

Laboratorio 1

Introducción a MARS y al MIPS

Arquitectura de Ordenadores

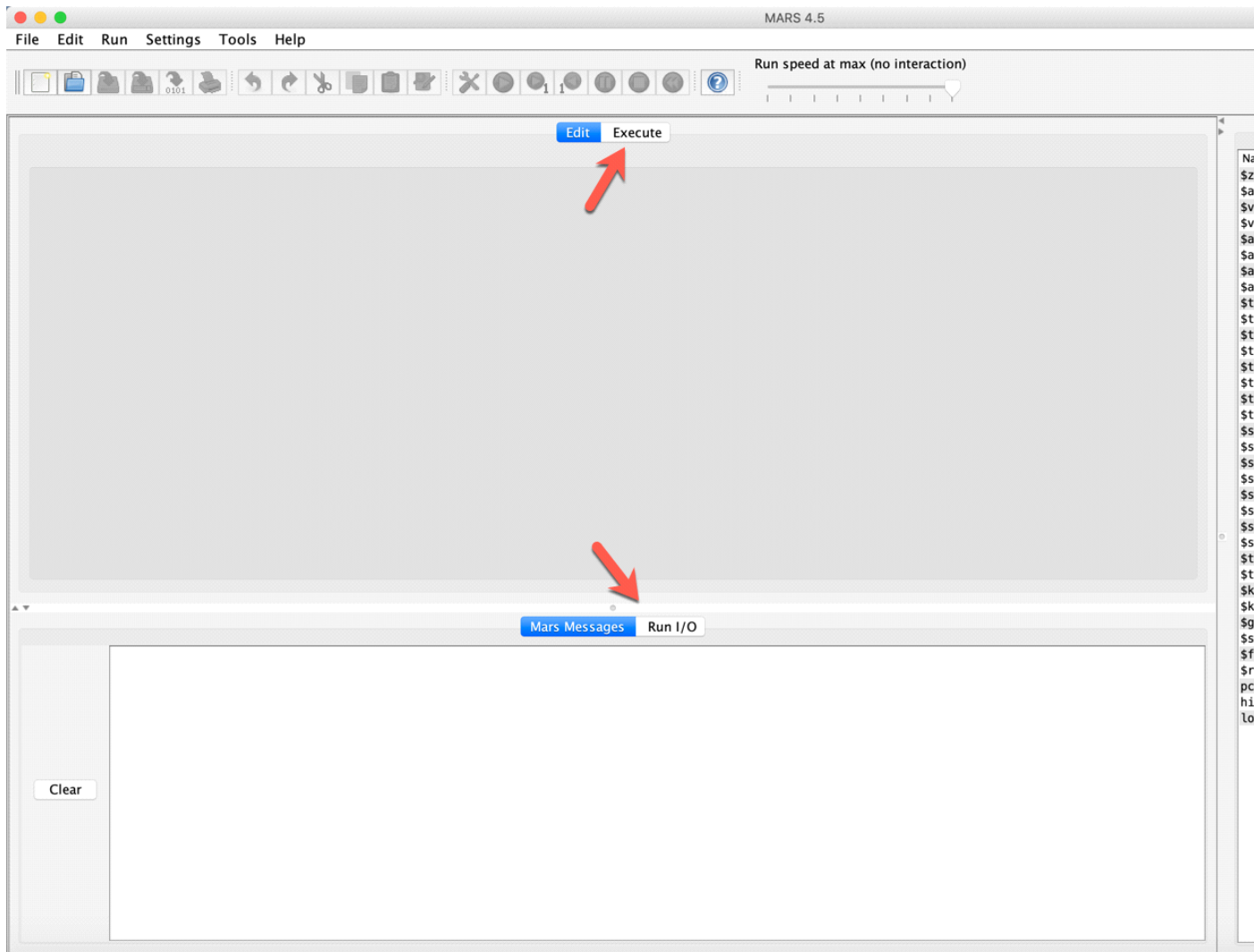
En este laboratorio se introducirá el entorno de programación MARS que permite el desarrollo de programas en lenguaje ensamblador MIPS. Puede descargar MARS utilizando el enlace que se especifica a continuación:

<http://courses.missouristate.edu/KenVollmar/MARS/>

Fundamentos de MARS

Ejercicio 1 (4 puntos):

Arranque la aplicación y observe el entorno de programación.



