

MEMORIA

Índice

1. PRÁCTICA 4 (p.2)
2. PRÁCTICA 5 (p.3-4)
3. PRÁCTICA 6 (p.5-7)
4. Ejercicios optativos (p.8-19)
 - Optativos Práctica 4 (p.8-13)
 - Optativos Práctica 5 (p.14-16)
 - Optativos Práctica 6 (p.17-19)

DATOS

- Nombre: Raúl Beltrán Marco
- Email: rbm61@alu.ua.es/raulbeltmarc@gmail.com
- Grupo: lunes 17/03/19

PRÁCTICA 4

Aritmética de enteros (3) y funciones

Escribe el código que lee el valor x y escribe por pantalla la solución de la ecuación:

$$5x^2 + 2x + 3.$$

Para poder elevar al cuadrado tendremos que hacer uso de la función cuadrado en la que usaremos la función mult para multiplicar por si mismo el número introducido

The screenshot shows the Mars MIPS simulator interface. The main window displays assembly code for a program that reads an integer, calculates $5x^2 + 2x + 3$, and prints the result. The code includes comments in Spanish. The right panel shows the state of the MIPS registers. The bottom panel shows the program's execution output, which includes the input value 5 and the calculated result 138.

```
1 .text
2     li $a0, '>'
3     li $v0, 11
4     syscall
5     li $v0, 5
6     syscall #Leer un entero
7
8     move $a0, $v0 #parámetro a pasar en $a0
9     jal cuadrado #llamamos a la función
10    addi $t0, $0, 5 #Guardamos el valor en $t0
11    mult $v0, $t0 #5x^2
12    mflo $t0 #Mover a $v0
13
14    addi $t1, $0, 2 #Guardamos el valor en $t1
15    mult $a0, $t1
16    mflo $s0 #Guardamos el resultado de la multiplicación en $s0
17
18    add $k0, $t0, $s0 # Hacemos 5x^2+2x
19    add $v0, $k0, 3 #Guardamos en $v0
20
21    move $a0, $v0
22    jal imprim #Llamamos a la función imprim
23    li $v0, 10 #Acaba el programa
24    syscall
25    #-----Funcions-----#
26    imprim:
27        addi $v0, $0, 1 #función imprim
28        syscall #Escribe el valor en $a0
29        li $a0, '\n' #Salto de línea
30        li $v0, 11
31        syscall
32        jr $ra #Vuelve al programa principal
33
34    cuadrado:
35        mult $a0, $a0
36        mflo $v0
37        jr $ra
```

Registers

Na...	Nu...	Value
...	0	0x000...
\$at	1	0x000...
\$v0	2	0x000...
\$v1	3	0x000...
\$a0	4	0x000...
\$a1	5	0x000...
\$a2	6	0x000...
\$a3	7	0x000...
\$t0	8	0x000...
\$t1	9	0x000...
\$t2	10	0x000...
\$t3	11	0x000...
\$t4	12	0x000...
\$t5	13	0x000...
\$t6	14	0x000...
\$t7	15	0x000...
\$s0	16	0x000...
\$s1	17	0x000...
\$s2	18	0x000...
\$s3	19	0x000...
\$s4	20	0x000...
\$s5	21	0x000...
\$s6	22	0x000...
\$s7	23	0x000...
\$s8	24	0x000...
\$s9	25	0x000...
\$k0	26	0x000...
\$k1	27	0x000...
\$gp	28	0x100...
\$sp	29	0x7ff...
\$fp	30	0x000...
\$ra	31	0x004...
pc		0x004...
hi		0x000...
lo		0x000...

Mars Messages Run I/O

```
>5
138
-- program is finished running --
>0
3
```

PRÁCTICA 5

Estructuras de control

Haz el código que lea dos enteros de la consola y escriba la suma y vuelva a comenzar si el resultado es distinto de 0. Es pseudocódigo sería:

(bucle do-while)

seguir: Leer el primer valor (A)

Leer el segundo valor (B)

Imprimir A+B

Si (A+B) == 0 ir a acabar

ir a seguir

acabar:

Para poder realizar la condición para acabar el bucle haremos uso de la función beq, la cual salta a acabar cuando \$a0=0

The screenshot shows the MARS MIPS simulator interface. The main window displays assembly code for a program that reads two integers, prints their sum, and loops until the sum is zero. The code is as follows:

```
1 .text
2 seguir: li $a0, '\n'
3         li $v0, 11 #Salto de linea
4         syscall
5
6         li $a0, '>'
7         li $v0, 11 #Imprimir
8         syscall
9         li $v0, 5 #Leemos A
10        syscall
11        move $s0, $v0
12        li $a0, '>'
13        li $v0, 11 #Imprimir
14        syscall
15        li $v0, 5 #Leemos B
16        syscall
17        move $s1, $v0
18        add $a0, $s0, $s1
19        li $v0, 1
20        syscall
21
22        beq $a0, $zero, acabar #Salta a acabar, porque vale 0
23        bne $a0, $zero, seguir #Salta a seguir, porque no es 0
24
25 acabar: li $v0, 10 #Acabamos el programa
26
```

