

# ESTRUCTURA DE LOS COMPUTADORES



## MEMORIA DE PRÁCTICAS 4-5-6

Luis Girona Pérez

12/03/2020

## PRÁCTICA 4

### CUESTIÓN 5

- Observeu l'últim codi escrit de la qüestió 4, s'ajusta al conveni MIPS d'utilització de registres? Què hauríeu de modificar? Reescriuiu el codi perquè s'ajuste al conveni. D'ara endavant feu servir el conveni d'utilització de registres.

```
1
2 li $a0, '>' # imprimimos el caracter >
3 li $v0, 11
4 syscall
5
6 li $v0, 5 # leemos un entero
7 syscall
8
9 jal quadruple
10
11
12 move $a0,$s0 # Movemos el numero que hemos calculado en la funcion en $a0
13 li $v0,1 # para que pueda ser impreso por pantalla
14 syscall
15
16
17 li $v0,10 # terminamos el programa
18 syscall
19
20 quadruple:
21     move $s0, $v0 # movemos el entero leído a la variable $s0
22     rol $s0,$s0,2 # desplazamos 2 posiciones ya que 2^2 es 4
23     jr $ra
24
```

Este código es el que tenemos que modificar para cumplir el convenio. Tendríamos que cambiar las variables de paso de parámetros en la función de cuádruple, ya que utilizamos las variables s y son para el programa principal. Por lo que el programa nos quedaría tal que así.

```
1
2 li $a0, '>' # imprimimos el caracter >
3 li $v0, 11
4 syscall
5
6 li $v0, 5 # leemos un entero
7 syscall
8
9 move $a0, $v0 # Movemos el numero que introducimos a a0 para pasarlo por parametro
10 jal quadruple
11
12
13 li $v0,1 # para que pueda ser impreso por pantalla
14 syscall
15
16
17 li $v0,10 # terminamos el programa
18 syscall
19
20 quadruple:
21     rol $a0,$a0,2 # desplazamos 2 posiciones ya que 2^2 es 4
22     jr $ra
23
```

He señalado los cambios que he realizado. El move modificado lo he hecho para tener el numero introducido en \$a0 y pasarlo por parámetro a la función. El segundo cambio ha sido cambiar los \$s utilizados dentro de la función (los cuales solo se usan en el programa principal) por \$a0, para tenerlo guardado en una variable que se pase por parámetro y poder imprimirlo.

## CUESTION 6

- Modifiqueu el codi de l'activitat 3 perquè ara hi haja una funció mult5 que multiplique per 5 i mostreu el resultat per consola. (Fixeu-vos que al valor obtingut en la multiplicació per 4 haureu de sumar-li una vegada el valor original,  $x*5=x*4+x$ ). Comproveu que el resultat és correcte.

```
4 # Multiplicació per 4 #
5 # #
6 #####
7 # Codi de partida de l'activitat 3
8 #
9 # Multiplicació per 4
10 #
11 .text
12 li $a0, '>' #Comença demanant un enter
13 li $v0, 11
14 syscall
15 li $v0, 5
16 syscall
17 move $a0, $v0 #Llig un enter
18 jal mult5 #cridem la funció mult4
19 move $a0, $v0 #paràmetre a passar en $a0
20 jal imprim #cridem la funció imprim
21 li $v0, 10 #Acaba el programa
22 syscall
23 #####
24 # #
25 # Funcions #
26 # #
27 #####
28 imprim:
29 addi $v0, $0, 1 #funció imprim
30 syscall #Escriu el valor en $a0
31 li $a0, '\n' #Salt de línia
32 li $v0, 11
33 syscall
34 jr $ra #Torna al programa principal
35
36 mult5:
37 sll $v0, $a0, 2 #Funció multiplica per 4
38 add $v0, $v0, $a0 # para realizar la multiplicacion por 5 multiplicamos primero por 4
# y despues sumamos una vez el numero al resultado de la multiplicacion
```

Line: 38 Column: 82 Show Line Numbers

Mars Messages Run I/O

```
>5
25
-- program is finished running --

Clear

Reset: reset completed.

>10
50
```

Coproc 0		
Coproc 1		
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...