Note las pestañas **Edit/Execute** y **Mars Messages/Run I/O**. El significado de estas pestañas es evidente. Observe también el panel con todos los registros que se halla a su derecha. Se muestran todos los registros de la arquitectura con sus números y etiquetas.

Al final de la lista aparecen también unos registros de propósito específico (**pc** o contador de programa y **hi** y **lo** cuyo significado se explicará más adelante)

Utilice **File...Open** para abrir el fichero **lab1.asm** que se adjunta con este manual. El fichero contiene un programa que suma un vector de cinco números. Podrá observar el fichero abierto en el panel **Edit**.

NOTA: (todos los iconos tienen un equivalente en la barra de menús; en el resto de este tutorial intentaremos hacer referencia a los iconos siempre que sea posible)



Ensamble el programa utilizando el icono (disponible también en el menú **Run**). Examine el panel **Mars Messages** y compruebe que el mensaje indica que el ensamblado ha sido realizado con éxito.

Observe también que la pestaña cambia automáticamente de **Edit** a **Execute** y que ahora se muestran los paneles **Text Segment** y **Data Segment**.

-*-

1. El Text Segment contiene el código de la sección **.text** del programa (las instrucciones del programa). Explique, con sus propias palabras, qué representan cada una de las columnas de este panel:

Bkpt: Breakpoint. Si se activa, el código para en esa línea para que se pueda observar el proceso paso a paso, con acceso a los valores de los datos usados

Address: Dirección de la instrucción. Dirección que usa el program counter para ejecutar dicha instrucción

Code: Codificación en hexadecimal de la instrucción.

Basic: Instrucción en sí. Se ve en la forma de texto en la que es escrita en lenguaje mips.

Source: En caso de que la instrucción original haya tenido que ser fragmentada en 2, referencia a la línea que existe en el programa (si un **lw** en la línea 14 causa un **lui** y un **ori**, el primero tiene de source la línea 14 y el otro ninguna, por lo que se infiere que la línea 14 ha sido dividida en 2 operaciones conjuntas)

-*-

2. Conteste también a las preguntas siguientes:

¿Cuál es la dirección de inicio del programa?

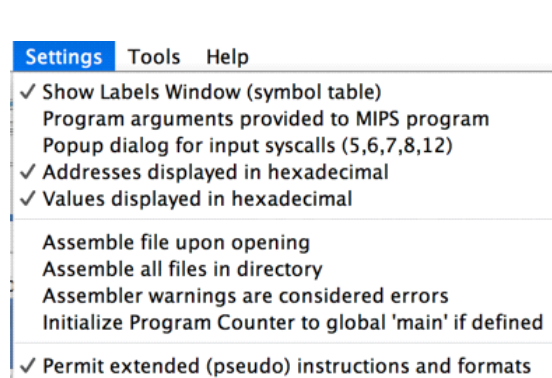
El programa empieza en el label `main`, línea 14 (primera línea utilizada en la sección `.global main`, donde se empieza la ejecución), que tiene como direcciones 4194304 y 4194308 (direcciones usadas por el pc) al ser conjuntas.

El **Data Segment** contiene el código de la sección `.data` del programa (las variables y las constantes definidas en el programa). ¿Cuál es la dirección de inicio del **Data Segment**?

Es el valor 5, el primero en ser cargado. Tiene dirección 268500992 (primera palabra, value +0)

Cada fila del **Data Segment** muestra el contenido de 8 palabras de memoria, cada una de las cuales contiene 32 bits, o 4 bytes, de datos. Observe que las primeras 14 palabras del **Data Segment** contienen valores no nulos. ¿Por qué?