The right panel shows the registers, with the 'Value' column displaying hexadecimal values. The 'Registers' table is as follows:

Na...	Nu...	Value
...	0	0x000...
\$at	1	0x000...
\$v0	2	0x000...
\$v1	3	0x000...
\$a0	4	0x000...
\$a1	5	0x000...
\$a2	6	0x000...
\$a3	7	0x000...
\$t0	8	0x000...
\$t1	9	0x000...
\$t2	10	0x000...
\$t3	11	0x000...
\$t4	12	0x000...
\$t5	13	0x000...
\$t6	14	0x000...
\$t7	15	0x000...
\$s0	16	0x000...
\$s1	17	0x000...
\$s2	18	0x000...
\$s3	19	0x000...
\$s4	20	0x000...
\$s5	21	0x000...
\$s6	22	0x000...
\$s7	23	0x000...
\$t8	24	0x000...
\$t9	25	0x000...
\$k0	26	0x000...
\$k1	27	0x000...
\$gp	28	0x100...
\$sp	29	0x7ff...
\$fp	30	0x000...
\$ra	31	0x000...
pc		0x004...
hi		0x000...
lo		0x000...

The bottom panel shows the 'Mars Messages' window with the following output:

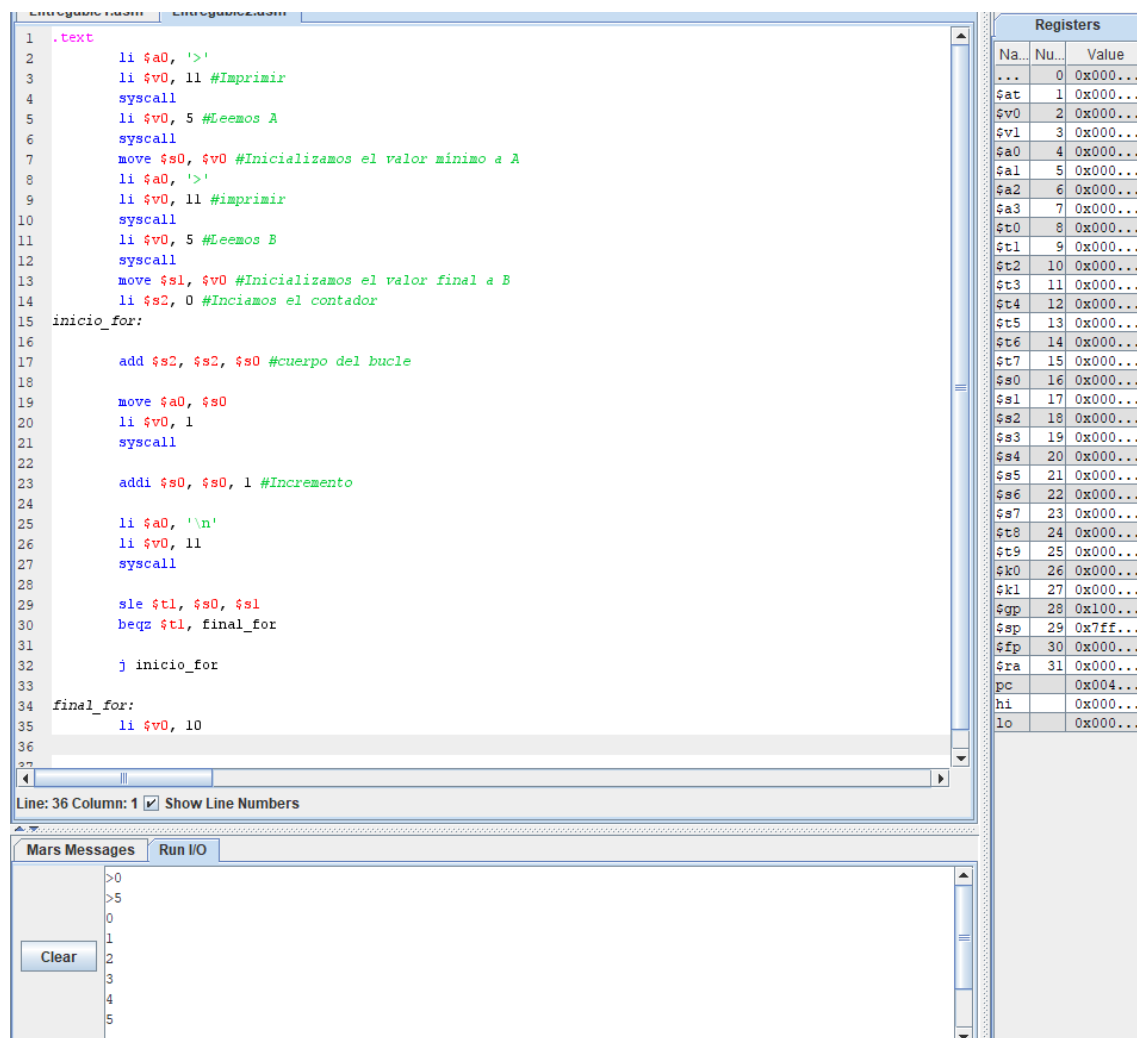
```
>5
>6
11
>2
>4
6
>-4
>4
0
-- program is finished running (dropped off bottom) --
```