Para obtener la multiplicación por 5, lo único que he tenido que hacer ha sido añadir una línea de código con la instrucción add, lo que hace es sumar el número que hemos introducido en la consola con la multiplicación por 4 que hemos hecho en la línea de arriba, obteniendo así la multiplicación x5.

## CUESTION 8

- **Escriviu una funció que multiplique per 60. Escriviu el programa principal que lliga una quantitat de minuts i retorne per consola la quantitat en segons.**

Tenemos que multiplicar el número que introduzcamos por 60. Al pasarlo a binario obtenemos el número 111100. Podemos decir que multiplicar por 60 es lo mismo que multiplicar el número por  $1x2^5 + 1x2^4 + 1x2^3 + 1x2^2 + 0x2^1 + 0x2^0$ , por lo que la multiplicación la realizaremos mediante desplazamientos.

He traducido la operación descrita en el párrafo anterior como 4 desplazamientos a la izquierda de 5, 4, 3 y 2 bits. Después he sumado los resultados mediante 3 add, lo muevo a \$a0 y lo muestro por pantalla.

The screenshot shows the Mars MIPS simulator interface. The main window displays the assembly code for 'question8.asm'. The code starts with a main function that prompts the user for an input, calls a function 'mult60', and then prints the result. The 'mult60' function uses four shift instructions (sll) to shift the input left by 5, 4, 3, and 2 bits, then adds the results together using three 'add' instructions, and finally moves the result to register \$a0.

On the right side, the 'Coproc 1 Registers' window shows the state of the registers. The 'Na...' column shows the register name, 'Nu...' shows the register number, and 'Value' shows the current value. The registers are initialized to 0x000...

At the bottom, the 'Mars Messages' window shows the output of the program. The first run shows the input '1' and the output '60'. The second run shows the input '2' and the output '120'. The program is finished running in both cases.

```

1  li $a0, '>'          #Comença demanant un enter
2  li $v0, 11
3  syscall
4  li $v0, 5
5  syscall              #lleg un enter
6  move $a0, $v0        #paràmetre a passar en $a0
7
8  jal mult60           #cridem la funció mult60
9
10 li $v0, 1            # mostrem el resultat
11 syscall
12
13 li $v0, 10
14 syscall
15
16
17 mult60:
18     sll $s0, $a0, 5    # desplaçamos 5
19     sll $s1, $a0, 4    # desplaçamos 4
20     sll $s2, $a0, 3    # desplaçamos 3
21     sll $s3, $a0, 2    # desplaçamos 2
22
23     add $s4, $s0, $s1   # sumamos los resultados de los desplazamientos
24     add $s5, $s2, $s3   # sumamos los resultados de los desplazamientos
25     add $s6, $s4, $s5   # sumamos y obtenemos el total
26
27     move $a0, $s6       # movemos el resultado total a a0 para mostrarlo
28     jr $ra
  