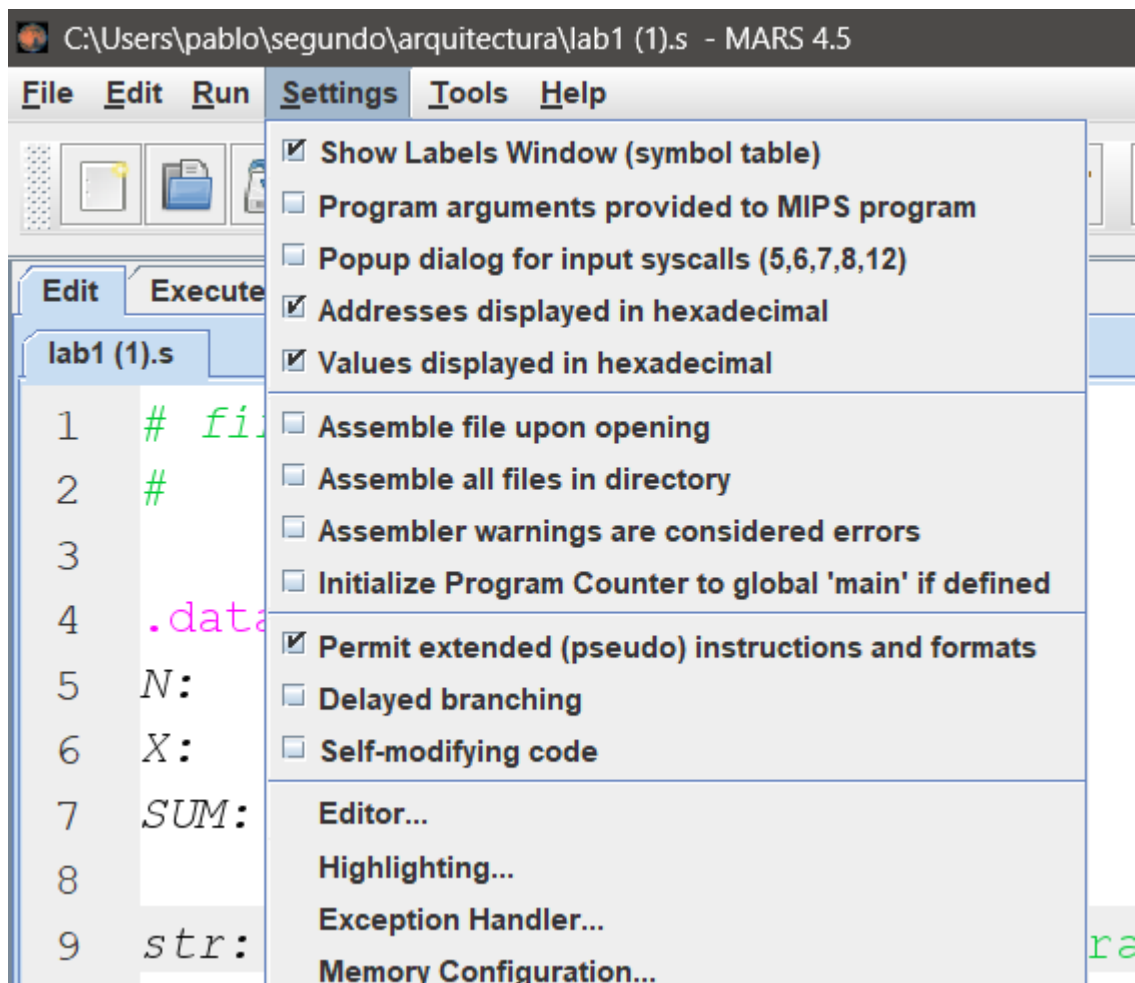
Porque en la sección `.data` primero se carga el valor `n = 5` y luego se añade un vector de 5 valores a sumar y el valor del resultado, sumando 7 espacios. Los otros 7 valores son las palabras del string ("the sum of the array is = ", donde los espacios no cuentan como palabra y "=" sí que cuenta) en código ascii, por lo que el siguiente valor es 0 forzado para poder separar la palabra del resto de valores que sigan en memoria.



