

# ESTRUCTURA DE LOS COMPUTADORES



## MEMORIA DE PRÁCTICAS 1-2-3

Luis Girona Pérez

19/02/2020

## PRÁCTICA 1

### CUESTIÓN 4

Com s'escriu la instrucció que fa  $\$8 = \$8 - 1$ ?

```
prac1.asm  cuestion6.asm  cuestion5.asm  cuestion4.asm
1  #cuestion 4
2  addi $8, $8, -1
```

La instrucció addi suma elements, per lo que haria lo siguiente, al segundo parámetro (\$8), le sumamos un -1.

El resultado de esta operación es guardado en el primer parámetro que aparece, \$8.

Com en quedaria la codificació en binari

0010 0001 0000 1000 1111 1111 1111 1111

001000	01000	01000	1111 1111 1111 1111
Operación	RS	RT	K

## CUESTIÓN 5

prac1.asm	cuestion6.asm	cuestion5.asm	cuestion4.asm
<pre>1  .text 0x00400000 2  #addi \$8,\$0,1 3  #addi \$9,\$8,25 4  #addi \$10,\$8,5 5  #addi \$12,\$10,0 6  #Convenio de registros 7  addi \$t0,\$zero,1 8  addi \$t1,\$t0,25 9  addi \$t2,\$t0,5 10 addi \$t4,\$t2,0</pre>			

Como vemos en la imagen, el texto comentado sería el código anterior, mientras que el de abajo es con el convenio de registros.

## CUESTIÓN 6

Escriu el codi que faça les següents accions utilitzant el conveni de registres i utilitzant la instrucció addi:

$\$12 = 5$

$\$10 = 8$

$\$13 = \$12 + 10$

$\$10 = \$10 - 4$

$\$14 = \$13 - 30$

$\$15 = \$10$

Assembleu i executeu el programa i comproveu que el resultat final és  $\$t7 = \$t2 = 4$ ,  $\$t6 = -15$ ,  $\$t4 = 5$ ,  $\$t5 = 15$ .

The screenshot shows an assembly editor with the following code in `cuestion6.asm`:

```
1 #Cuestion 6
2 addi $t4,$zero,5
3 addi $t2,$zero,8
4 addi $t5,$t4,10
5 addi $t2,$t2,-4
6 addi $t6,$t5,-30
7 addi $t7,$t2,0
8
```

On the right, the Register window displays the state of the registers:

Name	Number	Value
\$zero	0	0
\$at	1	0
\$v0	2	0
\$v1	3	0
\$a0	4	0
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	0
\$t1	9	0
\$t2	10	4
\$t3	11	0
\$t4	12	5
\$t5	13	15
\$t6	14	-15
\$t7	15	4
\$s0	16	0
\$s1	17	0
\$s2	18	0
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$t8	24	0
\$t9	25	0

Dado el “esquema” del código, utilizamos las instrucciones de suma pertinentes y el convenio de registros. Nos quedaría un código como el de arriba.

Como vemos, podemos comprobar el resultado es el deseado.

## PRÁCTICA 2

### Cuestión 8

Feu el codi que llig dos nombres x i y i obté per pantalla el valor de la suma x+y.

The screenshot shows the Mars MIPS simulator interface. The main window displays assembly code for 'Cuestion8.asm'. The code reads two integers from the user, stores them in registers \$t0 and \$t1, adds them, and prints the result to the console.

```
1
2 addi $v0,$zero,5      #leemos x
3 syscall
4 addi $t0,$v0,0         #guardamos el numero leído en $t0
5
6 addi $v0,$zero,5      #leemos y
7 syscall
8 addi $t1,$v0,0         #movemos y a $t1
9
10
11 add $a0,$t0,$t1       #almacenamos la suma de X e Y en a0
12 addi $v0,$zero,1      #imprimimos el numero por pantalla
13 syscall
14
```

