

MEMORIA

Índice

1. PRÁCTICA 7 (p.2-5)
2. PRÁCTICA 8 (p.6-7)
3. PRÁCTICA 9 (p.8-10)
4. Ejercicios optativos (p.11-27)
 - Optativos Práctica 7 (p.11-19)
 - Optativos Práctica 8 (p.20-25)
 - Optativos Práctica 9 (p.26-27)

DATOS

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

PRÁCTICA 7

Vectores°

Entregable 7_1

Calcula la suma de los valores positivos y negativos del vector, dirección del cual se pasa como parámetro en \$a0 y la longitud en \$a1. La función devuelve en \$v0 la suma de los valores positivos y en \$v1 la suma de los negativos.

Para poder realizar este ejercicio hemos tenido que implementar la función sum en la cual hemos realizado los siguientes pasos:

Primero hemos inicializado a 0 el contador(\$s1) y el índice (\$t2) los cuales usaremos para recorrer el bucle, además de **inicializar a 0 el valor de \$v0** y guardar en \$s0 la dirección del vector para evitar problemas.

Dentro del bucle, hacemos uso de la beq para comparar el valor de nuestro contador y la longitud del bucle(\$a1), después guardamos en \$t1 el valor del vector leído con lw, para después usar slt para **comprobar si es positivo** el valor de \$t1.

Si es **positivo** (\$t0=0) entonces saltará a “positivo” donde se sumarán los valores positivos. Tanto al final de “loop” como de “positivo”, aumentaremos en 1 el contador y el índice además de aumentar en 4 la dirección del vector para leer el siguiente valor.

Una vez **fuera del bucle** usaremos jr \$ra para volver al programa principal

```
21 la $a0, msgC
22 syscall
23
24 li $v0, 1
25 move $a0, $v1 # Resultado 2 de la función
26 syscall # imprimir suma negativos
27
28 li $v0, 10 # Acabar programa
29 syscall
30
31 sum:
32     add $v0, $zero, $zero #Reseteamos el valor de $v0 a 0
33     move $s0, $a0 #Guardamos en $s0 la dirección del vector
34     add $s1, $zero, $zero #Iniciamos el contador a 0
35     li $t2, 0 #Iniciamos el índice a 0
36     loop:
37         beq $s1, $a1, exit #Mientras $s1 y $a1 sean distintos el bucle continuará
38         lw $t1, 0($s0) #Guardamos el valor del elemento en $t1
39         slt $t0, $t1, $zero #Comparamos si $t1 es menor que 0
40         beq $t0, $zero, positivo #Salta a positivo si $t0 vale 0
41
42         add $v1, $v1, $t1 #Sumamos los negativos
43
44         addi $t2, $t2, 1 #Incremento índice
45         addi $s1, $s1, 1 #Incrementamos el contador
46         addi $s0, $s0, 4 #Incrementamos la dirección del vector en 4 bytes
47         j loop
48     exit:
49         jr $ra
50     positivo:
51         add $v0, $v0, $t1 #Sumamos los positivos
52
53         addi $t2, $t2, 1 #Incremento índice
54         addi $s1, $s1, 1 #Incrementamos el contador
55         addi $s0, $s0, 4 #Incrementamos la dirección del vector en 4 bytes
56         j loop #volvemos al bucle
```

Line: 2 Column: 26 Show Line Numbers

Mars Messages Run I/O

La suma de los valores positivos es = 13
La suma de los valores negativos es = -5
-- program is finished running --

Entregable 7_2

Haz 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

En esta primera parte del programa, podemos observar como declaramos un **vector “matriz” vacío** con 16 espacios, además de guardar en memoria el tamaño de las columnas y filas (es 4x4) y la longitud del vector declarado junto a todos los mensajes que usaremos en el programa

Lo primero que hacemos es guardar la dirección de matriz en \$s0 y su longitud en \$t3, luego inicializamos el contador (\$t2) y el contador de columnas (\$t4) a 0 y escribimos el mensaje que explica como se van a introducir los datos.

Comenzamos el bucle y llamamos a la **etiqueta escribir**, la cual sirve para que el usuario introduzca el número por teclado y lo guardemos en \$t5, la cual junto a la **instrucción sw** lo guardaremos en la matriz. Después aumentamos en 1 los dos contadores que usamos y aumentamos de 4 en 4 la dirección del vector para introducir el siguiente valor.

Usando la instrucción **beq**, comparamos el valor del contador de \$t4 y \$t0 (número de columnas), cuando \$t4=4, entonces saltamos a **fila_completa** y notificamos al usuario que la primera fila ha sido completada. Y con la instrucción **bne**, hacemos que el bucle reitere hasta que \$t2 valga 16 (todas las posiciones están completadas)

Una vez **acaba el bucle** llamaremos a la etiqueta **exit** para comenzar la segunda parte del programa

```
1  .data
2      matriz: .word 0:16 #Vector vacío con 16 de capacidad
3      cf: .word 4 # Numero de columnas y filas
4      long: .word 16 #Longitud del vector
5      aviso: .asciiz "Los datos se introducen de izquierda a derecha, arriba hacia abajo"
6      mensaje: .asciiz "Introduce un entero --> "
7      fila: .asciiz "Fila completada"
8      resultado: .asciiz "La suma de los elementos de la diagonal principal introducida es: "
9      n: .asciiz "\n"
10     diagonal: .asciiz "Teniendo como diagonal principal: "
11  .text
12  la $s0, matriz
13  lw $t0, cf
14  lw $t3, long
15  li $t2, 0 #Contador iniciado a 0
16  li $t4, 0 #Contador de columnas
17  la $a0, aviso
18  li $v0, 4
19  syscall
20  la $a0, n
21  li $v0, 4
22  syscall
23  loop: jal escribir
24         sw $t5, 0($s0)
25         addi $t2, $t2, 1 #Aumentamos en 1 el indice
26         addi $t4, $t4, 1 #Aumentamos el contador de las columnas
27         addi $s0, $s0, 4
28         beq $t4, $t0, fila_completa
29         bne $t2, $t3, loop #Si no son iguales vuelve al loop
30         jal exit
31  escribir:
32         la $a0, mensaje
33         li $v0, 4
34         syscall
35         la $v0, 5
36         syscall
37         move $t5, $v0 #Guardamos el valor introducido en $t5
38         jr $ra
39  fila_completa:
40         la $a0, fila
41         li $v0, 4
42         syscall
43         la $a0, n
44         li $v0, 4
45         syscall
46         li $t4, 0 #Reseteamos el valor de $t4
47         bne $t2, $t3, loop
```

En la segunda parte del programa, volvemos a inicializar todas las variables necesarias para nuestro bucle “suma”, dirección de la matriz ya rellena en \$t0, longitud del mismo en \$t3 y esta vez usaremos como contadores \$t2 como contador y \$t5 como índice.