*

Utilice el menú **Settings** para configurar los paneles de MARS. La configuración se guardará y se utilizará en la próxima sesión.

- El **panel Labels** contiene las direcciones de las instrucciones del código ensamblador con una etiqueta, este panel no se muestra por defecto por lo que hay que seleccionarlo en el menú **Settings**.
- Seleccione también la opción para permitir pseudo-instrucciones (sustituciones de algunas instrucciones con códigos más amigables para el programador y otros atajos).
- Seleccione la opción para mostrar los valores y las direcciones en formato hexadecimal.



Run speed at max (no interaction)







Utilice el *slider* para cambiar la velocidad de ejecución a 1 instrucción por segundo.



Esto nos permitirá observar las instrucciones ejecutándose en vez de terminar el programa directamente.

Hay varias formas de ejecutar un programa:



- El icono  ejecuta el programa hasta. Utilizando este icono debería observar la barra evidenciadora amarilla mostrando cómo progresa la ejecución del programa en el **Text Segment** y la barra evidenciadora verde mostrando cómo se van modificando los registros en el panel **Registers**. También los cambios en el **Data Segment** son evidenciados.
- El icono  resetea el programa a sus valores iniciales. El contenido de la memoria es el que se especifica en el programa y los registros están, en general, puestos a cero.

- El icono  “single-step”, permite avanzar de una instrucción, mientras que , “single-step backwards” retrocede de una instrucción deshaciendo la operación anterior.

Ejecute el programa hasta que complete su ejecución utilizando una velocidad de ejecución muy baja (una instrucción por Segundo). Al completarse la ejecución se visualizará lo siguiente:

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

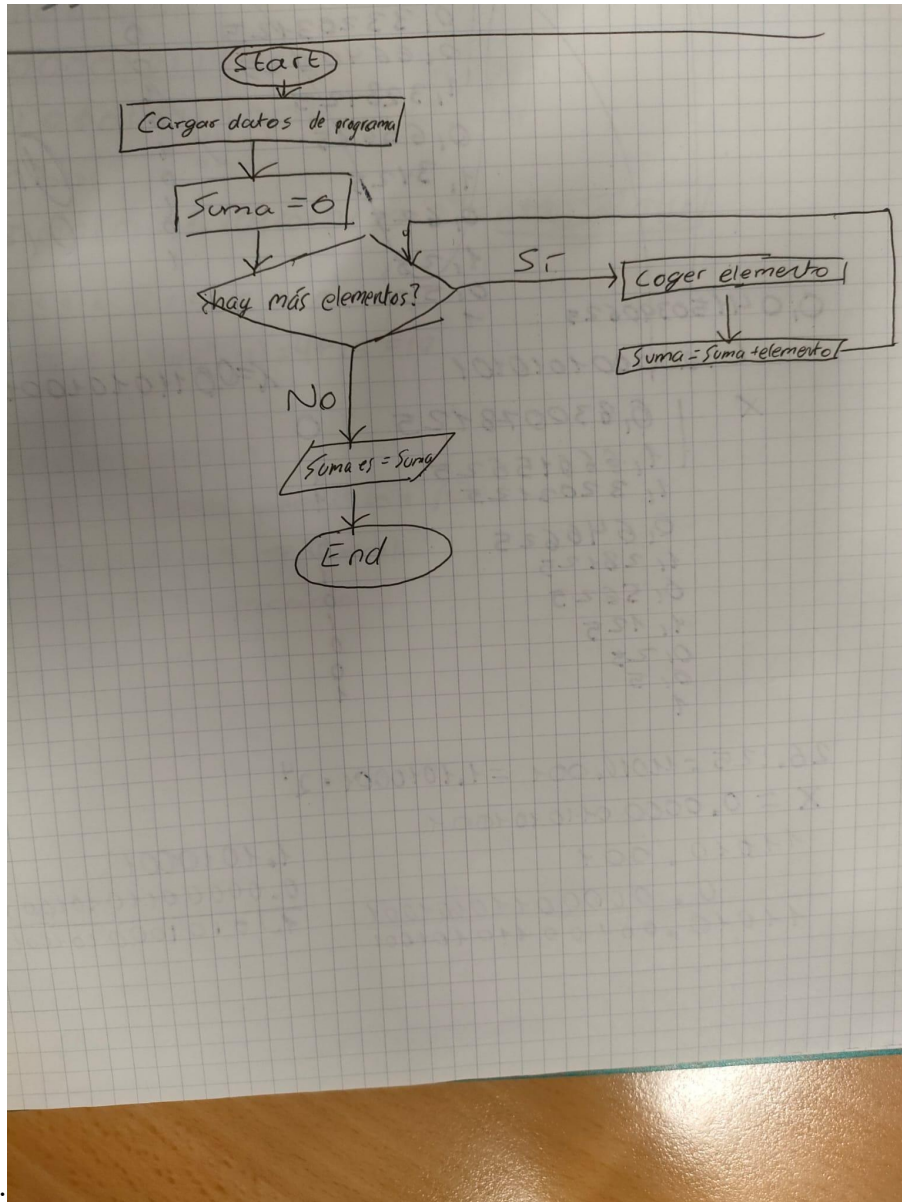
```
the sum of the array is = 30
-- program is finished running --
```