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

Para realizar este programa en Mips tendremos que pedir los dos números introducidos para luego usar la función move para mover los datos introducidos a las variables que usaremos para indicar el valor inicial y final del bucle for.



```
1 .text
2     li $a0, '>'
3     li $v0, 11 #Imprimir
4     syscall
5     li $v0, 5 #Leemos A
6     syscall
7     move $s0, $v0 #Inicializamos el valor minimo a A
8     li $a0, '>'
9     li $v0, 11 #imprimir
10    syscall
11    li $v0, 5 #Leemos B
12    syscall
13    move $s1, $v0 #Inicializamos el valor final a B
14    li $s2, 0 #Iniciamos el contador
15 inicio_for:
16     add $s2, $s2, $s0 #cuerpo del bucle
17
18     move $a0, $s0
19     li $v0, 1
20     syscall
21
22     addi $s0, $s0, 1 #Incremento
23
24     li $a0, '\n'
25     li $v0, 11
26     syscall
27
28     sle $t1, $s0, $s1
29     beqz $t1, final_for
30
31     j inicio_for
32
33 final_for:
34     li $v0, 10
35
36
37
```

Registers

Na...	Nu...	Value
...	0	0x000...
\$at	1	0x000...
\$v0	2	0x000...
\$v1	3	0x000...
\$a0	4	0x000...
\$a1	5	0x000...
\$a2	6	0x000...
\$a3	7	0x000...
\$t0	8	0x000...
\$t1	9	0x000...
\$t2	10	0x000...
\$t3	11	0x000...
\$t4	12	0x000...
\$t5	13	0x000...
\$t6	14	0x000...
\$t7	15	0x000...
\$s0	16	0x000...
\$s1	17	0x000...
\$s2	18	0x000...
\$s3	19	0x000...
\$s4	20	0x000...
\$s5	21	0x000...
\$s6	22	0x000...
\$s7	23	0x000...
\$t8	24	0x000...
\$t9	25	0x000...
\$k0	26	0x000...
\$k1	27	0x000...
\$gp	28	0x100...
\$sp	29	0x7ff...
\$fp	30	0x000...
\$ra	31	0x000...
pc		0x004...
hi		0x000...
lo		0x000...

Mars Messages Run I/O

```
>0
>5
0
1
2
3
4
5
```

Clear

Line: 36 Column: 1 Show Line Numbers

PRÁCTICA 6

Variables

Escribe el código que lee dos enteros del teclado mostrando sendos mensajes por consola: uno que pida al usuario que introduzca el primer valor y tras haberlo leído que muestre otro solicitando el segundo valor.

Los datos se almacenarán en la memoria, para lo cual debes haber reservado previamente espacio en el segmento de datos. Una vez almacenados los datos tienes que llamar a una función, que denominaremos SWAP, que intercambie el contenido de las dos posiciones de memoria.

Para finalizar se leerán los valores guardados en la memoria y se mostrarán ordenados de menor a mayor en la pantalla.

Para este programa realizamos el siguiente programa, siendo este su cuerpo:

```
.data
A: .word 0
B: .word 0
FRAS1: .asciiz "Introduce el primer valor: "
FRAS2: .asciiz "Introduce el segundo valor: "
RES1: .asciiz "A es mayor que B: "
RES2: .asciiz "B es mayor que A: "
n: .asciiz "\n" #Salto de línea
.text
    la $a0, FRAS1
    li $v0, 4
    syscall
    li $v0, 5 #Leer un entero
    syscall
    move $s0, $v0 #El valor de A está en $s0
    la $a0, n
    li $v0, 4
    syscall
    la $a0, FRAS2
    li $v0, 4
    syscall
    li $v0, 5
    syscall
    move $s1, $v0 #El valor de B está en $s1
    la $a0, n
    li $v0, 4
    syscall
    la $t0, A
    la $t1, B
    sw $s0, 0($t0) #Copiamos el valor de $s0 en $t0(A)
    sw $s1, 0($t1) #Copiamos el valor de $s1 en $t1(B)
    jal swap
    jal comparar

    li $v0, 10 #Finalizar programa
    syscall
```