A su vez inicializamos 4 valores, que son las correspondientes posiciones de la **diagonal principal** de una matriz 4x4.

Al **empezar el bucle** usaremos la **instrucción lw**, para guardar en \$t1 el valor leído en esa posición.

Después usaremos la **instrucción beq**, para comprobar el contador del vector con las respectivas posiciones de la diagonal principal. En caso de coincidir, se llamará a la **etiqueta “op”**, donde se guardará en \$t9 la suma de estos elementos además de imprimir con la etiqueta “imprimir” los elementos que conforman la diagonal principal.

Otra vez haremos uso de la instrucción bne, para poder leer todas posiciones del vector “matriz”. Al **finalizar el bucle** llamaremos a la **etiqueta “res”** en la cual después de un salto de línea imprimiremos el mensaje correspondiente y el resultado obtenido y guardado en \$f9.

```
48 exit:
49 la $t0, matriz #Guardamos la dirección de la matriz
50 lw $t3, long #Guardamos en $t3 el numero de columnas
51 li $t2, 0
52 li $t5, 0
53 li $s1, 1
54 li $s2, 6
55 li $s3, 11
56 li $s4, 16
57 la $a0, diagonal
58 li $v0, 4
59 syscall
60 suma:
61 lw $t1, 0($t0)
62
63 addi $t2, $t2, 1 #Aumentamos en 1 el contador
64 addi $t5, $t5, 1
65 addi $t0, $t0, 4
66 beq $t2, $s1, op
67 beq $t2, $s2, op
68 beq $t2, $s3, op
69 beq $t2, $s4, op
70 bne $t2, $t3, suma #Si no son iguales vuelve al loop
71 jal res
72 li $v0, 10
73 syscall
74 op:
75 add $t9, $t9, $t1 #Guardamos la suma de la diagonal en t9
76 move $a0, $t1
77 jal imprimir
78 bne $t2, $t3, suma
79 res:
80 la $a0, n
81 li $v0, 4
82 syscall
83 la $a0, resultado
84 li $v0, 4
85 syscall
86 move $a0, $t9
87 li $v0, 1
88 syscall
89 li $v0, 10
90 syscall
```

Por último, mostraremos en esta parte un ejemplo de ejecución del programa, así como la etiqueta imprimir, la cual no había suficiente espacio para mostrar anteriormente:

Etiqueta imprimir:

```

91  imprimir:
92      move $a0, $t1
93      li, $v0, 1
94      syscall
95      li $a0, ' '
96      li $v0, 11
97      syscall
98      jr $ra
99

```

Ejecución:

The screenshot displays the Mars MIPS simulator interface. The 'Text Segment' window shows the assembly code for the 'imprimir' label, which prints a space character and a newline character. The 'Data Segment' window shows the memory layout, including the stack and data segments. The 'Mars Messages' window shows the program's output, which includes prompts for input, completion of rows, and the final sum of the main diagonal.

Text Segment

Bkpt	Address	Code	Basic	Source
	4194304	0x3c011001	lui \$1,4097	12: la \$s0, matriz
	4194308	0x34300000	ori \$16,\$1,0	
	4194312	0x3c011001	lui \$1,4097	13: lw \$t0, cf
	4194316	0x8c280040	lw \$8,\$4(\$1)	
	4194320	0x3c011001	lui \$1,4097	14: lw \$t3, long
	4194324	0x8c2b0044	lw \$11,\$8(\$1)	
	4194328	0x240a0000	addiu \$10,\$0,0	15: li \$t2,0 #Contador iniciado a 0
	4194332	0x240c0000	addiu \$12,\$0,0	16: li \$t4,0 #Contador de columnas
	4194336	0x3c011001	lui \$1,4097	17: la \$a0, aviso
	4194340	0x34240048	ori \$4,\$1,72	
	4194344	0x24020004	addiu \$2,\$0,4	18: li \$v0, 4
	4194348	0x0000000c	syscall	19: syscall
	4194352	0x3c011001	lui \$1,4097	20: la \$a0, n
	4194356	0x342400f7	ori \$4,\$1,247	
	4194360	0x24020004	addiu \$2,\$0,4	21: li \$v0, 4
	4194364	0x0000000c	syscall	22: syscall
	4194368	0x0c100018	jal 4194400	23: loop: jal escribir

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	10	3	4	6	8	5	5	3
268501024	0	0	20	0	5	7	8	5
268501056	4	16	544436044	1869898084	1702043763	1953392928	1969516402	544105827
268501088	1763730788	1769304442	1633972837	1679843616	1667592805	539779432	1769108065	1746952546
268501120	1634296673	1633837344	1224765290	1869771886	1701016932	544109856	1702129253	757100402
268501152	2113069	1634494790	1836016416	1952803952	6382689	1931501900	543255925	1814062436
268501184	1696625516	1701668204	1936681142	543515680	1679843692	1869046121	5439732712	1852404336

Mars Messages

```

Introduce un entero --> 4
Introduce un entero --> 6
Fila completada
Introduce un entero --> 8
Introduce un entero --> 5
Introduce un entero --> 5
Introduce un entero --> 3
Fila completada
Introduce un entero --> 0
Introduce un entero --> 0
Introduce un entero --> 20
Introduce un entero --> 0
Fila completada
Introduce un entero --> 5
Introduce un entero --> 7
Introduce un entero --> 8
Introduce un entero --> 5
Fila completada
Teniendo como diagonal principal: 10 5 20 5
La suma de los elementos de la diagonal principal introducida es: 40
-- program is finished running --

```

PRÁCTICA 8

Coma flotante 1

Entregable 8_1

Completa el siguiente código de partida que pide el radio por teclado y tiene que calcular y mostrar en la consola la longitud de la circunferencia y el área del círculo.

