddi $t2, $0, 2
            sd $t4, 0($t1)
            sd $t2, 0($t0)
            daddi $t2, $0, 4
            daddi $t3, $0, msj_oper
            sd $t3, 0($t1)
```

```
            sd $t2, 0($t0)
            daddi $t3, $0, msj_lf
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t2, $0, 2
            sd $t6, 0($t1)
            sd $t2, 0($t0)
            daddi $t2, $0, 4
            daddi $t3, $0, msj_igu
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t3, $0, msj_lf
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t2, $0, 2
            sd $t7, 0($t1)
            sd $t2, 0($t0)
            halt
```

d)

El presente programa realiza una tarea ligeramente diferente a aquélla solicitada en la consigna ya que no solo imprime en pantalla la superficie del triángulo, sino también la operación aritmética completa realizada para calcularla.

```
            .data
control:    .word32 0x10000
data:       .word32 0x10008
msj_ing_1: .asciiz "Ingrese el valor de la base (entero): "
msj_ing_2: .asciiz "Ingrese el valor de la altura (entero): "
msj_calc:  .asciiz "Calculo de la superficie del triangulo:"
msj_op_1:  .asciiz "*"
msj_op_2:  .asciiz "/"
msj_igu:   .asciiz "----"
msj_lf:    .asciiz "\n"

            .code
            lwu $t0, control($0)
            lwu $t1, data($0)
            daddi $t2, $0, 6
            sd $t2, 0($t0)
            daddi $t2, $0, 4
            daddi $t3, $0, msj_ing_1
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t2, $0, 8
            sd $t2, 0($t0)
            ld $t4, 0($t1)
            daddi $t2, $0, 4
            daddi $t3, $0, msj_ing_2
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t2, $0, 8
            sd $t2, 0($t0)
            ld $t5, 0($t1)
```

```
        daddi $t2, $0, 4
        daddi $t3, $0, msj_calc
        sd $t3, 0($t1)
        sd $t2, 0($t0)
        daddi $t3, $0, msj_lf
        sd $t3, 0($t1)
        sd $t2, 0($t0)
        daddi $t2, $0, 1
        sd $t4, 0($t1)
        sd $t2, 0($t0)
        daddi $t2, $0, 4
        daddi $t3, $0, msj_op_1
        sd $t3, 0($t1)
        sd $t2, 0($t0)
        daddi $t3, $0, msj_lf
        sd $t3, 0($t1)
        sd $t2, 0($t0)
        daddi $t2, $0, 1
        sd $t5, 0($t1)
        sd $t2, 0($t0)
        daddi $t2, $0, 4
        daddi $t3, $0, msj_op_2
        sd $t3, 0($t1)
        sd $t2, 0($t0)
        daddi $t3, $0, msj_lf
        sd $t3, 0($t1)
        sd $t2, 0($t0)
        daddi $t6, $0, 2
        daddi $t2, $0, 1
        sd $t6, 0($t1)
        sd $t2, 0($t0)
        daddi $t2, $0, 4
        daddi $t3, $0, msj_igu
        sd $t3, 0($t1)
        sd $t2, 0($t0)
        daddi $t3, $0, msj_lf
        sd $t3, 0($t1)
        sd $t2, 0($t0)
        dmul $t4, $t4, $t5
        ddiv $t4, $t4, $t6
        daddi $t2, $0, 1
        sd $t4, 0($t1)
        sd $t2, 0($t0)
        halt
```

Ejercicio 4

a)

```
            .data
control:    .word32 0x10000
data:       .word32 0x10008
msj_X:      .asciiz "Ingrese la coordenada X del punto (de 0 a 49): "
msj_Y:      .asciiz "Ingrese la coordenada Y del punto (de 0 a 49): "
coor_X:     .byte 0    ; coordenada X del punto
coor_Y:     .byte 0    ; coordenada Y del punto
```

```
color_RGB: .byte 255, 0, 255, 0        ; color expresado en RGB
                                       ; máximos rojo y azul → magenta