```

Line: 28 Column: 8 ☒ Show Line Numbers

Mars Messages Run I/O

```

>1
60
-- program is finished running --

Clear

Reset: reset completed.

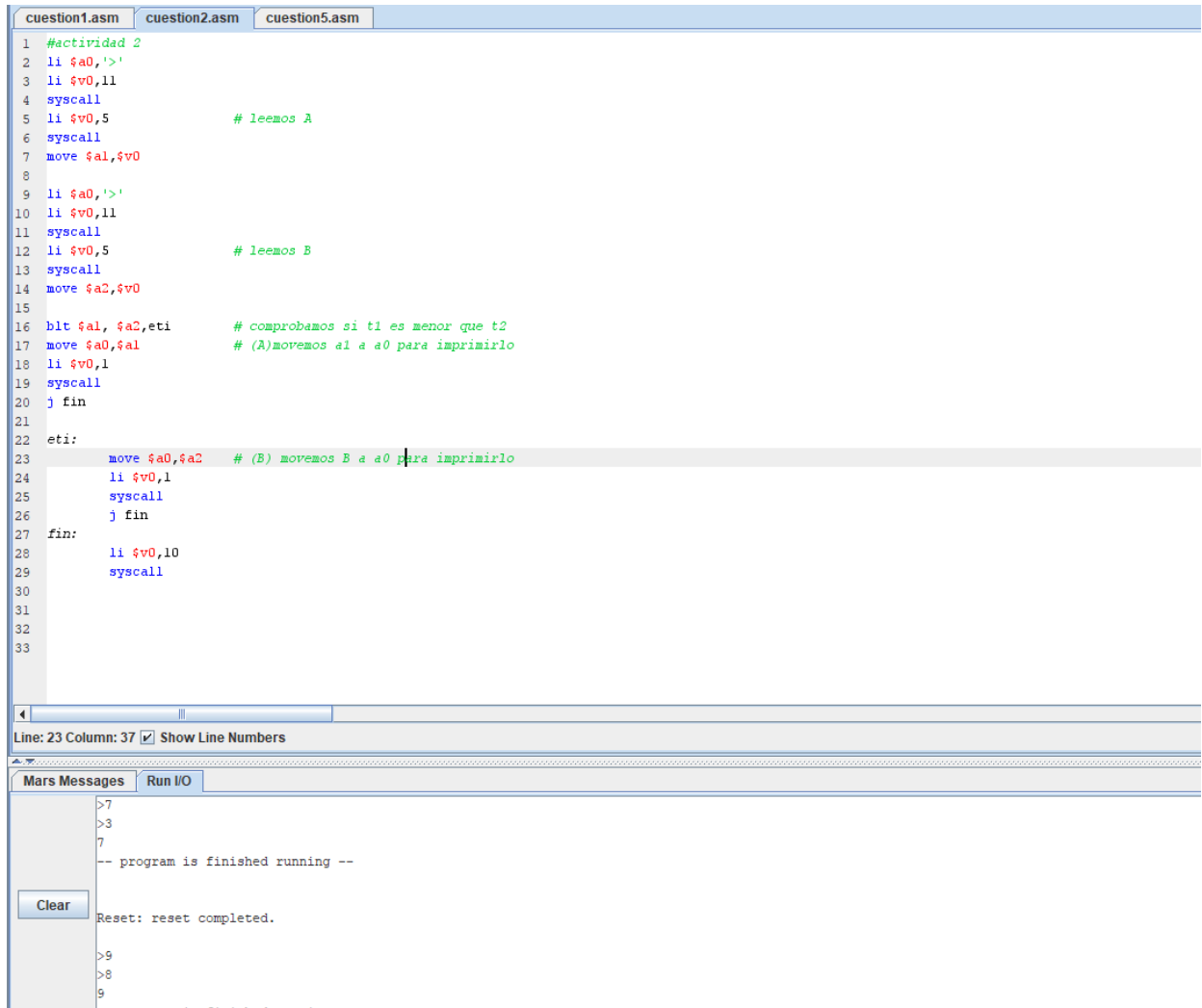
>2
120
-- program is finished running --
  
```

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...

## PRÁCTICA 5

### CUESTIÓN 2

- Escribiu el programa que llig dos enters del teclat i escriu en la consola el més gran. El programa ha de tenir l'estructura següent:  
Assembla i prova el programa amb diferents valors d'A i B



```
1 #actividad 2
2 li $a0,'>'
3 li $v0,11
4 syscall
5 li $v0,5          # leemos A
6 syscall
7 move $a1,$v0
8
9 li $a0,'>'
10 li $v0,11
11 syscall
12 li $v0,5          # leemos B
13 syscall
14 move $a2,$v0
15
16 blt $a1, $a2,eti  # comprobamos si t1 es menor que t2
17 move $a0,$a1      # (A)movemos a1 a a0 para imprimirlo
18 li $v0,1
19 syscall
20 j fin
21
22 eti:
23     move $a0,$a2    # (B) movemos B a a0 para imprimirlo
24     li $v0,1
25     syscall
26     j fin
27
28 fin:
29     li $v0,10
30     syscall
31
32
33
```