Para poder realizar este ejercicio entregable hemos tenido que implementar dos funciones llamadas calculaLog y calculaArea. Como podemos observar en la imagen de la derecha, ambas funciones se han implementado usando **mul.s** para multiplicar entre floats. Especial mención en calculaLog, debido que para implementar el valor de 2 para poder aplicar la fórmula de la longitud hemos usado la herramienta de Mips para obtener 0x40000000(podríamos a ver usador las funciones mtc1 y cvt.s.w como podremos ver en el entregable 8_2) y poder pasarlo a flotante

```
1 .data
2     demanaPi : .asciiz "Dame el valor de pi..."
3     pideRadio: .asciiz "Dame el radio..."
4     longr:    .asciiz "Longitud de la circunferencia = "
5     super:    .asciiz "Área del círculo = "
6     n:        .asciiz "\n"
7 .text
8     li $v0,4
9     la $a0,demanaPi
10    syscall
11    li $v0,6
12    syscall
13
14    mov.s $f1,$f0 #Pi en $f1 y Radio en $f0
15    li $v0,4
16    la $a0,pideRadio
17    syscall
18    li $v0,6
19    syscall
20    #Calculamos la longitud
21    li $v0,4
22    la $a0,longr
23    syscall
24    jal calculaLog #Llamamos a la función para calcular la longitud
25    li $v0,4
26    la $a0,n
27    syscall
28    #Calculamos el area
29    li $v0, 4
30    la $a0, super
31    syscall
32    jal calculaArea #Llamamos a la función para calcular el area
33
34    li $v0,10 #Finalizar programa
35    syscall
36
37    calculaLog:#L= 2*pi*radio
38    li $s0, 0x40000000
39    mtc1 $s0, $f2
40    mul.s $f4,$f0,$f1 #Guardamos en $f4 el resultado de radio*pi
41    mul.s $f12,$f4,$f2 #Multiplicamos por 2 y guardamos a $f5
42    #Guardamos n $f12 para poder imprimirlo
43    li $v0, 2 #Imprimimos el float
44    syscall
45    jr $ra #Volvemos al programa
46
47    calculaArea:#A= pi*r^2
48    mul.s $f4,$f0,$f0 #Multiplicamos radio*radio
49    mul.s $f12,$f4,$f1 #Multiplicamos pi*$f4(radio^2)
50    #Guardamos n $f12 para poder imprimirlo
51    li $v0, 2 #Imprimimos el float
52    syscall
53    jr $ra
```

Line: 34 Column: 30 ☒ Show Line Numbers

Mars Messages Run I/O

```
Dame el valor de pi...3.14
Dame el radio... 5
Longitud de la circunferencia = 31.400002
Área del círculo = 78.5
-- program is finished running --
```

Line: 34 Column: 30 ☒ Show Line Numbers

Mars Messages Run I/O

```
Dame el valor de pi...3.14
Dame el radio... 5
Longitud de la circunferencia = 31.400002
Área del círculo = 78.5
-- program is finished running --
```

Entregable 8_2

Haz el código que suma los elementos del vector y calcula el valor medio en coma flotante. Muestra el resultado por la consola

Para poder realizar este ejercicio, (a parte de los `.ascii` para mostrar mensajes) hemos tenido que realizar varios pasos.

El **primer paso** ha sido guardar la dirección del vector array dado además de su tamaño haciendo uso de `lw`. Después hemos inicializado a 0 el contador de nuestro bucle “loop” y el índice que usaremos para recorrer el vector.

Una vez **dentro del bucle**, hacemos uso de `beq`, para comprobar que el contador(`$s1`) no es igual al tamaño del vector(`$s5`), mientras sean diferentes el bucle continuará. Mientras realizamos el bucle guardamos en `$s6` el valor de la suma que se va acumulando, después aumentamos en 1 el valor del contador y del índice, además de, **importante**, aumentar en 4 bytes la dirección de array(`$s0`) para poder leer la siguiente posición

Una vez **fuera del bucle** mostramos el resultado obtenido en `$s6` y llamamos a la función `avg`, donde calculamos el valor medio.

En la **función avg**, sencillamente implementaremos las reglas `mtcl` y `cvt.s.w` para poder “mover” a Coproc 1(líneas 40-43) el valor de `$s5` y `$s6` para realizar la operación tipo float: `div.s` para dividir la suma total (`$s6`→`$f1`) entre el total de elementos (`$s5`→`$f0`) para luego mostrar el resultado obtenido en `$f12` usando un `syscall`

```

1 .data
2     array: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
3     loopr: .word 10
4     suma: .word 0
5
6     r.suma: .ascii "La suma de los elementos es: "
7     r.media: .ascii "Obtenemos un valor medio de: "
8     nr: .ascii "\n"
9
10 .text
11 la $s0, array #Dirección del vector array
12 lw $s5, loopr #Guardamos en $s5 el tamaño del vector
13 add $t1, $zero, $zero #Iniciamos el contador a 0
14 li $t2, 0 #Iniciamos índice para recorrer el vector
15
16 loop:#Bucle para obtener la suma de los elementos del vector array
17     beq $t1, $s5, exit #Si $t1 y $s5 son iguales salimos del bucle
18     lw $t0, 0($s0)#Guardamos el valor del elemento en $t0
19     move $a0, $t0
20
21     add $s6, $s6, $t0 #Sumamos
22
23     addi $t2, $t2, 1 #Incremento índice
24     addi $t1, $t1, 1 #Incrementamos el contador
25     addi $s0, $s0, 4 #Aumentamos 4 bytes la dirección para leer el siguiente valor
26     j loop
27
28 exit:
29     la $a0, r.suma
30     li $v0, 4
31     syscall
32     move $a0, $s6 #Imprimimos el valor de la suma obtenido después de acabar el bucle
33     li $v0, 1
34     syscall
35
36 .globl avg
37
38 avg:
39     mtcl $s5, $t5
40     mtcl $s6, $t6
41     cvt.s.w $f0, $t5 #Guardamos en $f0 el valor de $s5, después de convertirlo en entero
42     cvt.s.w $f1, $t6 #Guardamos en $f1 el valor de $s6, después de convertirlo en entero
43     la $a0, r.media
44     li $v0, 4
45     syscall
46     div.s $f12, $f1, $f0
47     li $v0, 2
48     syscall
49     jr $ra
50
51

```

Line: 14 Column: 1 Show Line Numbers

Mars Messages Run I/O

```

La suma de los elementos es: 61
Obtenemos un valor medio de: 6.1
-- program is finished running --

La suma de los elementos es: 55
Obtenemos un valor medio de: 5.5
-- program is finished running --

```

Line: 51 Column: 2 Show Line Numbers

Mars Messages Run I/O

```

La suma de los elementos es: 61
Obtenemos un valor medio de: 6.1
-- program is finished running --

La suma de los elementos es: 55
Obtenemos un valor medio de: 5.5
-- program is finished running --

```

PRÁCTICA 9

Coma flotante 2

Entregable 9_1

Haz el código que suma los elementos del vector y calcula el valor medio. Muestra el resultado por la consola.