Reinicie  el programa y ejecútelo una instrucción a la vez utilizando la opción *single-step* . Examine el contenido de los registros después de cada instrucción y asegúrese de entender los cambios.

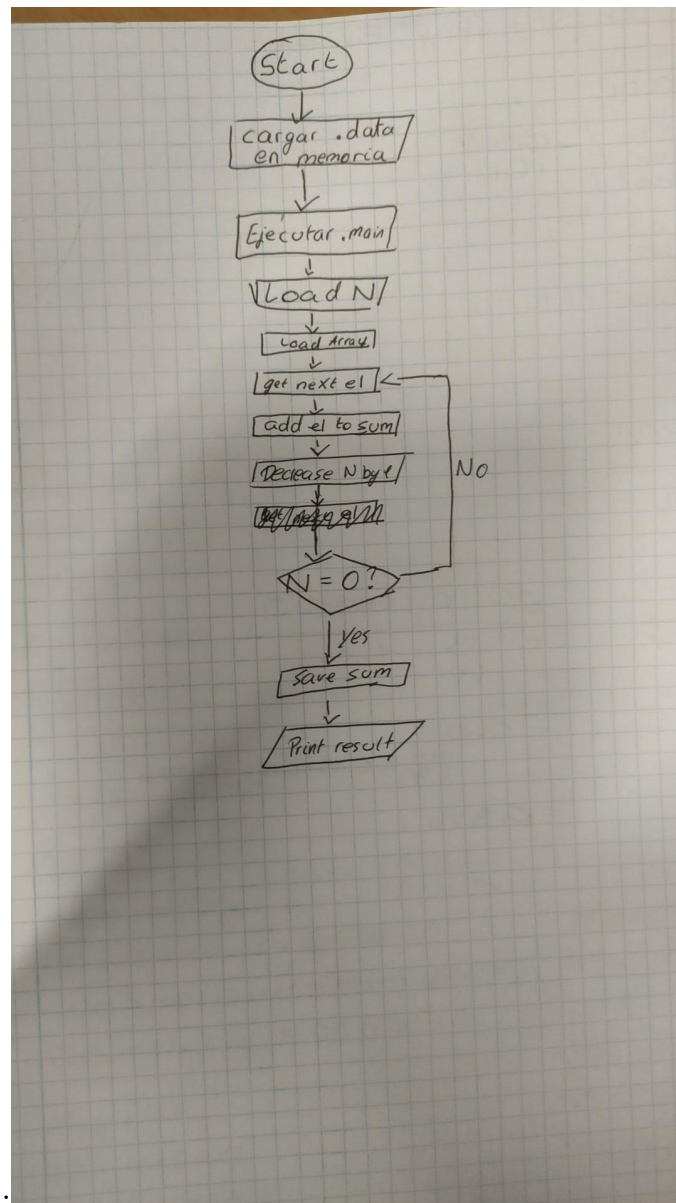
*

3. Dibuje un diagrama de flujo del programa:

Como no estoy seguro si el diagrama debe ser a nivel coloquial o más centrado en el funcionamiento de mips, hice ambos:





Coloquial:




Más cercano a mips:


~*~

4. Fije un breakpoint en la dirección 0x400014 seleccionando el *checkbox* correspondiente al lado de la instrucción.

- Reinicie  y ejecute  el programa otra vez. Esta vez se parará al *breakpoint* antes de ejecutar la instrucción. Examine el valor de **\$t0** a este punto. ¿Cuál es?
- El valor es 0x10010004. Esto corresponde con la dirección de memoria del primer valor del vector: el vector empieza en la posición 0x10010000 Value +4 (0x10010004)

<input type="checkbox"/>	0x00400000	0x3c011001	lui \$1,0x00001001	14: main: lw \$s0, N	# load l..
<input type="checkbox"/>	0x00400004	0x8c300000	lw \$16,0x00000000(\$1)		
<input type="checkbox"/>	0x00400008	0x3c011001	lui \$1,0x00001001	16: la \$t0, X	# load t..
<input type="checkbox"/>	0x0040000c	0x34280004	ori \$8,\$1,0x00000004		
<input type="checkbox"/>	0x00400010	0x02208824	and \$17,\$17,\$0	17: and \$s1, \$s1, \$zero	# clear ..
<input checked="" type="checkbox"/>	0x00400014	0x8d090000	lw \$9,0x00000000(\$8)	18: loop: lw \$t1, 0(\$t0)	# load t..
<input type="checkbox"/>	0x00400018	0x02298820	add \$17,\$17,\$9	19: add \$s1, \$s1, \$t1	# add it..
<input type="checkbox"/>	0x0040001c	0x21080004	addi \$8,\$8,0x00000004	20: addi \$t0, \$t0, 4	# increm..
<input type="checkbox"/>	0x00400020	0x2210ffff	addi \$16,\$16,0xffff..	21: addi \$s0, \$s0, -1	# decrem..