Line: 23 Column: 37 ☒ Show Line Numbers

Mars Messages Run I/O

```
>7
>3
7
-- program is finished running --

Clear
Reset: reset completed.

>9
>8
9
-- program is finished running --
```

Lo que he hecho ha sido primero, pedir dos números con el syscall 5, después he utilizado blt para que salte a “eti” cuando el número A sea menor que B. Dentro de eti movemos el valor de B a \$a0 para poder imprimirlo por pantalla y después saltamos a la etiqueta para finalizar el programa.

En caso de que B sea menor que A no saltaríamos a eti y seguiríamos con la instrucción siguiente, que sería mover el valor de A a \$a0 para imprimir por pantalla y finalizaríamos el programa.

## CUESTIÓN 7

- En ejemplos com l'anterior, en què sabem que almenys el bucle s'executa una vegada, és més eficient posar la condició d'eixida del bucle al final. Reescriuiu el codi anterior amb la condició d'eixida al final del bucle (es tracta d'un bucle del tipus do-while).

cuestion7.asm			Name	Number	Value
1	#####		\$zero	0	0
2	#		\$at	1	1
3	# Código ejemplo #		\$v0	2	10
4	# Bucle FOR-WHILE #		\$v1	3	0
5	#		\$a0	4	0
6	#####		\$a1	5	0
7	li \$s0, 1	#Iniciamos contador	\$a2	6	0
8	li \$s1, 11	#Condición de finalización	\$a3	7	0
9	li \$s2, 0	#Contador	\$t0	8	0
10	inicio_dowhile:		\$t1	9	0
11	add \$s2, \$s2, \$s0	#cuerpo del bucle	\$t2	10	0
12	addi \$s0, \$s0, 1	#incremento del contador	\$t3	11	0
13	sle \$t1, \$s0, \$s1	# nos indica si s0 es menor o igual que s1, si no lo es, pone t1 a 0	\$t4	12	0
14	beqz \$t1, final_dowhile	# si t1 es igual a 0, el bucle finaliza	\$t5	13	0
15	j inicio_dowhile		\$t6	14	0
16			\$t7	15	0
17			\$s0	16	12
18			\$s1	17	11
19			\$s2	18	66
20			\$s3	19	0
21			\$s4	20	0
22			\$s5	21	0
23			\$s6	22	0
			\$s7	23	0
			\$t8	24	0
			\$t9	25	0
			\$k0	26	0
			\$k1	27	0
			\$gp	28	268468224
			\$sp	29	2147479548
			\$fp	30	0
			\$ra	31	0
			pc		4194352
			hi		0
			lo		0

Lo que he hecho aquí ha sido modificar el código del ejemplo anterior de tal manera que haga primero las operaciones y después que haga las comprobaciones. Por lo que pasaríamos de un for a un do-while.

Al inicio del bucle realiza la operación e incrementamos el contador, después comprueba si el contador y el límite del contador son iguales. En caso de que lo fueran el registro \$t1 se pondría a 0, en caso contrario a 1.

Por último, se comprueba si \$t1 es igual a 0, es decir si el contador ha llegado a su límite o no. Si no ha llegado al límite, saltará de nuevo a la etiqueta inicio\_dowhile y volverá a ejecutar lo mismo que antes. En caso de que hubiera llegado irá a la etiqueta final\_dowhile y finalizará el programa.

## CUESTIÓN 10

- Feu el codi que llig de teclat dos valors positius A i B en els quals  $A < B$ . El programa ha d'escriure per consola els valors compresos entre ells inclosos ella mateixa. Es a dir, si  $A=3$  i  $B=6$ , escriu en la consola 3 4 5 6 (podeu escriure, per exemple, un salt de línia després de cadascun dels valors a mostrar).

The screenshot shows the Mars IDE with the assembly code for 'cuestion10.asm' and its execution output.