Para realizar este entregable, lo primero que hacemos antes de empezar el bucle es inicialiar a 0 el contador (\$t1) y el índice (\$t0) que usaremos, además de guardar en \$s0, la dirección del array y en \$s1, la longitud del mismo.

Una vez comenzamos el bucle usaremos la instrucción `lwcl`, para guardar en \$f0, el valor de un vector tipo float, después usaremos la variable \$f5, para guardar la suma total de los elementos del vector con la función `add.s`.

Una vez aumentamos el contador y el índice en 1, además de aumentar en 4 la dirección del vector para poder leer el siguiente valor. Al final del bucle usaremos la instrucción `bne`, la cual repetirá el bucle hasta que $t1 = s1$.

Al finalizar el bucle guardaremos la longitud del vector en \$f1 y procederemos a dividir con la instrucción `$div.s $f5` (La suma total) y \$f4 (Longitud del vector en float).

Por último moveremos el valor de la división a \$f12 para poder imprimirla después de su mensaje correspondiente

```
1  .data
2      Array: .float 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
3      long: .word 10
4      Suma: .float 0
5      msg: .asciiz "El valor medio del vector Array es: "
6      n: .asciiz "\n"
7  .text
8      la $s0, Array
9      lw $s1, long
10     add $t1, $zero, $zero #Inicilizamos el contador a 0
11     add $t2, $zero, $zero #Inicilizamos el indice a 0
12 loop:
13     lwcl $f0, 0($s0) #Guardamos en $f0 el valor del vector
14
15     add.s $f5, $f5, $f0 #Guardamos la suma total en $f5
16
17     addi $t1, $t1, 1
18     addi $t2, $t2, 1
19     addi $s0, $s0, 4
20
21     bne $t1, $s1, loop
22     mtcl $s1, $f1 #Guardamos en $f1 el valor de $s1
23     cvt.s.w $f4, $f1 #Contertimos a entero y guardamos en $f4
24
25     div.s $f9, $f5, $f4 #Dividimos el resultado y lo guardamos en $f9
26
27     la $a0, msg
28     li $v0, 4
29     syscall
30     mov.s $f12, $f9 #Movemos a $f12 para poder imprimir el resultado
31     li $v0, 2
32     syscall
33     li $v0, 10
34     syscall
35
```

Line: 35 Column: 1 ☒ Show Line Numbers

Mars Messages Run I/O

El valor medio del vector Array es: 5.5
-- program is finished running --

Entregable 9_2

Implementar la función `float pow(float x;int n)` que calcula la potencia n -ésima de x . Los argumentos y los valores se pasan según convenio: x en $\$f12$, n en $\$a0$. El resultado se devuelve en $\$f0$. Utilizad el siguiente código de partida:

Para poder realizar este programa, lo primero que tenemos que hacer es usar la instrucción `move` para mover a $\$s1$ las n iteraciones que tiene que hacer nuestro bucle y evitar problemas con $\$a0$, inicializamos a 0 el contador y $\$t5$ a 1, que será el registro que usaremos para guardar las multiplicaciones que se hagan para después moverlo a $\$f5$ con las instrucciones `mtc1` y `cvt.s.w`. (Convertimos de coma flotante a word para poder usarlo)

Estando en el bucle lo único que hacemos es usar la instrucción `mul.s` para ir acumulando el valor en $\$f5$ y usar la instrucción `bne` después de aumentar en 1 el contador, para que el bucle reitere en el caso de que el contador ($\$t0$), sea distinto de las iteraciones introducidas ($\$s1$).

Una vez acabamos el bucle, movemos el valor de $\$f5$ a $\$f0$, para que el programa que ha sido dado en el enunciado de la práctica lo pueda imprimir.

```
1  .data
2      Xpide:.asciiz "X = "
3      Npide:.asciiz "n = "
4      powRes:.asciiz "X^n = "
5  .text
6  la $a0, Xpide
7  li $v0,4
8  syscall
9  li $v0,6
10 syscall
11
12 la $a0, Npide
13 li $v0,4
14 syscall
15 li $v0,5
16 syscall
17
18 mov.s $f12,$f0
19 move $a0,$v0
20 jal pow
21 la $a0,powRes
22 li $v0,4
23 syscall
24 mov.s $f12,$f0
25 li $v0,2
26 syscall
27 li $v0,10
28 syscall
29
30 pow:
31     move $s1, $a0 #Movemos a $s1 para usarlo en el bucle(n veces)
32     li $t0, 0 #Inicializamos a 0 el contador
33     li $t5, 1 #Inicializamos a 1 la variable donde vamos a guardar las mults
34     mtc1 $t5, $f5
35     cvt.s.w $f5, $f5
36     loop:
37         mul.s $f5, $f5, $f0 #Multiplicamos X por si mismo y guardamos el resultado
38         #en $f5
39         addi $t0, $t0, 1
40         bne $t0, $s1, loop
41         mov.s $f0, $f5
42     jr $ra
43
Line: 43 Column: 1 Show Line Numbers
Mars Messages Run I/O
Clear
X = 5
n = 3
X^n = 125.0
-- program is finished running --
```

Entregable 9_3

Implementar la función max que nos devuelve el valor mayor de dos números en coma flotante. Los argumentos se pasan según convenio en \$f12 y \$f14 y el resultado se devuelve en \$f0. Utilizad el siguiente código de partida:

Para poder realizar la función max, tendremos que hacer uso de la instrucción c.le.s la cual compara entre dos elementos. En el caso de que \$f1 (valor X) sea menor o igual a \$f2 (valor Y), entonces el flag vale 1.