<input type="checkbox"/>	0x00400024	0x1410ffff	bne \$0,\$16,0xfffffff	22: bne \$0, \$s0, loop	# loop b..

- Avance de una instrucción  para ejecutar la **add**. Examine de **\$t0** otra vez.
¿Cuánto vale ahora?
- Vale lo mismo porque no ha sido actualizado \$t0, sino \$s1 (se observa en la foto anterior)

- Haga click en  para continuar la ejecución desde el breakpoint.

*

5. Ahora cambie el programa para cumplir con estas condiciones e indique qué instrucciones se han modificado:

Se añade un número más al array X (por ejemplo, el 14):

Para hacer eso, actualizo el valor de N ya que representa el tamaño de X, por lo que ahora valdrá 6. luego, en la línea en la que se declara x añado el valor 14 por detrás de el último valor seguido de una coma.

```

C:\Users\pablo\segundo\arquitectura\lab1 (1).s* - MARS 4.5
File Edit Run Settings Tools Help
Run speed 1 inst/sec
lab1 (1).s*
1 # first MIPS program
2 #
3
4 .data                                # Put Global Data here
5 N:      .word 6                      # loop count
6 X:      .word 2, 4, 6, 8, 10, 14    # array of numbers to be added
7 SUM:    .word 0                     # location of the final sum

```

El bucle avanza de dos en dos:

Si va saltando las posiciones pares (del primero va al tercero) puede haber 2 casos al comprobar si el array ha sido completado cuando es cierto: el valor comparado puede ser 0 o -1 ya que, si se empieza con 1 elemento, restarle 2 sobrepasa 0. Se puede sustituir el bne por una comparación slt1 para ver si es menor que 1 (ya que operamos con enteros no hay que preocuparse por fracciones, por lo que indicaría si es igual o menor que 0) el valor será 1 si es cierto por lo que se haría un branch if not equal (bne) comparando ese valor con \$zero.