Para llevar a cabo este programa usamos las funciones SWAP, comparar y else, esta última para diferenciar según el resultado obtenido en comparar:

```
swap:    move $a0, $s0
         move $a1, $s1
         sw $a0,0($t1) #Copiamos el valor de $s1 en $t1(B) el valor de A
         sw $a1,0($t0) #Copiamos el valor de $s1 en $t0(A) el valor de B

         jr $ra

comparar:
         lw $s1,0,($t0) #Guardamos en s1 el valor de B
         lw $s0,0,($t1) #Guardamos en s0 el valor de A
         bgt $s1, $s0, else #Primero mayor que segundo B>A se activa else
         la $a0, RES2 #B es mayor que A
         li $v0,4
         syscall
         la $a0,n
         li $v0,4
         syscall
         move $a0, $s1
         li $v0, 1
         syscall
         la $a0,n
         li $v0,4
         syscall
         move $a0, $s0
         li $v0, 1
         syscall
         jr $ra

else:
         la $a0, RES1 #RES1 #A es mayor que B
         li $v0,4
         syscall
         la $a0,n
         li $v0,4
         syscall
         move $a0, $s1
         li $v0, 1
         syscall
         la $a0,n
         li $v0,4
         syscall
         move $a0, $s0
         li $v0, 1
         syscall
         jr $ra
```

Obteniendo las siguientes salidas obtenidas a la hora de ejecutar el programa en los dos casos que se nos plantean (No se tiene en cuenta que se introduzcan dos números iguales)

```
Introduce el primer valor: 8
Introduce el segundo valor: 4

B es mayor que A:
4
8
-- program is finished running --

Clear

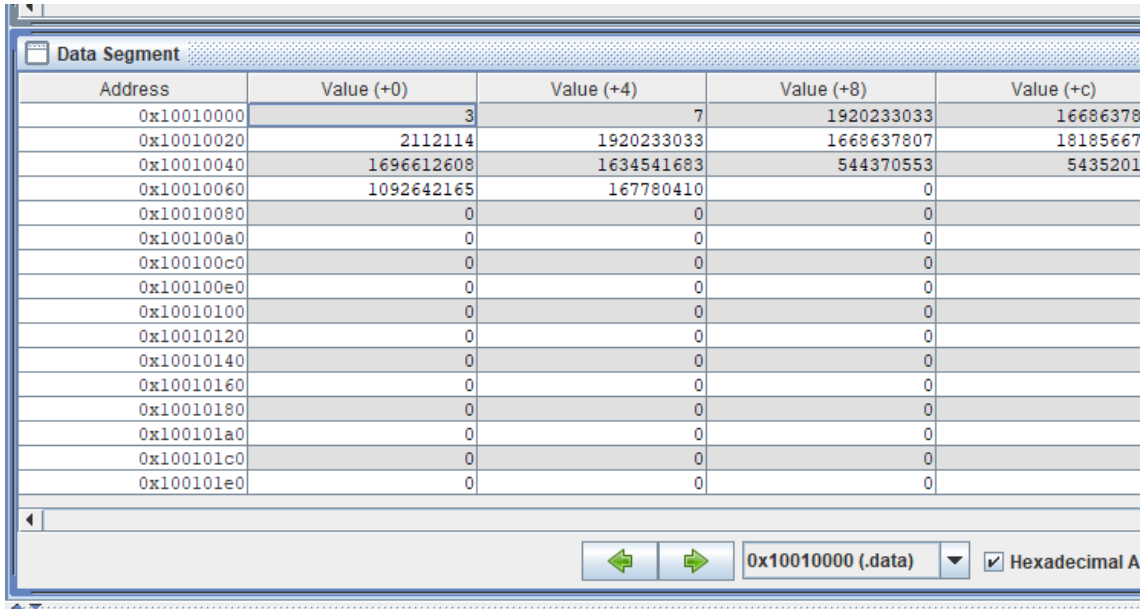
Reset: reset completed.

Introduce el primer valor: 3
Introduce el segundo valor: 7

A es mayor que B:
7
3
-- program is finished running --
```

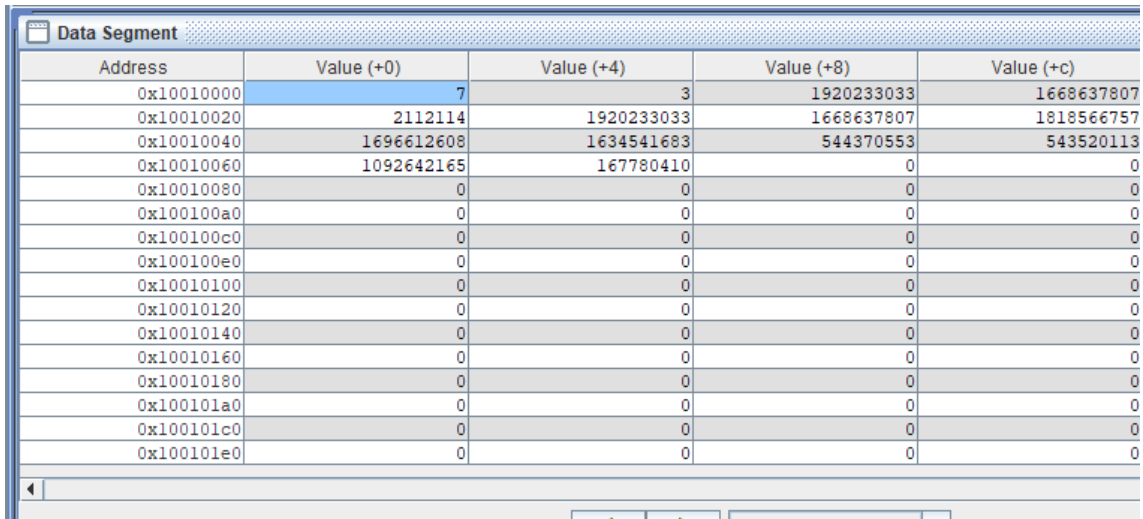
Además, podemos observar como hemos guardado correctamente los datos introducidos dentro de las variables A (Primer número introducido) y B (Segundo número introducido)