Para poder distinguir el valor del flag usaremos bclt, para saltar a la etiqueta X_menor, lo cual significará que flag valía 1, en caso contrario no saltará a la etiqueta. En ambos casos fácilmente implementamos la función mov.s para mover a \$f0, el valor que mayor correspondiente

```
1  .data
2      Xpide:.asciiz "X = "
3      Ypide:.asciiz "Y = "
4      MaxRes:.asciiz "El mayor es "
5      n:.asciiz "\n"
6  .text
7      la $a0, Xpide
8      li $v0,4
9      syscall
10     li $v0,6
11     syscall
12     mov.s $f1, $f0#Guardamos en $f1 el valor de X
13     mov.s $f12,$f0
14
15     la $a0, Ypide
16     li $v0,4
17     syscall
18     li $v0,6
19     syscall
20     mov.s $f2, $f0#Guardamos en $f2 el valor de Y
21     mov.s $f14,$f0
22
23     jal max
24
25     la $a0,MaxRes
26     li $v0,4
27     syscall
28     mov.s $f12,$f0
29     li $v0,2
30     syscall
31
32     li $v0,10
33     syscall
34
35     max:#El valor de X está en $f1 y el valor de Y está en $f2
36     c.le.s $f1, $f2 #Comparamos si $f1 <= $f2
37     #Si flag a 1 --> $f1 es menor(X es menor)
38     bclt X_menor
39     mov.s $f0, $f1
40     jr $ra
41 X_menor:
42     mov.s $f0, $f2
43     jr $ra
..
```

Mars Messages	Run I/O
	X = 5 Y = 35 El mayor es 35.0 -- program is finished running --
Clear	X = 10.10 Y = 10 El mayor es 10.1 -- program is finished running --

Ejercicios optativos Práctica 7

Ensambla y ejecuta el código. ¿Cuántos caracteres tiene la cadena?

The screenshot shows the MARS MIPS simulator. The assembly code in the main window is as follows:

```
#Contar caracteres de una cadena
.data
str:.ascii "Estructuras de los"
      .ascii "Computadores" #Tiene '\0' al final
.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 # Y si no incrementamos el contador
    j loop
exit: li $v0, 10
      syscall
```

The register window on the right shows the state of the registers. The register `$s1` is highlighted with a green box and contains the value 30. The register `$t0` contains 268501022, and `$s0` contains 268500992. The status bar at the bottom indicates "Line: 16 Column: 1" and "Show Line Numbers". The "Mars Messages" window at the bottom shows the message "-- program is finished running --".

➔ Como podemos observar tiene 30 caracteres

¿En qué dirección de la memoria se encuentra el carácter null?)

➔ En 1e

¿Cómo se actualiza el índice del vector?

➔ En `$s1` y usando la instrucción `addi $s1, $s1, 1` para ir aumentando de 1 en 1

¿El programa funcionaría si la cadena solo constara del carácter null?

➔ Sí que funcionaria y `$s1` valdría 0

Modifica el código para que muestre por pantalla el mensaje “El número de caracteres de la cadena es:” y a continuación el resultado.

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

```
.data
str:.ascii "Estructuras de los"
      .asciiz "Computadores" #Tiene '\0' al final
cad:
      .ascii "El número de caracteres 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 # Y si no incrementamos el contador
j loop
exit:
la $a0, cad
li $v0, 4
syscall

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

li $v0, 10
syscall
```

Below the code editor, the status bar indicates "Line: 26 Column: 1" and a checkbox for "Show Line Numbers".

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

\$zero	0	0
\$at	1	268500992
\$v0	2	10
\$v1	3	0
\$a0	4	30
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	268501022
\$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	30
\$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		4194372
hi		0
lo		0

At the bottom, the "Mars Messages" window shows the output of the program:

```
-- program is finished running --
El número de caracteres de la cadena es: 30
-- program is finished running --
```

Modifica el código para que calcule el número de veces que se repite la vocal “u”.

Parte del .data y el bucle

```
.data
str:.ascii "Estructuras de los"
    .asciiz "Computadores" #Tiene '\0' al final
cad1:
    .asciiz "El número de caracteres de la cadena es: "
cad2:
    .asciiz "El número de veces que se repite la letra 'u' es: "
n: .asciiz "\n" #Salto de línea
.text
la $s0, str
add $s1, $zero, $zero # Iniciamos contador a 0
li $s2, 0 #Contador de u
li $s3, 'u'
loop:
    add $t0, $s0, $s1 # dirección del byte a examinar
    lb $t1, 0( $t0 )

    bne $t1, $s3, no_es_u
    addi $s2, $s2, 1

no_es_u:

    beq $t1, $zero, exit # Salimos si carácter leído='\0'
    addi $s1, $s1, 1 # Y si no incrementamos el contador

    j loop
```

Parte del código que se activa cuando acaba el bucle (exit)

```
    j loop

exit:
la $a0, cad1
li $v0, 4
syscall


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

la $a0, n #Salto de línea
li $v0, 4
syscall

la $a0, cad2
li $v0, 4
syscall

move $a0, $s2
li $v0, 1
syscall

li $v0, 10
syscall
```



Ensambla y ejecuta el programa. Comprueba que el vector A se copia en el vector B.

```
#Ejemplo: recorrer un vector de enteros?
.data
A: .word 2, 4, 6, 8, 10 # vector A iniciado con valores
B: .word 0:5 # Vector B vacío
C: .space 50 # Otra definición de vector vacío(Reserva 50 bytes)

.text
la $s0, A # Dirección base del vector A
la $s1, B # Dirección base del vector B
li $s5, 5 # Tamaño del vector
loop:
add $t1, $s0, $t0
add $t2, $s1, $t0
addi $s2, $s2, 1 # Índice del vector

lw $t3, 0($t1)
sw $t3, 0($t2)

sll $t0, $s2, 2 # Índice del vector x4
bne $s2, $s5, loop #si no es igual a 5 lo repite

li $v0, 10
syscall
```

Line: 26 Column: 1 ☐ Show Line Numbers

Mars Messages

Run I/O

-- program is finished running --

¿En qué dirección empieza el vector B?

➔ En value (+14)

¿Porque no se pueden acabar los vectores con el carácter null igual que se hace con las cadenas de caracteres?

➔ Los vectores no son caracteres, no tienen que ser bytes

En el programa se recorren los vectores actualizando el índice con la instrucción sll. ¿De qué otra manera se podrían recorrer los vectores?

➔ Sumando de 4 en 4

Se ha utilizado un bucle del tipo do-while, modifica el programa por que el bucle sea de tipo for-while.

```
.data
A: .word 2, 4, 6, 8, 10 # vector A iniciado con valores
B: .word 0:5 # Vector B vacío
C: .space 50 # Otra definición de vector vacío(Reserva 50 bytes)

.text
la $s0, A # Dirección base del vector A
la $s1, B # Dirección base del vector B
li $s5, 5 # Tamaño del vector
loop:
beq $s2, $s5, exit

add $t1, $s0, $t0
add $t2, $s1, $t0
addi $s2, $s2, 1 # Índice del vector

lw $t3, 0($t1)
sw $t3, 0($t2)
j loop
exit:
li $v0, 10
syscall
```

Modifica el programa 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 0:5 # Vector B vacío
C: .space 50 # Otra definición de vector vacío(Reserva 50 bytes)
mensaje: .asciiz "Introduce un entero--> "
n: .asciiz "\n"

