ENTREGA 2 - ESTRUCTURA DE LOS COMPUTADORES

Universitat d'Alacant

Nombre: Iván Soler Sánchez

Grupo: ARA

Fecha: 02/02/2024

ÍNDICE

PRÁCTICA 5

Objetivos

El objetivo principal de esta práctica es entender el funcionamiento de ciertas instrucciones que sirven para romper la ejecución secuencial de los programas y conocer la traducción del lenguaje ensamblador a las estructuras de control básicas de la programación estructurada.

Realizaremos las cuestiones C2, C4 y C10.

C2

Cuestión 2.

- Escribid el programa que lea dos enteros del teclado y escriba en la consola el mayor. La estructura será:

```
        Leer el primer valor (A)
        Leer el segundo valor (B)
        Si (A<B) ir a eti
        Imprimir A
        Ir a acabar:
eti:    Imprimir B
acabar:
```

- Ensambla y prueba el programa con distintos valores de A y B.

```
8   .text
9
10  li $v0, 5
11  syscall
12  move $s1, $v0
13
14  li $v0, 5
15  syscall
16  move $s2, $v0
17
18  slt $t2, $s1, $s2
19  beq $t2, $zero, imprimirA
20
21  j imprimirB
22
23  imprimirA:
24          move $a0, $s1
25          li $v0, 1
26          syscall
27          j final
28
29  imprimirB:
30          move $a0, $s2
31          li $v0, 1
32          syscall
33          j final
34
35  final:
36          li $v0, 10
37          syscall
38
```

En primer lugar estoy leyendo los número y pasandolos a los registros $s1 y $s2, seguidamente, compruebo si $s2 es mayor que $s1, de ser así $t2 pasa a 1 y por ende se imprime A ($s2), de lo contrario se imprime B ($s1).

C4

Cuestión 4.

➢ Observa de nuevo el programa ensamblado que acabas de escribir y rellena los distintos campos en binario de la instrucción *slt* según el formato R. Escribe primero la instrucción a codificar.

Instrucción:

Cod op (6 bits)	Rs (5 bits)	Rt (5 bits)	Rd (5 bits)	Shamt(5 bits)	Funció (6 bits)

`0x0232502a slt $10,$17,$18` Así se codifica en hexadecimal, en binario quedaría de la siguiente manera:
00000000001000110010010100000010
Cod op = 000000, Rs = 00001, Rt = 00011, Rd=, 00100
Shamt = 10100, Funció = 000010

C10

Cuestión 10.

➢ Haz el código que lee de teclado dos valores positivos A y B en los que A<B. El programa tiene que escribir por consola los valores comprendidos entre ambos, incluyéndolos a ellos mismos. Es decir, si A=3 y B=6, escribe en la consola 3 4 5 6 (puedes escribir, por ejemplo, un salto de línea después de cada uno de los valores a mostrar).

```
1  li $v0, 5
2  syscall
3  move $t0, $v0
4
5  li $v0, 5
6  syscall
7  move $t1, $v0
8
9  sle $t3,$t0,$t1
10 beq $t3, $0, final_for
11
12 inicio_for:
13         li $a0, '\n'
14         li $v0, 11
15         syscall
16
17         addi $a0, $t0, 0
18         li $v0, 1
19         syscall
20
21         beq $t0, $t1, final_for
22         addi $t0, $t0, 1
23
24         j inicio_for
25
26 final_for:
27         li $v0, 10
28         syscall
29
30
```

En primer lugar pediremos ambos números por pantalla, seguidamente comprobaremos que el segundo número es mayor que el primero, de no ser así finalizamos el programa, sin embargo, de ser así comenzamos el bucle. En primer lugar guardamos el valor del primer número en $a0 y lo imprimimos, en segundo lugar, comprobamos que no sea igual al segundo número, de no ser así sumamos 1 al primer número, y así sucesivamente hasta que el primer número sea igual que el segundo, una vez suceda esto, daremos por finalizado el bucle.

PRÁCTICA 6

Objetivos

El objetivo principal de esta práctica es conocer cómo definir datos en memoria, conocer las instrucciones que nos permiten leer y escribir en esta y por último, funciones que permiten la entrada y salida de cadenas de caracteres.

Realizaremos las cuestiones C2, C3 y C8

C2

Cuestión 2.