Antes de la función SWAP:



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	3	7	1920233033	16686378
0x10010020	2112114	1920233033	1668637807	18185667
0x10010040	1696612608	1634541683	544370553	5435201
0x10010060	1092642165	167780410	0	
0x10010080	0	0	0	
0x100100a0	0	0	0	
0x100100c0	0	0	0	
0x100100e0	0	0	0	
0x10010100	0	0	0	
0x10010120	0	0	0	
0x10010140	0	0	0	
0x10010160	0	0	0	
0x10010180	0	0	0	
0x100101a0	0	0	0	
0x100101c0	0	0	0	
0x100101e0	0	0	0	

Después de la función SWAP:



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	7	3	1920233033	1668637807
0x10010020	2112114	1920233033	1668637807	1818566757
0x10010040	1696612608	1634541683	544370553	543520113
0x10010060	1092642165	167780410	0	0
0x10010080	0	0	0	0
0x100100a0	0	0	0	0
0x100100c0	0	0	0	0
0x100100e0	0	0	0	0
0x10010100	0	0	0	0
0x10010120	0	0	0	0
0x10010140	0	0	0	0
0x10010160	0	0	0	0
0x10010180	0	0	0	0
0x100101a0	0	0	0	0
0x100101c0	0	0	0	0
0x100101e0	0	0	0	0

Siendo la columna Value (+0) la que corresponde a la variable A y la Value (+4) la que corresponde a la variable B

Ejercicios optativos Práctica 4

¿Con qué instrucciones traducirá el ensamblador las pseudoinstrucciones `rol` y `ror`?

Escribe un código sencillo de prueba de ambos operadores y ensámblalos para comprobarlo.

```
.text
    li $t1, 0xABCDABCD
    ror $t2, $t1, 4
```

Haciendo uso de Mips podemos ver que se traduce por las reglas: `sll`, `srl` y `or`

Descubre la palabra escondida.

Dado el siguiente código, complétalo de tal manera que mediante instrucciones lógicas y de desplazamientos puedas escribir en la consola cada uno de los caracteres que se encuentran almacenados en cada byte del registre `$t1`.

```
1  #Palabra escondida
2  li $t1, 1215261793 #0012 1526 1793
3
4  move $t2, $t1
5  srl $t2, $t2, 24 #Desplazar 24 bits
6  andi $t2, $t2, 0x00000000FF
7  move $a0, $t2
8  li $v0, 11
9  syscall
10
11 move $t2, $t1
12 srl $t2, $t2, 16 #Desplazar 16 bits
13 andi $t2, $t2, 0x00000000FF
14 move $a0, $t2
15 li $v0, 11
16 syscall
17
18 move $t2, $t1
19 srl $t2, $t2, 8 #Desplazar 8 bits a la derecha
20 andi $t2, $t2, 0x00000000FF
21 move $a0, $t2
22 li $v0, 11
23 syscall
24
25 move $t2, $t1
26 #No hay que rotarlo
27 andi $t2, $t2, 0x00000000FF
28 move $a0, $t2
29 syscall
30
31 li $v0, 10 #Acaba el programa
32 syscall
33
```

Line: 33 Column: 1 ☒ Show Line Numbers

Mars Messages

Run I/O

```
Hola
-- program is finished running --
```

Como podemos ver, después de ejecutar el programa, la palabra secreta es “Hola”.

Ensamblad el programa y observad en qué direcciones se colocan las instrucciones.

Ejecutadlo a pasos. Fijaos en la ventana de registros como van cambiando los contenidos de los registros PC y \$ra.

```
1  # Prueba de llamada a una función
2  .text
3  li $a0, '>' #Comienza programa principal
4  li $v0,11
5  syscall
6  li $v0,5
7  syscall #Leer un enter
8  addi $t1,$v0,10
9  move $a0, $t1 #argumento a pasar en $a0
10 jal imprim #llamamos a la función
11 add $t1, $t1,$t1
12 move $a0, $t1 #argumento a pasar en $a0
13 jal imprim #llamamos a la función
14 li $v0,10 #Acaba el programa
15 syscall
16 #-----Funcions-----
17 imprim: addi $v0,$0,1 #comienza la función
18 syscall #Escribe un valor
19 li $a0, '\n' #Salto de línea
20 li $v0,11
21 syscall
22 jr $ra #Vuelta al programa principal
23
```

Line: 23 Column: 1 ☒ Show Line Numbers

Mars Messages Run I/O

>5
15
30

Clear -- program is finished running --

Este programa se encarga de pedir un número, para luego sumarle 10 (línea 8) para luego sumar por si mismo el resultado obtenido (línea 11)

Escribe una función con instrucciones suma que devuelve el cuádruplo del número entero que se le pasa.