Below the code editor, the 'Mars Messages' window shows the output of the program:

```
Reset: reset completed.

-- program is finished running (dropped off bottom) --

5
3
8
-- program is finished running (dropped off bottom) --
```

On the right side, the register window displays the state of the MIPS registers:

Name	Number	Value
\$zero	0	0
\$at	1	0
\$v0	2	1
\$v1	3	0
\$a0	4	8
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	5
\$t1	9	3
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	0
\$s1	17	0
\$s2	18	0
\$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		4194340
hi		0
lo		0

Ponemos \$v0 con valor 5 para leer un entero (X), lo guardamos en otra variable y volvemos a leer otro entero (Y). Utilizamos la instrucción addi para sumar los enteros X e Y, y la almacenamos en la variable \$a0. Por último, imprimimos el numero sumado por pantalla.

## Cuestion 12

Reescribiu el codi de partida Aritmètica d'enters de l'activitat 4 canviant addi o addiu per ori:

The screenshot shows a MIPS assembly editor with the following code in the main window:

```
1 #####
2 # #
3 # Codi de l'activitat 4 #
4 # Aritmètica d'enters #
5 # #
6 #####
7 .text 0x00400000
8 ori $t0, $zero, 25
9 ori $t1, $zero, 5
10 sub $t2, $t0, $t1
11 ori $v0, $zero, 10 #Eixir del programa
12 syscall
13
14
```

On the right, the Register window displays the state of the MIPS registers:

Name	Number	Value
\$zero	0	0
\$at	1	0
\$v0	2	10
\$v1	3	0
\$a0	4	0
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	25
\$t1	9	5
\$t2	10	20
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	0
\$s1	17	0
\$s2	18	0
\$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		4194324
hi		0
lo		0

Quin és el valor del codi d'operació de la instrucció ori?

El código de la instrucción en hexadecimal: 0x34080019

Código de la instrucción en binario: 0011 0100 0000 1000 0000 0000 0001 1001

El código de operación son los primeros 6 bits, es decir 001101

### Cuestión 13

Escriuiu el codi que faça l'operació lògica OR de \$t1 i \$t2 i el guardeu en \$t3, l'operació lògica AND de \$t1 i \$t2 i la guardeu en \$t4 i l'operació lògica XOR de \$t1 i \$t2 i la guardeu en \$t5. Escriuiu en la finestra de registres, després d'assemblat, els següents valors per als registres \$t1=0x55555555 i \$t2= 0xAAAAAA. Executa el codi i estudia els resultats.

The screenshot shows an assembly editor with three tabs: 'Cuestion8.asm', 'Cuestion12.asm', and 'cuestion13.asm'. The active tab 'cuestion13.asm' contains the following assembly code:

```
or $t3,$t1,$t2
and $t4,$t1,$t2
xor $t5,$t1,$t2
```

To the right of the code editor is a 'Registers' window displaying a table of registers and their values:

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x55555555
\$t2	10	0xaaaaaaaa
\$t3	11	0xffffffff
\$t4	12	0x00000000
\$t5	13	0xffffffff
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000

Como vemos en la OR, nos hace una suma de 5(\$t1) y 10(\$t2), la puerta AND al no ser iguales obtenemos un resultado de 0, y por último, en la puerta XOR al ser distintos \$t1 y \$t2, obtenemos la suma de estos, en caso de que tuvieran el mismo valor, el resultado sería 0.

### Práctica 3

#### Cuestión 2

Escribiu el codi (3 línies entre instruccions i pseudoinstruccions) que fa aquestes accions:

**\$t0=5**

**\$t1=\$t0+10**

**\$t2=\$t1-30**

The screenshot shows the MARS MIPS simulator interface. On the left, the assembly code for 'cuestion2.asm' is displayed:

```
1 li $t0,5
2 addi $t1,$t0,10
3 subi $t2,$t1,30
4
```

On the right, the 'Registers' window shows the current state of the MIPS registers:

Name	Number	Value
\$zero	0	0
\$at	1	30
\$v0	2	0
\$v1	3	0
\$a0	4	0
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	5
\$t1	9	15
\$t2	10	-15
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	0
\$s1	17	0
\$s2	18	0
\$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

Utilizamos la pseudo-instrucción li para cargar un inmediato en \$t0.