Además, la operación para conseguir la siguiente palabra tiene que ser cambiada para sumar 8 en vez de 4 (2 palabras en vez de 1) y la operación de decremento deberá restar de 2 en 2.


```

main:    lw $s0, N                # load loop counter into $s0

        la $t0, X                # load the address of X into $t0
        and $s1, $s1, $zero      # clear $s1
loop:    lw $t1, 0($t0)           # load the next value of array X
        add $s1, $s1, $t1        # add it to the running sum
        addi $t0, $t0, 8         # increment to the next address
        addi $s0, $s0, -2        # decrement the loop counter
        slti $s2, $s0, 1        # check if value <= 0
        bne $0, $s2, loop        # loop back until complete
        sw $s1, SUM              # store the final total

```

*

NOTA: Observe que puede modificar directamente los valores de los registros y de la memoria directamente al *breakpoint* antes de continuar se así fuera necesario para realizar alguna prueba específica sobre el programa



Abra el Help para obtener informaciones sobre las instrucciones del MIPS, pseudoinstrucciones, directivas y *syscalls*.

Ejercicio 2 (4 puntos):

Escriba el programa que se muestra a continuación (no es necesario escribir también los comentarios), y guárdelo en el escritorio de su ordenador como **lab1-2.s**:

```

.text
.globl main

main:
    li $v0, 4          #syscall to print a string
    la $a0, prompt     #address of string to print
    syscall            #print the string

    li $v0, 1          #syscall to print an integer
    lb $a0, val        #load the integer to print to $a0
    syscall            #print it

    addi $t0, $t0, 1    #increment the value of $t0

    li $v0, 10         #sys call for exit
    syscall

.data
    prompt: .asciiz "tu codigo es: "
    val:    .byte 20

```

Ejecute el programa y observe su salida (en caso de error en la línea `prompt: .asciiz "tu codigo es"`, sustituir las comillas “ ”).

```

tu codigo es: 20
-- program is finished running --

```

-*-

1. Examine el **Segmento de Datos**. Transcriba los valores de los primeros 16 bytes de datos en memoria (escribalos exactamente como se muestran en MARS, donde 4 bytes son agrupados para formar una entrada):

En address 0x10010000:

Value (+0): 0x63207574

Value (+4): 0x6769646f

Value (+8): 0x7365206f

Value (+c): 0x1400203a

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+C)
0x10010000	0x63207574	0x6769646f	0x7365206f	0x1400203a

-*-

Asimismo, conteste las siguientes preguntas:

2. Transcriba direcciones y datos de cada byte del Segmento de Datos utilizando un modelo de memoria tipo pila.

- Muestre un byte por fila.
- La dirección más baja debe hallarse en la **base** de la pila.
- Etiquete la dirección a la que corresponde el comienzo de la cadena de caracteres de la **pregunta**
- Etiquete la dirección que corresponde a la variable **val**.

Value (+0): 0x63207574

Value (+4): 0x6769646f

Value (+8): 0x7365206f

Value (+c): 0x1400203a

<u>Address</u>	<u>Label</u>	<u>Data</u>
0x10010015	val	14
0x10010014		00
0x10010013	prompt	20
0x10010012	prompt	3a
0x10010011	prompt	73
0x10010010	prompt	65
0x10010009	prompt	20
0x10010008	prompt	6f
0x10010007	prompt	67
0x10010006	prompt	69
0x10010005	prompt	64
0x10010004	prompt	6f
0x10010003	prompt	63
0x10010002	prompt	20
0x10010001	prompt	75
0x10010000	prompt	75

3. ¿Cuál es el significado del primer byte que tiene el valor 00?