Escribid el programa principal que lea el número del teclado y escriba el cuádruplo en la consola aprovechando la función imprim.

```
1  # Devuelve el cuádruplo
2  .text
3      li $a0, '>' #Comienza programa principal
4      li $v0,11
5          syscall
6      li $v0,5
7          syscall #Leer un enter
8
9      move $a0, $v0 #argumento a pasar en $a0
10     jal cuadruple
11     move $a0, $v0
12     jal imprim #llamamos a la función
13         syscall
14     li $v0,10 #Acaba el programa
15         syscall
16     #-----Funcions-----#
17 imprim: addi $v0,$0,1 #comienza la función
18         syscall #Escribe un valor
19     li $a0, '\n' #Salto de línea
20     li $v0,11
21         syscall
22     jr $ra #Vuelta al programa principal
23
24 cuadruple:
25     add $v0, $a0, $a0
26     add $v0, $a0, $a0
27     add $v0, $a0, $a0
28     add $v0, $a0, $a0
29     jr $ra #Vuelta al programa principal
30
31
32
```

Line: 30 Column: 1 ☒ Show Line Numbers

Mars Messages Run I/O

```
>12
24
-- program is finished running --
```

Clear

Observad el último código escrito, ¿se ajusta al convenio de MIPS de utilización de registros? De ahora en adelante haced servir el convenio.

→No, no se ajusta al convenio

Ensambla y prueba el programa.

```
3  # Multiplicación por 4
4  .text
5  li $a0, '>'
6  li $v0, 11
7  syscall
8  li $v0, 5
9  syscall #Leer un entero
10 move $a0, $v0 #parámetro a pasar en $a0
11 jal mult4 #llamamos a la función mult4
12 move $a0, $v0
13 jal imprim #llamamos a la función imprim
14 li $v0, 10 #Acaba el programa
15 syscall
16 #-----Funcions-----#
17 imprim: addi $v0, $0, 1 #función imprim
18 syscall #Escribe el valor en $a0
19 li $a0, '\n' #Salto de línea
20 li $v0, 11
21 syscall
22 jr $ra #vuelve al programa principal
23 mult4: sll $v0, $a0, 2 #Función para multiplicar por 4
24         jr $ra
25
```

Line: 25 Column: 1 ☒ Show Line Numbers

Mars Messages Run I/O

>3
12
-- program is finished running --

Clear

Modifica el código en el que ahora haya una función multi5 que multiplique por 5 y muestre el resultado por consola. Comprobar que el resultado es correcto

```
1  # Multiplicación por 5
2  .text
3  li $a0, '>'
4  li $v0, 11
5  syscall
6  li $v0, 5
7  syscall #Leer un entero
8  move $a0, $v0 #parámetro a pasar en $a0
9  jal mult5 #llamamos a la función mult4
10 move $a0, $v0
11 jal imprim #llamamos a la función imprim
12 li $v0, 10 #Acaba el programa
13 syscall
14 #-----Funcions-----#
15 imprim: addi $v0, $0, 1 #función imprim
16 syscall #Escribe el valor en $a0
17 li $a0, '\n' #Salto de línea
18 li $v0, 11
19 syscall
20 jr $ra #vuelve al programa principal
21 mult5: sll $v0, $a0, 2 #Función para multiplicar por 5
22         add $v0, $v0, $a0
23         jr $ra
24
```

Line: 24 Column: 1 ☒ Show Line Numbers

Mars Messages Run I/O

>7
35
-- program is finished running --

Clear

Como podemos observar realiza la operación $7 \times 5 = 35$

Modifícalo ahora para tener una nueva función mult10 que multiplique por 10. Comprobar que el resultado es correcto.

```
1  # Multiplicación por 10
2  .text
3  li $a0, '>'
4  li $v0, 11
5  syscall
6  li $v0, 5
7  syscall #Leer un entero
8  move $a0, $v0 #parámetro a pasar en $a0
9  jal mult10 #llamamos a la función mult 10
10
11 move $a0, $v0
12 jal imprim #Llamamos a la función imprim
13 li $v0, 10 #Acaba el programa
14
15 syscall
16 #-----Funcions-----#
17 imprim: addi $v0, $0, 1 #función imprim
18 syscall #Escribe el valor en $a0
19 li $a0, '\n' #Salto de línea
20 li $v0, 11
21 syscall
22 jr $ra #Vuelve al programa principal
23 mult10: sll $v0, $a0, 3 #Fución para multiplicar por 5(2 elevado a 3 = 8)
24         add $v0, $v0, $a0
25         add $v0, $v0, $a0
26         jr $ra
27
```