Y para las otras dos líneas utilizamos dos instrucciones para suma y resta.

**Assembleu i executeu el programa. Comproveu que el resultat final (\$t0=5, \$t1=15, \$t2=-30) és correcte.**

Registers		
Name	Number	Value
\$zero	0	0
\$at	1	30
\$v0	2	0
\$v1	3	0
\$a0	4	0
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	5
\$t1	9	15
\$t2	10	-15
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	0
\$s1	17	0

Como vemos los resultados obtenidos son los mismos salvo en \$t2, que según la instrucción, subi \$t2, \$t1,30, tendríamos que restar 30 a \$t1, con valor de 15, lo que nos daría -15. Ese es el resultado que nos da en el MARS aunque en el enunciado nos indica -30.



#### Cuestión 4

Amb la finalitat de facilitar la claredat en l'execució dels programes, modifiqueu el codi de la qüestió 3 perquè mostre '>' o '?' abans de llegir el caràcter del teclat com una manera de demanar-lo.

```
1
2 li $a0, '>'      #imprimimos el caracter >
3 li $v0, 11
4 syscall
5
6
7 li $v0, 12      # Funció 12. Read character
8 syscall        # Caràcter llegit en $v0
9
10 move $t0, $v0  # guardamos el caracter en la variable t0
11 li $a0, '\n'   # guardamos en a0 el caracter de salto de linea
12 li $v0, 11     # imprimimos el salto de linea
13 syscall
14 move $a0, $t0  # guardamos en a0 el caracter que queremos imprimir, almacenado en t0
15
16
17 li $v0, 11     # Funció 11. Print character
18 syscall
19 li $v0, 10     #Funció 10. Acaba programa
20 syscall
```

Usamos li para cargar en la variable \$a0 el carácter '>', después usamos el syscall 11 para imprimir el carácter en la consola. Una vez hecho esto nos mostrará el carácter, seguido de la introducción del dato que pedimos en el siguiente syscall.

Mars Messages	Run I/O
	>3
	3
	-- program is finished running --

## Cuestion 9

Itereu el codi que acabeu d'escriure. És a dir, utilitzeu la instrucció `j` perquè s'execute indefinidament

The screenshot shows the Mars MIPS simulator interface. The main window displays the assembly code for 'cuestion9.asm' with line numbers 1 through 13. The code is as follows:

```

1
2  etil:
3      li $v0,12      # LEEMOS CHARACTER
4      syscall
5
6      addi $v0,$v0,32 # AL CHARACTER LEIDO LE SUMAMOS 32
7      move $a0, $v0   # GUARDAMOS EL CHARACTER EN A0
8
9  eti2:
10     li $v0,11       # IMPRIMIMOS CHARACTER
11     syscall
12     j etil
13

```

Below the code editor, the 'Mars Messages' window shows the output: 'AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz' and 'Reset: reset completed.' A 'Clear' button is visible.

On the right side, the 'Registers' window displays the state of the MIPS registers. The 'Name' column lists registers from \$zero to \$ra, and the 'Value' column shows their current values. The 'PC' (Program Counter) is at 4194304.

Name	Number	Value
\$zero	0	0
\$at	1	0
\$v0	2	0
\$v1	3	0
\$a0	4	0
\$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	0
\$s1	17	0
\$s2	18	0
\$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		4194304
hi		0
lo		0

Lo que hacemos aquí es leer un carácter con valor de \$v0 en 12. El carácter introducido es guardado en \$v0, a esta variable le sumamos 32, ya que es la diferencia entre letras mayúsculas y minúsculas en la tabla ASCII. Imprimimos el carácter y hacemos un `j etil` lo que nos haría volver al principio del código y volveríamos a repetir esta operación.