**Assembly Code (cuestion10.asm):**

```

1  #código que lee de teclado dos valores positivos A y B en los que A<B
2
3  li $a0, '>'
4  li $v0, 11
5  syscall
6  li $v0, 5      # leemos A
7  syscall
8
9  move $s0, $v0  # guardamos en s1 el numero A
10
11 li $a0, '>'
12 li $v0, 11
13 syscall
14 li $v0, 5      # leemos B
15 syscall
16
17 move $s1, $v0  # guardamos en s1 el numero B
18
19 bucle:
20     move $a0, $s0      # movemos el numero A a a0 para imprimir
21     li $v0, 1          # imprimos el numero
22     syscall
23     li $a0, '\n'       # imprimimos un salto de linea
24     li $v0, 11
25     syscall
26     beq $s0, $s1, final # si $s0 es igual a $s1 saltamos a finalizar el programa
27     add $s0, $s0, 1    # aumentamos el contador
28     j bucle
29
30 final:
31     li $v0, 10
32     syscall

```

**Execution Output:**

```

>3
>8
3
4
5
6
7
8
-- program is finished running --

```

The output shows the program reading two values, 3 and 8, and then printing the numbers from 3 to 8, each on a new line. The program then finishes running.

Solicitamos los dos números y dentro de un bucle movemos el número A (que lo utilizamos como contador) a \$a0 para imprimirlo nada más comenzar, después tenemos un salto de línea. Ahora es cuando con la instrucción beq comprobamos si tenemos que terminar el bucle o no, miramos si \$s0(contador) es igual a \$s1(límite del contador), si lo es salta a final y termina el programa. Si no lo es, pasa a la siguiente instrucción que aumenta en 1 el contador y utiliza la instrucción de salto para volver al principio de bucle.

## PRÁCTICA 6

### CUESTIÓN 3

- Quin valor té el registre \$t1 quan s'executa la instrucció lw \$s1,0(\$t1)?

The screenshot shows the Mars MIPS simulator interface. The 'Text Segment' window displays the following assembly code:

Bkpt	Address	Code	Basic	Source
	4194304	0x3c011001	lui \$1,4097	10: la \$t0,A
	4194308	0x34280000	ori \$8,\$1,0	
	4194312	0x3c011001	lui \$1,4097	11: la \$t1,B
	4194316	0x34290004	ori \$9,\$1,4	
	4194320	0x3c011001	lui \$1,4097	12: la \$t2,C
	4194324	0x342a0008	ori \$10,\$1,8	
	4194328	0x8d100000	lw \$16,0(\$8)	13: lw \$s0,0(\$t0)
	4194332	0x8d310000	lw \$17,0(\$9)	14: lw \$s1,0(\$t1)
	4194336	0x02309020	add \$18,\$17,\$16	15: add \$s2,\$s1,\$s0
	4194340	0x02529020	add \$18,\$18,\$18	16: add \$s2,\$s2,\$s2
	4194344	0xad520000	sw \$18,0(\$10)	17: sw \$s2,0(\$t2)
	4194348	0x2402000a	addiu \$2,\$0,10	18: li \$v0, 10 #Acaba el programa syscall

The 'Data Segment' window shows memory values:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value
268500992	25	10	0	0	0
268501024	0	0	0	0	0
268501056	0	0	0	0	0
268501088	0	0	0	0	0
268501120	0	0	0	0	0
268501152	0	0	0	0	0
268501184	0	0	0	0	0
268501216	0	0	0	0	0
268501248	0	0	0	0	0
268501280	0	0	0	0	0
268501312	0	0	0	0	0
268501344	0	0	0	0	0
268501376	0	0	0	0	0
268501408	0	0	0	0	0

The 'Registers' window shows the following values:

Name	Number	Value
\$zero	0	0
\$at	1	268500992
\$v0	2	0
\$v1	3	0
\$a0	4	0
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	268500992
\$t1	9	268500996
\$t2	10	268501000
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	25
\$s1	17	10
\$s2	18	0
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$s8	24	0
\$s9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	0
pc		4194336
hi		0
lo		0

Como vemos en la imagen, \$t1 tiene como valor 268500996. Podemos comprobar que ese valor coincide con la dirección del valor que tenemos en \$s1. Ya que, si vemos, la flecha de la izquierda señala a 268500992 y la segunda flecha a +4, lo que coincidiría al hacer la suma con la dirección del valor 10 almacenado en \$s1.