.text
la $s0, A # Dirección base del vector A
la $s1, B # Dirección base del vector B
li $s5, 5 # Tamaño del vector

loop:
la $a0, mensaje
li $v0, 4
syscall

la $v0, 5
syscall

move $t3, $v0 #Movemos a $t3 el valor introducido en $v0
la $a0, n
li $v0, 4
syscall

add $t1, $s0, $t0
add $t2, $s1, $t0
addi $s2, $s2, 1 # Índice del vector

#lw $t3, 0($t1) No hace falta los valores de A
sw $t3, 0($t2)

sll $t0, $s2, 2 # Índice del vector x4
bne $s2, $s5, loop #si no es igual a 5 lo repite

li $v0, 10
syscall
```

Como podemos observar se guardan bien los datos:

dress	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	2	4	6	8	10	12	14	16
0x10010020	2	4	6	8	10	12	14	16
0x10010040	0	0	0	0	0	0	1850277888	1685025396
0x10010060	543515509	1696624245	1919251566	1043148143	655392	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


```
-- program is finished running --

Introduce un entero--> 6

Introduce un entero--> 7

Introduce un entero--> 8

Introduce un entero--> 9

Introduce un entero--> 2
```


Completa el programa para que se rellene el vector C con la suma de los elementos del vector A y del B ($C[y]=A[y]+B[y]$).?

```

1  .data
2  A: .word 2, 4, 6, 8, 10 # vector A iniciado con valores
3  B: .word 0:5 # Vector B vacío
4  C: .space 50 # Otra definición de vector vacío(Reserva 50 bytes)
5  mensaje: .asciiz "Introduce un entero--> "
6  n: .asciiz "\n"
7
8  .text
9  la $s0, A # Dirección base del vector A
10 la $s1, B # Dirección base del vector B
11 la $s3, C # Dirección base del vector C
12 li $s5, 5 # Tamaño del vector
13 loop:
14 la $a0, mensaje
15 li $v0, 4
16 syscall
17
18 la $v0, 5
19 syscall
20
21 move $t3, $v0 #Moveremos a $t3 el valor introducido en $v0
22
23 la $a0, n
24 li $v0, 4
25 syscall
26
27 add $t1, $s0, $t0
28 add $t2, $s1, $t0
29 add $t6, $s3, $t0
30
31 lw $t4, 0($t1) #Guardamos el valor de la posición de A en $t4
32 add $t7, $t4, $t3
33
34 sw $t7, 0($t6)
35
36 addi $s2, $s2, 1 #Índice 0 1 2....
37 sll $t0, $s2, 2 # Índice del vector x4
38 bne $s2, $s5, loop #si no es igual a 5 lo repite
39
40 li $v0, 10
41 syscall

```

Como podemos observar hemos obtenido los resultados esperados

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	2	4	6	8	10	0	0	0
0x10010020	0	0	10	10	10	14	20	0
0x10010040	0	0	0	0	0	0	1850277889	1685025396
0x10010060	543515509	1696624245	1919251566	1043148143	655392	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
0x100101a0	0	0	0	0	0	0	0	0
0x100101c0	0	0	0	0	0	0	0	0

Mars Messages	Run I/O
Introduce un entero--> 8	
Introduce un entero--> 6	
Introduce un entero--> 4	
Clear	Introduce un entero--> 6
Introduce un entero--> 10	

En el siguiente ejemplo se muestra la utilización de los distintos modos de direccionamiento:

```
1  # Ejemplo de direccionamiento
2  .data
3  A:.word 6
4  B:.word 8
5  C:.space 4
6  .text
7  la $t0,A # En $t0 la dirección de A
8  lw $t1,0($t0) # Direccionamiento indirecto (dirección en $t0)
9  lw $t2,4($t0) # Direccionamiento relativo (dirección = $t0+4)
10 add $t3,$t1,$t2
11 sw $t3,C # Direccionamiento absoluto (dirección =C)
12
```

¿Cuántas pseudoinstrucciones contiene el código?

→ Hay dos pseudoinstrucciones la y sw

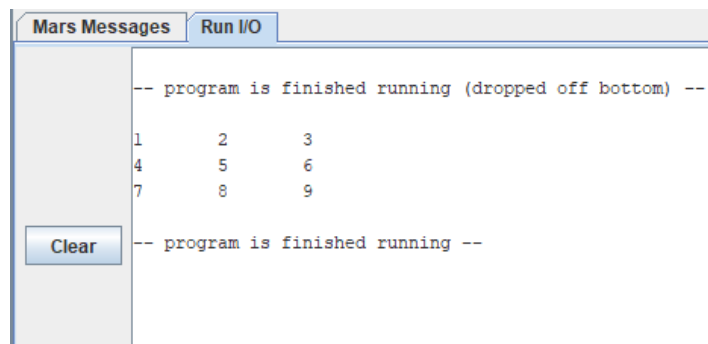
Ensambla el código y observa la traducción de las pseudoinstrucciones en instrucciones del MIPS. ¿En qué instrucciones se ha traducido sw \$t3, C?

→ En lui y sw

¿Qué registro auxiliar se ha utilizado?

→ El \$at

```
3  #Imprimir matriz transpuesta
4  .data
5  matriz: .byte 1, 4, 7,
6           2, 5, 8
7           3, 6, 9
8  columnas: .word 3 # Numero de columnas
9  .text
10 la $t0, matriz
11 lw $t3, columnas
12 li $t2, 0 #iniciamos indice para recorrer matriz
13
14 bucle:
15 lb $t1, 0($t0) #Fila 0
16 move $a0, $t1
17 jal imprimir
18
19 lb $t1, 3($t0) #Fila 1 (1*3elementos)
20 move $a0, $t1
21 jal imprimir
22
23 lb $t1, 6($t0) #Fila 2 (2*3elementos)
24 move $a0, $t1
25 jal imprimir
26
27 jal nuevalin
28
29 addi $t2, $t2, 1 # incremento indice
30 addi $t0, $t0, 1 #nueva columna
31 bne $t2, $t3, bucle
32 li, $v0, 10
33 syscall
34
35 imprimir: li, $v0, 1
36 syscall
37 li $a0, '\t'
38 li $v0, 11
39 syscall
40 jr $ra
41 nuevalin: li $a0, '\n'
42 li, $v0, 11
43 syscall
44 jr $ra
```



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

1      2      3
4      5      6
7      8      9

-- program is finished running --
```

Identifica los distintos modos de direccionamiento en memoria utilizados: direccionamiento absoluto, direccionamiento indirecto y direccionamiento