- ¿Cuántos bytes de la memoria principal están ocupados por datos del programa?
- ¿Cuántas instrucciones de acceso a la memoria contiene el programa?

Bkpt	Address	Code	Basic
	0x00400000	0x3c011001	lui $1,0x00001001
	0x00400004	0x34280000	ori $8,$1,0x00000000
	0x00400008	0x3c011001	lui $1,0x00001001
	0x0040000c	0x34290004	ori $9,$1,0x00000004
	0x00400010	0x3c011001	lui $1,0x00001001
	0x00400014	0x342a0008	ori $10,$1,0x00000008
	0x00400018	0x8d100000	lw $16,0x00000000($8)
	0x0040001c	0x8d310000	lw $17,0x00000000($9)
	0x00400020	0x02309020	add $18,$17,$16
	0x00400024	0x02529020	add $18,$18,$18
	0x00400028	0xad520000	sw $18,0x00000000($10)
	0x0040002c	0x2402000a	addiu $2,$0,0x0000000a
	0x00400030	0x0000000c	syscall

Teniendo en cuenta que hemos almacenado 3 palabras, y que cada palabra son 4 bytes, hemos empleado un total de 12 bytes de la memoria principal para el programa.

El programa contiene 3 instrucciones de acceso a la memoria, y son las siguientes:

```
lw $s0,0($t0)
lw $s1,0($t1)
```
```
sw $s2,0($t2)
```

C3

Cuestión 3.

- ¿Qué valor tiene el registro $t1 cuando se ejecuta la instrucción `lw $s1,0($t1)`?
- ¿En qué dirección se almacena el resultado?

Hemos almacenado en $t1 variable B que contiene a su vez .word 10. Dicha instrucción no modifica el valor que hay en $t1, por ende en $t1 tenemos

$t1	9	0x10010004

C8

Cuestión 8

- Modifica el programa de la cuestión 7 para que muestre en la pantalla los datos guardados en la memoria ordenados de menor a mayor valor.

```
.data

M1: .asciiz "Introduce el primer número:\n"
M2: .asciiz "Introduce el segundo número:\n"

.align 2

num1: .space 4
num2: .space 4

.text
la $a0, M1
li $v0, 4
syscall

li $v0, 5
syscall
move $t0, $v0
sw $t0,num1


la $a0, M2
li $v0, 4
syscall

li $v0, 5
syscall
move $t1, $v0
sw $t1,num2
```

En primer lugar, guardamos las dos variables que contienen las cadenas de texto y estamos reservando el espacio en memoria correspondiente a los números. Seguidamente, mostramos los mensajes por pantalla y a su vez pedimos los números. Hago uso de la pseudoinstrucción sw para que lo que haya en $t1 y en $t0 vaya a los huecos que previamente les hemos reservado en memoria.

```
slt $t2, $t0, $t1
beq $t2, $0, mayor2

j mayor1

mayor2:
move $a0, $t1
li $v0, 1
syscall
la $a0, ' '
li $v0, 11
syscall
move $a0, $t0
li $v0, 1
syscall
j final

mayor1:
move $a0, $t0
li $v0, 1
syscall
la $a0, ' '
li $v0, 11
syscall
move $a0, $t1
li $v0, 1
syscall
j final

final:
li $v0, 10
syscall
```

En segundo lugar, estoy viendo si $t1 es mayor que $t0, de ser así, guardo un 1 en $t2, a continuación compruebo si hay en $t2 un 0, de ser así voy a la etiqueta mayor1, de lo contrario voy a la de mayor2. En cada respectiva etiqueta muestro los números ordenados de menor a mayor y por último, finalizamos el programa.

```
Introduce el primer número:
13
Introduce el segundo número:
10
10 13
-- program is finished running --
```

PRÁCTICA 7

Objetivos

El objetivo principal de esta práctica es aprender cómo recorrer y operar vectores y matrices.

Realizaremos las cuestiones C1, C4 y C8

C1

Cuestión 1.