Line: 27 Column: 1 ☒ Show Line Numbers

Mars Messages Run I/O

```
>5
50

-- program is finished running --
```

Clear

Modifica el código de tal manera que ahora lo que lea sea una cantidad de hora y muestre por consola la cantidad de segundos.

```
1  .text
2  li $a0, '>'
3  li $v0,11
4  syscall
5  li $v0,5
6  syscall #Leer un entero
7
8  move $a0, $v0 #parámetro a pasar en $a0
9  jal mult60 #llamamos a la función mult60
10
11 move $a0, $v0 #parámetro a pasar en $a0
12 jal mult60 #llamamos a la función mult60 otra vez 60x60
13
14 move $a0, $v0
15 jal imprim #Llamamos a la función imprim
16 li $v0,10 #Acaba el programa
17 syscall
18 #-----Funcions-----#
19 imprim: addi $v0,$0,1 #función imprim
20 syscall #Escribe el valor en $a0
21 li $a0, '\n' #Salto de línea
22 li $v0,11
23 syscall
24 jr $ra #Vuelve al programa principal
25
26 mult60: sll $v0, $a0, 5
27         sll $t0, $a0, 4
28         add $v0, $v0, $t0
29         sll $t0, $a0, 3
30         add $v0, $v0, $t0
31         sll $t0, $a0, 2
32         add $v0, $v0, $t0
33 jr $ra
34
```

Line: 34 Column: 1 ☒ Show Line Numbers

Mars Messages Run I/O

Clear

```
>1
3600
-- program is finished running --
>2
7200
-- program is finished running --
```

Ejercicios optativos Práctica 5

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:

```
1  .text
2      li $a0, '>'
3      li $v0, 11 #Imprimir
4      syscall
5      li $v0, 5 #Leemos el entero A
6      syscall
7      move $s0, $v0
8      li $a0, '>'
9      li $v0, 11 #Imprimimos >
10     syscall
11     li $v0, 5 #Leemos el entero B
12     syscall
13     move $s1, $v0
14
15     slt $t0, $s0, $s1 #Comparamos
16     beq $t0, $0, s0_Mayor #Va a s0_Mayor cuando $t0 vale cero
17
18     move $a0, $s1
19
20     j imprime
21 s0_Mayor:    move $a0, $s0
22
23 imprime:     li $v0, 1
24             syscall
25
26
27
28
29
```

Line: 25 Column: 2 ☒ Show Line Numbers

Mars Messages Run I/O

```
>4
>6
6
-- program is finished running (dropped off bottom) --
>7
>2
7
-- program is finished running (dropped off bottom) --
```

Clear

La instrucción `bltz` (branch if less than zero) salta si el valor es menor que cero y es similar a `bgez` ya que también compara un registro con 0 pero siendo contraria la condición de salto. Cambiad la instrucción `bgez` por `bltz`, ¿Qué modificaciones tendríais que hacer en el código?

```
1  .text
2      li $a0, '>'
3      li $v0, 11 #Indicación de escribir un valor
4      syscall
5      li $v0, 5
6      syscall #Leer el entero A
7      bltz $v0, else # Si (A < 0) salta a exit
8      j exit #Acaba parte if-then
9
10 else: sub $a0, $zero, $v0 #Ahora el else resta
11 exit: li $v0, 1 #Imprimir lo que hay en $a0
12      syscall
13      li $v0, 10 #Acaba el programa
14      syscall
15
```

Line: 15 Column: 1 ☒ Show Line Numbers

Mars Messages Run I/O

```
>-3
3
-- program is finished running --
```

Añade al código anterior la posibilidad de leer un valor *n* por teclado y escribe el resultado de la suma por consola. Haz que haya un bucle infinito que lea por teclado excepto en el caso de que se escriba 0, en el cual se saldrá del programa

```
1  .text
2  inicio:
3      li $a0, '>'
4      li $v0, 11 #Imprimir
5      syscall
6      li $v0, 5 #Leemos n
7      syscall
8
9      move $s1, $v0 #Ahora s1 es v0
10     li $s0, 1 #Iniciamos contador
11     #li $s1, 11 Ya no hace falta
12     li $s2, 0 #Contador
13 inicio_for:
14
15     add $s2, $s2, $s0 #cuerpo del bucle
16     addi $s0, $s0, 1 #incremento del contador
17
18     move $a0, $s2
19     li $v0, 1
20     syscall
21
22     li $a0, '\n'
23     li $v0, 11
24     syscall
25
26     sle $t1, $s0, $s1
27     beqz $t1, final_for
28
29     j inicio_for
30
31 final_for:
32     li $v0, 10
33 final: bne $s1, $zero, inicio
```

Line: 34 Column: 1 ☒ Show Line Numbers

Mars Messages Run I/O

Clear

>2
1
3
>5
1
3
6
10
15
>0
1

Ensamblad el programa y comprobad que podéis ver tanto el segmento de texto como el segmento de datos. Comprobad que podéis ver los datos en hexadecimal y en decimal.

```
1 .text #Comienza el programa
2     la $t0,A
3     la $t1,B
4     la $t2,C
5     lw $s0,0($t0) #Guarda en s0 el valor 25(A)
6     lw $s1,0($t1) #Guarda en s1 el valor de 10(B)
7     add $s2,$s1,$s0
8     add $s2,$s2,$s2
9     sw $s2,0($t2) #Copiar el valor de $s2 a $t2
10    li $v0, 10 #Acaba el programa
11    syscall
12
13
14
15
16
17
```