- En quina adreça s'emmagatzema el resultat?

El resultado final del programa se almacena en la dirección 268501000

Data Segment				
Address	Value (+0)	Value (+4)	Value (+8)	
268500992	25	10	70	
268501024	0	0	0	
268501056	0	0	0	
268501088	0	0	0	
268501120	0	0	0	
268501152	0	0	0	



## CUESTIÓN 7

- Feu el codi que llig dos enters del teclat. Amb aquesta finalitat heu de mostrar dos missatges en la consola: un primer que demane a l'usuari que introduïska un valor i una vegada llegit, que mostre un altre missatge demanant el segon valor. Les dades s'emmagatzemaran en posicions consecutives de la memòria; per això, prèviament haureu reservat espai en el segment de dades amb la directiva `.space`. A continuació el programa llegirà els valors guardats en la memòria i els mostrarà en la pantalla.

The screenshot shows the Mars MIPS simulator interface. The main window displays the assembly code for 'cuestion7.asm'. The code is as follows:

```
1 .data
2 espacio: .space 8
3 texto1: .asciiz "Introduce el primer valor: "
4 texto2: .asciiz "Introduce el segundo valor: "
5
6
7 .text
8 la $s0, espacio
9
10 #Primer Numero
11 la $a0, texto1      # mostramos texto1
12 li $v0, 4
13 syscall
14 li $a0, '\n'
15 li $v0, 11
16 syscall
17
18 li $v0, 5           # leemos entero y lo guardamos en s1
19 syscall
20 move $s1, $v0       # guardamos num1 en s1
21 sw $s1, 0($s0)      # lo almacenamos en la memoria
22
23
24
25 #Segundo Numero
26 la $a0, texto2      # mostramos texto2
27 li $v0, 4
28 syscall
29 li $a0, '\n'
30 li $v0, 11
31 syscall
32
33 li $v0, 5           #leemos entero y lo guardamos en s2
34 syscall
35 move $s2, $v0       #lo guardamos en s2
36 sw $s2, 4($s0)      # lo almacenamos en memoria
37
38 # leer memoria e imprimir
39
40 li $a0, '\n'
```

The right panel shows the registers table:

Name	Number	Value
\$zero	0	0
\$at	1	268500992
\$v0	2	10
\$v1	3	0
\$a0	4	7
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	0
\$t1	9	0
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	268500992
\$s1	17	3
\$s2	18	7
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$s8	24	0
\$s9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	0
pc		4194456
hi		0
lo		0

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

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

The status bar at the bottom indicates 'Line: 1 Column: 1' and 'Show Line Numbers' is checked.

```

37
38 # leer memoria e imprimir
39
40 li $a0, '\n'
41 li $v0, 11
42 syscall
43
44 lw $a0, 0($s0)      # sacamos el primer numero y lo imprimimos
45 li $v0, 1
46 syscall
47
48 li $a0, '\n'
49 li $v0, 11
50 syscall
51
52 lw $a0, 4($s0)      # sacamos el segundo numero y lo imprimimos
53 li $v0, 1
54 syscall
55
56 li $v0, 10
57 syscall
58
59
60
61

```

Line: 1 Column: 1 ☒ Show Line Numbers

Mars Messages Run I/O

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

Clear

\$t6	14	0
\$t7	15	0
\$s0	16	268500992
\$s1	17	3
\$s2	18	7
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$t8	24	0
\$t9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	0
pc		4194456
hi		0
lo		0

Reservamos un espacio de memoria de 8. Declaramos dos .asciiz que serán los textos que mostraremos más a delante. Guardamos la dirección del espacio en \$s0. Introducimos el primer numero y lo guardamos en la variable \$s1, y mediante un sw lo almacenamos en memoria con sw \$s1, 0(\$s0), siendo 0 el comienzo del espacio y \$s0 la dirección del espacio.