- Modifica el código de la actividad 1 para que muestre por pantalla el mensaje "El número de caracteres de la cadena es: " y a continuación el resultado.

```
.data
str: .ascii "El número de caracteres "
.asciiz "de la cadena es:"

.text
la $s0, str
add $s1, $zero, $zero # Iniciamos contador a 0

loop:
add $t0, $s0, $s1 # dirección del byte a examinar
lb $t1, 0( $t0 )
beq $t1, $zero, exit # salimos si carácter leído='\0'
addi $s1, $s1, 1 # incrementamos el contador

j loop

exit:

move $a0, $s0
li $v0, 4
syscall

move $a0, $s1
li $v0, 1
syscall

li $v0, 10
syscall
```

```
El número de caracteres de la cadena es:40
-- program is finished running --
```

C4

Cuestión 4.

- Modifica el programa de la actividad 2 para que el vector B se rellene con enteros leídos del teclado. Previamente se tiene que mostrar un mensaje por consola que pida los elementos.

```
.data
 A: .word 2, 4, 6, 8, 10 # vector A iniciado con valores
 B: .word 0:4 # Vector B vacío
 C: .space 20 # Otra definición de vector vacio
 str: .asciiz "Introduce un entero: "
 .text
la $s0, A # Dirección base del vector A
la $s1, B # Dirección base del vector B
li $s5, 5 # Tamaño del vector
li $t5, 0 # Entero a rellenar
loop:
        add $t2, $s1, $t0
        addi $s2, $s2, 1

        la $a0, str
        li $v0, 4
        syscall

        la $a0, str
        li $v0, 4
        li $v0, 5
        syscall
        move $t3, $v0
        sw $t3, 0($t2)

        li $v0, 11
        la $a0, '\n'
        syscall

        sll $t0, $s2, 2 # Índice del vector x4
        bne $s2, $s5, loop

li $v0, 10 #Acaba el programa
syscall
```

```
Introduce un entero: 5

Introduce un entero: 6

Introduce un entero: 7

Introduce un entero: 3

Introduce un entero: 9


-- program is finished running --
```

Value (+14)	Value (+18)	Value (+1c)
0x00000005	0x00000006	0x00000007
0x00000003	0x00000009	

En primer lugar, he creado una variable .asciiz con la string a imprimir para pedir que el usuario introduzca el entero. En segundo lugar, pido el entero por pantalla y lo guardo en $t2, registro que contiene la primera posición del vector B, por último, multiplico el índice por 4 para así pasar al siguiente elemento y así sucesivamente.

C8

Cuestión 8.

➢ Escribe el código que calcula la suma de los elementos de la diagonal principal de una matriz 4x4 de valores enteros introducida por teclado. Muestra la suma por pantalla.

```
1   .data
2   vector: .space 64
3   size: .word 16
4   msg: .asciiz "\nIntroduce los val
5
6   .text
7   la $s0, vector
8   la $t4, vector
9   lw $s1, size
10  la $s2, msg
11  li $s3, 4
12
13  loop:
14          la $a0, ($s2)
15          li $v0, 4
16          syscall
17
18          li $v0, 5
19          syscall
20          move $t1, $v0
21
22          sw $t1, 0($t4)
23
24          addi $t0, $t0, 1
25          sll $t2, $t0, 2
26          add $t3, $t2, $s0
27          move $t4, $t3
28
29          bne $s1, $t0, loop
30
```

```
30
31  sumDig:
32          lw $t1, 0($s0)
33          add $t5, $t1, $t5
34          addi $t7, $t7, 1
35          addi $s0, $s0, 20
36          bne $s3, $t7, sumDig
37
38
39  move $a0, $t5
40  li $v0, 1
41  syscall
42
43  li $v0, 10 # Acabar programa
44  syscall
45
46
```

He comenzado guardando el espacio en memoria correspondiente para almacenar la matriz 4x4, así como el tamaño del vector y el mensaje para que el usuario rellene la matriz. Seguidamente, guardamos los valores y las direcciones en los registros para usarlos más tarde. Comenzamos el bucle con el que el usuario va a rellenar la matriz.

Una vez el usuario rellena la matriz procedemos a realizar la suma de la diagonal. Para ellos vamos recorriendo la matriz pero esta vez no recorremos los elementos uno por uno, sino que tenemos que ir saltando directamente a los elementos que forman la diagonal para así sumarlos. Para ello realizaremos los saltos de 20 en 20 en vez de 4 en 4, porque entre 1 elemento y otro de la diagonal hay que cambiar 5 veces de índice, por ello al ser el vector .word (4 bytes), tendremos que saltar de 20 en 20 (5*4 = 20).