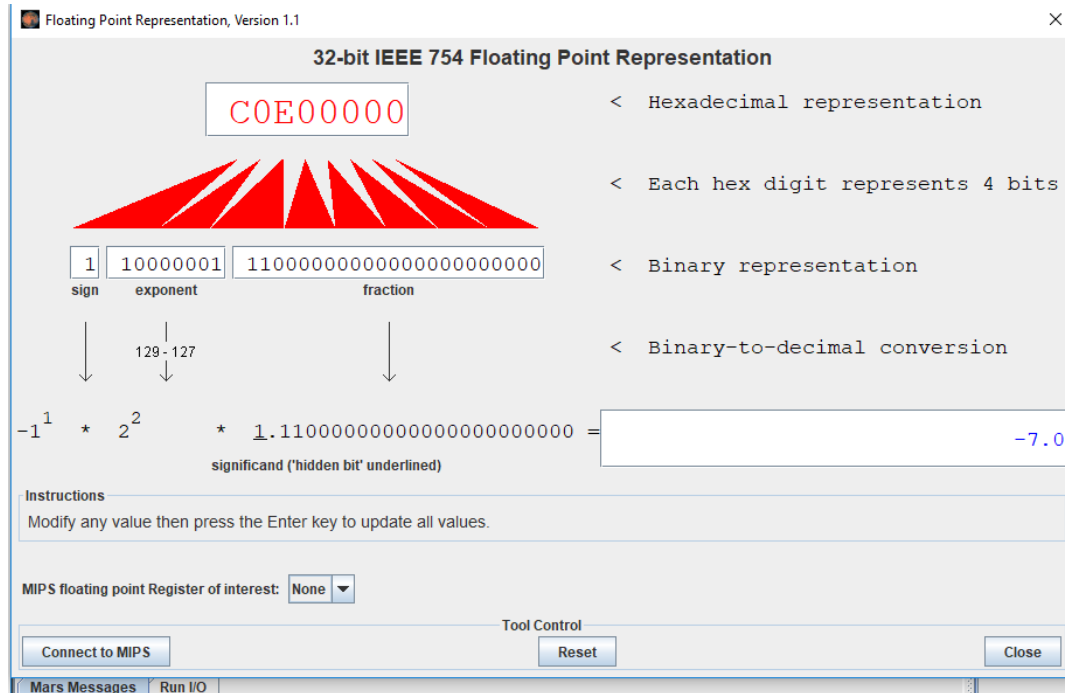
Absoluto: `lw $t3, columnas`

Indirecto: `lb $t1, 0($t0)`

Relativo: `lb $t1, 3($t0)` y `lb $t1, 6($t0)`

Ejercicios optativos Práctica 8

Comprobad que el número decimal de C0E00000 es -7.0 y que la representación en 32 bits de simple precisión de 4.25 es 40880000



EJEMPLO1

```
1 # Código ejemplo 1.
2 li $t0, 0xFF800000    # Menos infinito
3 mtc1 $t0, $f12        # movemos $t0 -> $f12=0xFF800000
4 li $t1, 0x7F8003A0    # Not a Number (NaN)
5 mtc1 $t1, $f20        # movemos $t1 -> $f20=0x7F8003A0
6
7
```

Haciendo uso de la herramienta de representación en coma flotante del MARS,

Comprueba que realmente en \$f12 hay un $-\infty$ y en \$f20 un NaN según el estándar IEEE 754.

→ 0xff800000: $-\infty$

→ 0x7f8003a0: NaN

Haz que el contenido de \$f1 sea el valor 1 en coma flotante y el de \$f2 el valor 2.5 en coma flotante.

```
li $t0, 0x3F800000    # 1
mtcl $t0, $f1         # movemos $t0 -> $f12=0xFF800000
li $t1, 0xC0200000    # -2.5
mtcl $t1, $f2         # movemos $t1 -> $f20=0x7F8003A0
```

\$f1	1.0	
\$f2	-2.5	1.592532974E-314
\$f3	0.0	
\$f4	0.0	0.0
\$f5	0.0	

Di una manera de escribir un 0.0 en \$f0 con sólo una instrucción máquina

```
1 mtcl $0, $f0 #Primera forma, Implementando 0 directamente
2
3 sub.s $f0, $f0, $f0 #Segunda forma, restando un registro con el mismo para
4 #obtener 0
5
```

¿Por qué no hay registros HI y LO para guardar el resultado de la multiplicación y división en coma flotante del mismo modo que con los números enteros?

Porque las multiplicaciones realizadas en IE³ conservan el formato IE³, en cambio cuando se multiplica sin este formato devuelve un resultado que es el doble de tamaño

EJEMPLO2

Haciendo uso de la herramienta de representación en coma flotante del MARS, comprueba que realmente en \$f0 después de la conversión hay un -8.

→ 0xc1000000: -8.0

¿Qué valor consideraría la máquina que habría en \$f0 si no hiciésemos la conversión con la instrucción cvt.s.w? Aprovecha que puedes ver contenidos en decimal y en hexadecimal.

→ Aparecería un NaN

Haz que el contenido de \$f1 sea el valor 1 en coma flotante y el de \$f2 el valor -2 en coma flotante utilizando las instrucciones de conversión de tipo.

```
1  # Código ejemplo 2. Conversión de tipo
2
3  li $s0, 1
4  mtcl $s0, $f1
5  cvt.s.w $f1, $f1
6
7  li $s1, -2
8  mtcl $s1, $f2
9  cvt.s.w $f2, $f2
10
```

Name	Float	Double
\$f0	0.0	0.0078125
\$f1	1.0	
\$f2	-2.0	1.591496843E-314
\$f3	0.0	
\$f4	0.0	0.0
\$f5	0.0	
\$f6	0.0	0.0
\$f7	n n	

EJEMPLO3

Ensambla el código y comprueba el resultado.

```
1  # Código ejemplo 3
2
3  li $s0, 841242345
4  mtcl $s0, $f0 # movemos de $s0 a $f0
5  cvt.s.w $f1, $f0 # Conversión de entero-simple precisión
6  cvt.w.s $f1, $f1 # Conversión de simple precisión-entero
7  mfc1 $s1, $f1 # movemos de $f1 a $s1
8
```

Name	Float	Double
\$f0	9.5658175E-9	3.772234231360013E-67
\$f1	9.565838E-9	
\$f2	0.0	0.0
\$f3	0.0	
\$f4	0.0	0.0
\$f5	0.0	
\$f6	0.0	0.0
\$f7	0.0	

¿Cuál es la razón por la que al finalizar el programa los contenidos de \$s0 y \$s1 son distintos?