Con el segundo numero hacemos exactamente lo mismo salvo a la hora de almacenarlo en memoria, ya que utilizaremos sw \$s2, 4(\$s0), aumentamos 4 para pasar a la siguiente palabra.

Ahora pasaríamos a leer los datos de la memoria e imprimirlos por pantalla. Para esto utilizaremos la instrucción lw \$a0, 0(\$s0), cargamos lo que haya almacenado en la posición 0(\$s0) en el registro \$a0 y con su syscall correspondiente lo imprimimos.

Con el segundo número sería igual cambiando la dirección de memoria por 4(\$s0).

## CUESTIÓN 8

- Modifiqueu el programa de la qüestió 7 perquè mostre en la pantalla les dues dades guardades en la memòria ordenades de menor a major valor.

El código es el mismo que en la cuestión anterior salvo en la parte de imprimir, que es la siguiente:

The screenshot shows the Mars MIPS simulator interface. The main window displays the assembly code for 'cuestion8.asm'. The code reads two numbers from memory, compares them, and prints them in ascending order. The Coprocessor 1 Registers window on the right shows the state of the registers, with \$t0 containing 7 and \$t1 containing 3.

```
38 # leer memoria e imprimir
39
40 lw $t0, 0($s0)      # sacamos el primer numero
41 lw $t1, 4($s0)      # sacamos el segundo numero
42 blt $t0,$t1,menormayor
43 j mayoromenor
44
45 menormayor:
46     li $a0, '\n'     # imprimimos salto de linea
47     li $v0, 11
48     syscall
49
50     move $a0,$t0      # movemos num1 para imprimirlo primero
51     li $v0,1
52     syscall
53     li $a0, '\n'
54     li $v0,11
55     syscall
56
57     move $a0, $t1     # imprimimos num2
58     li $v0,1
59     syscall
60     j fin             # terminamos programa
61 mayoromenor:
62     li $v0, '\n'
63     li $v0,11
64     syscall
65
66     move $a0,$t1      # movemos num2 a a0 para imprimirlo primero
67     li $v0,1
68     syscall
69     li $a0, '\n'
70     li $v0,11
71     syscall
72
73     move $a0,$t0      # movemos num1 y lo imprimimos en segundo lugar
74     li $v0,1
75     syscall
76     j fin
77 fin:
78     li $v0,10
79     syscall
```

Registers window (Coproc 1):

Na...	Nu...	Value
...	0	0
\$at	1	0
\$v0	2	10
\$v1	3	0
\$a0	4	7
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	7
\$t1	9	3
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	26850...
\$s1	17	7
\$s2	18	3
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$s8	24	0
\$s9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	26846...
\$sp	29	21474...
\$fp	30	0
\$ra	31	0
pc		4194532
hi		0
lo		0

Mars Messages window:

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

The screenshot shows the Mars MIPS simulator interface with the Mars Messages window displaying the output of the program. The output shows the two numbers being compared and printed in ascending order, followed by a message indicating the program is finished running.

```
Introduce el segundo valor:
3
3
7
-- program is finished running --
Introduce el primer valor:
3
Introduce el segundo valor:
7
3
7
-- program is finished running --
```

Lo que hacemos es cargar los números con la instrucción lw en un registro, \$t0 y \$t1 respectivamente. Utilizamos la instrucción blt para que salte a la etiqueta menormayor si \$t0 es menor que \$t1. En ese caso iríamos a la función menormayor e imprimiríamos primero \$t0 y luego \$t1.

En caso de que el mayor fuera \$t0, no realizaría el salto a la etiqueta menormayor, sino que pasaría a la siguiente instrucción que sería un salto a la etiqueta mayormenor, en la que imprimiríamos primero el registro \$t1 y luego \$t0 y finalizaría el programa.