           .code
           lwu $t0, control($0)
           lwu $t1, data($0)
           daddi $t2, $0, 6
           sd $t2, 0($t0)              ; limpiar pantalla alfanumérica
           daddi $t2, $0, 7
           sd $t2, 0($t0)              ; limpiar pantalla gráfica
           daddi $t2, $0, 4
           daddi $t3, $0, msj_X
           sd $t3, 0($t1)
           sd $t2, 0($t0)              ; imprimir "msj_X" en pantalla
                                       ; alfanumérica
           daddi $t2, $0, 8
           sd $t2, 0($t0)              ; leer coordenada X desde
                                       ; pantalla alfanumérica
           lbu $t2, 0($t1)
           sb $t2, coor_X($0)          ; actualizar valor de "coor_X"
           daddi $t2, $0, 4
           daddi $t3, $0, msj_Y
           sd $t3, 0($t1)
           sd $t2, 0($t0)              ; imprimir "msj_Y" en pantalla
                                       ; alfanumérica
           daddi $t2, $0, 8
           sd $t2, 0($t0)              ; leer coordenada Y desde
                                       ; pantalla alfanumérica
           lbu $t2, 0($t1)
           sb $t2, coor_Y($0)          ; actualizar valor de "coor_Y"
           daddi $t2, $0, 6
           sd $t2, 0($t0)              ; limpiar pantalla alfanumérica
           lbu $t2, coor_X($0)         ; $t2 = valor de coordenada X
           sb $t2, 5($t1)              ; DATA + 5 recibe el valor de
                                       ; coordenada X del punto
           lbu $t2, coor_Y($0)         ; $t2 = valor de coordenada Y
           sb $t2, 4($t1)              ; DATA + 4 recibe el valor de
                                       ; coordenada Y punto
           lwu $t2, color_RGB($0)      ; $t2 = valor expresado en RGB
                                       ; del color a pintar (4 bytes)
           sw $t2, 0($t1)              ; DATA recibe el valor RGB del
                                       ; color a pintar (4 bytes)
           daddi $t2, $0, 5            ; función 5: salida gráfica
           sd $t2, 0($t0)              ; pintar el punto en la pantalla
                                       ; gráfica
           halt
```

b)

```
           .data
coorX:     .byte 24                    ; coordenada X de un punto
coorY:     .byte 24                    ; coordenada Y de un punto
colorIni:  .byte 0, 0, 0, 0            ; color inicial: negro (0, 0, 0)
colorFin:  .byte 255, 0, 0, 0          ; color final: rojo (255, 0, 0)
CONTROL:   .word 0x10000
DATA:      .word 0x10008
```

```
            .code
            ld $t0, CONTROL($zero)      ; $t0 = dirección de CONTROL
            ld $t1, DATA($zero)         ; $t1 = dirección de DATA
            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
            lwu $t3, colorIni($zero)    ; $t3 = color inicial a pintar
                                        ; (negro)
            lwu $t4, colorFin($zero)    ; $t4 = color final a pintar
                                        ; (rojo)
bucle:      sw $t3, 0($t1)              ; DATA recibe el valor del color
                                        ; a pintar
            daddi $t2, $zero, 5         ; $t2 = 5 → función 5: salida
                                        ; gráfica
            sd $t2, 0($t0)              ; CONTROL recibe 5 y produce el
                                        ; dibujo del punto
            beq $t3, $t4, finProg       ; si ya se alcanzó a pintar el
                                        ; color final (rojo), terminar
                                        ; el programa
            daddi $t3, $t3, 1           ; actualizar el componente R
                                        ; (rojo) del valor RGB del color
                                        ; a pintar
            j bucle
finProg:    halt
```

c)

```
            .data
coorX:      .byte 0                 ; coordenada X de un punto
coorY:      .byte 0                 ; coordenada Y de un punto
color:      .word32 0xFF0000        ; color azul (0, 0, 255)
CONTROL:    .word32 0x10000
DATA:       .word32 0x10008