Line: 17 Column: 2 ☒ Show Line Numbers

Mars Messages Run I/O

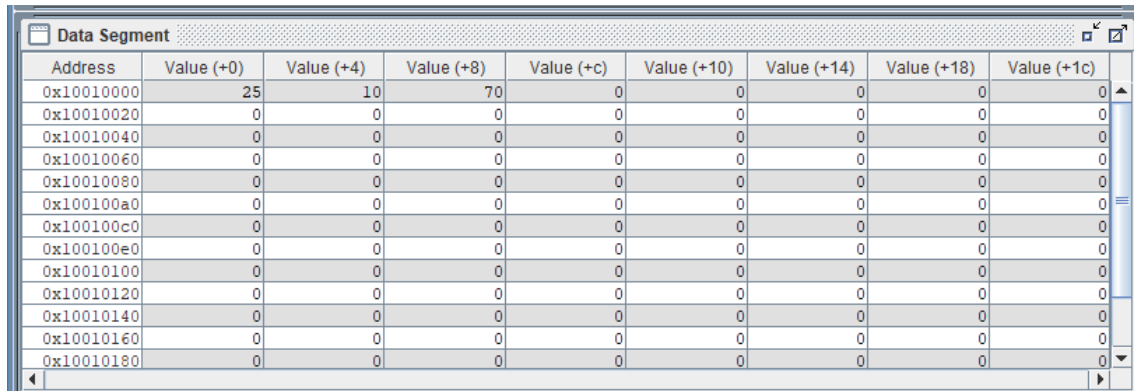
-- program is finished running --

Obtenemos en hexadecimal:

Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000019	0x0000000a	0x00000046	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) ☒ Hexadecimal Addresses ☒ Hexadecimal Values ☐ ASCII

Obtenemos en decimal:



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	25	10	70	0	0	0	0	0
0x10010020	0	0	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0
0x10010100	0	0	0	0	0	0	0	0
0x10010120	0	0	0	0	0	0	0	0
0x10010140	0	0	0	0	0	0	0	0
0x10010160	0	0	0	0	0	0	0	0
0x10010180	0	0	0	0	0	0	0	0

¿Cuántos bytes de la memoria principal están ocupados por datos del programa?

→ $4+4+4=12$

¿Cuántas instrucciones de acceso a la memoria contiene el programa?

→ 3

¿Qué valor tiene el registro \$t1 cuando se ejecuta la instrucción lw \$s1,0(\$t1)?

→ La dirección de B

¿En qué dirección se almacena el resultado?

→ En C

Sustituid la instrucción sw \$s2,0(\$t2) por sw \$s2,2(\$t2) ¿Qué ocurre cuando se intenta ejecutar el programa? Razonad la respuesta

→ Estamos cogiendo una palabra que no es múltiplo de cuatro, así que da error

¿Cuál es la codificación en lenguaje máquina de la instrucción lw \$s1,0(\$t1)? Desglosa la instrucción en los distintos campos del formato.

→ $0x8d310000 = 10011$ Cod operación

Analiza el código y averigua que hace.

```
1  #Codigo de partida
2  .data
3  A: .word 6
4  B: .word 8
5  C: .space 4
6  X: .byte 1
7  VAE1: .asciiz "Estructuras de los Computadores"
8  VAE2: .asciiz "Curso 2018-2019\n"
9  VAE3: .asciiz "\n El resultado de la suma es: "
10 .text
11     la $a0,VAE1
12     li $v0,4
13     syscall
14     li $a0,'\n'
15     li $v0,11
16     syscall
17     la $a0,VAE2
18     li $v0,4
19     syscall
20     la $a0,VAE3
21     li $v0,4
22     syscall
23     la $t0,A # $t0 = 6
24     lw $t1,0($t0) # $t1 = *$t0
25     lw $t2,4($t0) # $t2 = *($t0+4)
26     add $t3,$t1,$t2
27     sw $t3,8($t0)
28     move $a0, $t3
29     addi $v0,$v0,1
30     syscall #Escribe un valor
31     li $v0, 10 #Acaba el programa
32     syscall
33
```

Line: 33 Column: 3 ☒ Show Line Numbers

Mars Messages Run I/O

```
Estructuras de los Computadores
Curso 2018-2019

El resultado de la suma es: 14
-- program is finished running --
```

Clear

Ensambla el código y ejecútalo. ¿Qué hace la función 4 para la instrucción syscall?

→ Imprime cadena de caracteres

¿Cuál es la codificación máquina de la instrucción syscall?

→ 0x00000c

¿En qué dirección se guarda el resultado de la suma?

→ 0x100100008