→ Debido al redondeo que aparece cuando hacemos la conversión

EJEMPLO4

```
1  # Código ejemplo4 Desbordamiento
2
3  addi $s0, $0, 1
4  sll $s0, $s0, 30 # $s0 = 2^30
5  mtcl $s0, $f0
6  cvt.s.w $f0, $f0 # $f0 = 2^30
7  mul.s $f0, $f0, $f0 # $f0 = 2^60
8  mul.s $f0, $f0, $f0 # $f0 = 2^120
9  mul.s $f0, $f0, $f0 # $f0 = 2^240 -> overflow
```

¿Qué valor representa en el formato IEEE 754 el contenido final de \$f0?

→ Infinity

EJEMPLO4

Observa que no ha ocurrido ninguna excepción al ejecutar el código.
¿Ocurre lo mismo al dividir por cero? ¿Y al hacer 0/0?

Añade instrucciones al código anterior y haz la prueba.

```
1  li $s1, 1
2  mtc1 $s1, $f1
3  cvt.s.w $f1, $f1
4
5  sub.s $f0, $f0, $f0
6
7  div.s $f1, $f1, $f0 #n/0 --> Infinity
8  div.s $f0, $f0, $f0 #0/0 --> NaN
9
```

Name	Float	Double
\$f0	NaN	1.40444842999607E306
\$f1	Infinity	
\$f2	0.0	0.0
\$f3	0.0	
\$f4	0.0	0.0
\$f5	0.0	
\$f6	0.0	0.0
\$f7	0.0	

EJEMPLO5

Completa el código para que muestre en consola un mensaje de error por desbordamiento. Comprueba que funciona correctamente.

```
1  # detectar casos especiales del formato IEEE 754
2  .data
3      mmask: .word 0x007FFFFF
4      emask: .word 0x7F800000
5      exp1: .word 255
6      desborda: .asciiz "Hay desbordamiento"
7      Nodesborda: .asciiz "No hay desbordamiento"
8  .text
9      addi $s0, $0, 1
10     sll $s0, $s0, 30 # $s0 = 2^30
11     mtcl $s0, $f0
12     cvt.s.w $f0, $f0 # $f0 = 2^30
13     mul.s $f0, $f0, $f0 # $f0 = 2^60
14     mul.s $f0, $f0, $f0 # $f0 = 2^120
15     mul.s $f0, $f0, $f0 # $f0 = 2^240 -> overflow
16
17     #Valor a comprobar en $f0
18
19     mfc1 $s0, $f0
20     lw $t4, mmask # cargar mascara de la mantisa #Numero que tengo y la mantisa
21     and $t0, $s0, $t4 # extraer mantisa de $s0
22     lw $t4, emask # cargar mascara del exponente
23     and $t2, $s0, $t4 # extraer exponente de $s0 #Numero que tengo y el exponente
24     srl $t2, $t2, 23 # desplazar exponente
25     lw $t3, exp1 #cargamos valor exponente todo a unos
26     beq $t2, $t3, exp_a_1 #exponente todo a unos?
27
28     la $a0, Nodesborda
29     li $v0, 4
30     syscall
31
32     j fin
33
34     exp_a_1:
35         la $a0, desborda
36         li $v0, 4
37         syscall
38         j fin
39     fin:
40         li $v0, 10
41         syscall
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
```

Line: 1 Column: 1 ☒ Show Line Numbers

Mars Messages Run I/O

Clear Hay desbordamiento
-- program is finished running --

Ejercicios optativos Práctica 9

EJEMPLO1

¿Cuál es la razón por la que el registro base de las instrucciones lwcl y swcl pertenecen al banco de registros de enteros y no de la FPU?

→ La razón es porque las direcciones son números enteros

Haz una traza del programa a mano y obtén el valor de los registros del \$f0 al \$f9. Conviene notar que vector1 y vector2 tienen los mismos elementos, pero con diferente orden. Observa los resultados que hay en \$f4 y \$f9.

The screenshot displays the Mars SPIM simulator interface. The main window shows MIPS assembly code for a program that calculates the sum of two vectors. The code is as follows:

```

1  #Código ejemplo 1
2
3  .data
4
5  vector1: .float 5.6e+20, -5.6e+20, 1.2
6  vector2: .float 1.2, 5.6e+20, -5.6e+20
7
8  .text
9  la $t0, vector1
10 lwcl $f0, 0($t0)
11 lwcl $f1, 4($t0)
12 lwcl $f2, 8($t0)
13 #Sumar los componentes de un vector
14 add.s $f3, $f0, $f1
15 add.s $f4, $f2, $f3 #f4 tiene el valor de la suma de los elementos de vector1
16
17 la $t1, vector2
18 lwcl $f5, 0($t1)
19 lwcl $f6, 4($t1)
20 lwcl $f7, 8($t1)
21 add.s $f8, $f5, $f6
22 add.s $f9, $f7, $f8
23

```

The right-hand pane shows the register and floating-point register (FPU) state. The FPU registers \$f0 through \$f9 are visible, showing the results of the floating-point operations. For example, \$f3 contains the sum of the first two elements of vector1, and \$f4 contains the sum of the first three elements of vector1.

The bottom status bar indicates "Line: 23 Column: 2" and "Show Line Numbers" is checked. The "Mars Messages" pane at the bottom shows the message: "-- program is finished running (dropped off bottom) --".

Ensambla, ejecuta el programa y observa el contenido que adquieren los registros para verificar los resultados que has obtenido a mano. ¿Qué conclusión puedes sacar?

➔ Al sumar un número pequeño y un número grande, los resultados obtenidos en \$f4 y %f9 son distintos por el redondeo

No hay comparación mayor que, ¿cómo lo podéis solucionar?

→ Intercambiando los registros de posición a la hora de ejecutar la orden c.le.s

¿Dónde crees que se ejecutará la instrucción bclt? ¿En la CPU o en la FPU?

→ Al ser enteros se ejecutan en la CPU

EJEMPLO2

```
1  # Determinar el menor de dos números en coma flotante
2  ...
3  c.lt.s $f0,$f2    # es < A B?
4  bclt   print_A    # sí - escribir en consola A
5  c.lt.s $f2,$f0    # es B < A?
6  bclt   print_B    # sí - Escribir consuela B
7  ...
8
```

¿Qué código de condición se ve afectado?

→ El 1

¿Hasta cuándo permanecerá el valor del código de condición?

→ Hasta que se vuelva a modificar