            .code
            lwu $t0, CONTROL($0)
            lwu $t1, DATA($0)
            lwu $t2, color($0)
            sw $t2, 0($t1)
            lbu $t2, coorX($zero)
            sb $t2, 5($t1)
            lbu $t3, coorY($zero)
bucle:      sb $t3, 4($t1)
            daddi $t2, $0, 5
            sd $t2, 0($t0)
            daddi $t3, $t3, 1       ; actualizar la coordenada Y para
                                    ; pintar la línea vertical punto por
                                    ; punto hasta alcanzar el extremo
                                    ; superior de la pantalla gráfica (49)
            slti $t2, $t3, 50
            bnez $t2, bucle
            halt
```

Ejercicio 5

a)

```
            .data
X:          .byte 45
Y:          .byte 0
color:      .byte 255, 0, 0, 0    ; color rojo (255, 0, 0)
CONTROL:    .word32 0x10000
DATA:       .word32 0x10008

            .code
            lwu $s0, CONTROL($zero)
            lwu $s1, DATA($zero)
            lwu $t0, color($zero)
            sw $t0, 0($s1)
            lbu $t1, Y($zero)     ; 1° instrucción faltante
            lbu $t2, X($zero)
            daddi $t4, $zero, 50
            daddi $t5, $zero, 5
loop:       sb $t1, 4($s1)
            sb $t2, 5($s1)        ; 2° instrucción faltante
            daddi $t3, $zero, 5
            sd $t3, 0($s0)        ; pintar cuadrado rojo de 5x5 en la
                                  ; esquina inferior derecha de la
                                  ; pantalla gráfica
            daddi $t2, $t2, 1
            bne $t4, $t2, loop
            lbu $t2, X($zero)     ; 3° instrucción faltante
            daddi $t1, $t1, 1
            bne $t5, $t1, loop
            halt
```

b)

```
            .data
X:          .byte 0
Y:          .byte 0
color:      .word32 0xFF00        ; color verde (0, 255, 0)
CONTROL:    .word32 0x10000
DATA:       .word32 0x10008

            .code
            lwu $s0, CONTROL($zero)
            lwu $s1, DATA($zero)
            lwu $t0, color($zero)
            sw $t0, 0($s1)
            lbu $t0, X($zero)
            lbu $t1, Y($zero)
            daddi $t2, $zero, 5
            daddi $t3, $zero, 50
bucle:      sb $t0, 5($s1)
            sb $t1, 4($s1)
            sd $t2, 0($s0)        ; pintar de verde toda la pantalla
                                  ; gráfica
            daddi $t0, $t0, 1
            bne $t0, $t3, bucle
```

```
            lbu $t0, X($zero)
            daddi $t1, $t1, 1
            bne $t1, $t3, bucle
            halt
```

c)

```
            .data
X:          .byte 0
Y:          .byte 0
colores:    .word32 0, 0xFF, 0xFF00, 0xFF0000, 0xFFFF, 0xFF00FF
            .word32 0xFFFF00, 0xFFFFFF
            ; negro, rojo, verde, azul, amarillo, magenta, cian, blanco
CONTROL:    .word32 0x10000
DATA:       .word32 0x10008

            .code
            lwu $s0, CONTROL($zero)
            lwu $s1, DATA($zero)
            lbu $t0, X($zero)
            lbu $t1, Y($zero)
            daddi $t2, $0, 0
            daddi $t3, $zero, 5
            daddi $t4, $zero, 50
            daddi $t5, $zero, 8   ; cantidad de colores restantes
            lwu $t6, colores($t2)
bucle:      sb $t0, 5($s1)
            sb $t1, 4($s1)
            sw $t6, 0($s1)
            sd $t3, 0($s0)        ; pintar cada línea de la pantalla
                                  ; gráfica con un color distinto
            daddi $t0, $t0, 1
            bne $t0, $t4, bucle
            lbu $t0, X($zero)
            daddi $t1, $t1, 1
            daddi $t2, $t2, 4
            daddi $t5, $t5, -1
            bnez $t5, sig_color
            daddi $t2, $0, 0
            daddi $t5, $zero, 8   ; reiniciar colores restantes
sig_color: lwu $t6, colores($t2)
            bne $t1, $t4, bucle
            halt
```