Marca el final del string, por lo que el siguiente valor (hex 14, dec 20) es un valor numérico

¿De qué viene el valor 14?

Es el valor entero que es impreso, no cuenta como parte del String

*

Ejercicio 3 (2 puntos):

Cree otro fichero que contiene el programa siguiente y guárdelo como **lab1-3.s**:

.text

```

.globl main

main:
    li $v0,10
    syscall

.data
nums:    .byte  8,7,6,5
         .half  4,3,2,1
         .word  1,2,3,4
         .space 1
         .word  0x12
letters: .ascii  "ABC"
         .ascii  "abc"
negls:   .byte  -1,-1
         .word  15

```

*

Transcriba direcciones y datos de cada byte del Segmento de Datos utilizando un modelo de memoria tipo pila (en caso de ser necesario, añadir las filas que faltan).

- Muestre un Word (4 bytes) por fila.
- La dirección más baja debe hallarse en la **base** de la pila.
- Etiquete las direcciones correspondientes a las variables **nums** y **letters**.
- Marque el byte con la dirección de **negls** sombreando con color la celda en cuestión.

<u>Address</u>	<u>Label</u>	<u>Data</u>
0x1010034		15
<u>0x101002c</u>		<u>-1,-1</u>
0x1010028		a,b,c
0x1010024	(letters)	A,B,C,
0x1010020		18
0x101001c		
0x1010018		4
0x1010014		3
0x1010010		2
0x101000c		1
0x1010008		2,1
0x1010004		4,2
0x1010000	(nums)	8,7,6,5

(tambien he subrayado negls)

*

Ejercicio 4 (1 punto extra):

Considere el programa siguiente (no necesita crear un Nuevo fichero, lea simplemente el código siguiente y conteste las preguntas):

```
.text
    la $s0,neg1byte
    la $s1,oneword
    la $s2,twowords
    la $s3,smallstring
    lb $t0,3($s1)
    add $s1,$s1,$t0
    lb $t1,0($s1)
    addi $s1,$s1,2
    add $s2,$s1,$t1
    sb $t0,0($s2)
    addi $s1,$s1,7
    add $a0,$s1,$t0
    li $v0,4
    syscall
    lw $t0,0($s3)
    sh $t0,-5($s2)
    li $v0,10
    syscall

.data
    neg1byte:    .byte -1
    oneword:     .word 0x02030405
    twowords:     .word 02,03
    smallstring: .asciiz "abc"
    halfwords:   .half 10,11,12,13,14,15
```

Asuma que la variable **neg1byte** represente la dirección 0x10010000, y que todos los bytes de memoria no asignados a variables contienen 00.

*

1. ¿Cuántos bytes del segmento de datos son reservados para el segmento de datos desde la dirección de comienzo **neg1byte** hasta el último byte de dato almacenado en la variable **halfwords** (considere también eventuales bytes extra reservados por la directriz de datos)?

1 para neg1byte

1 para oneword y 2 para twowords al tener 2 valores de 1 palabra cada

Smallstring son 3 letras de 2 dígitos hexadecimales más el 00, por lo que cabe en 1

Hay 6 valores que valen media palabra en halfwords por lo que se crean 3

En total se crean 8

Escriba los valores de los bytes almacenados en el Segmento de Datos antes que el programa se ejecute:

Data Segment

Address (+18)	Value (+0) Value (+1C)	Value (+4)	Value (+8)	Value (+C)	Value (+10)	Value (+14)	Value
0x10010000	0x0000ff	0x02030405	0x000002	0x000003	0x00636261	0x000b000a	0x000d000c
	0x000f000e						

2. Rellene la tabla siguiente para mostrar los contenidos de los registros $\$s0-\$s3$ y $\$t0-\$t1$ después de que cada instrucción es ejecutada.

Para cada instrucción, rellene **sólo** aquellas entradas que **cambian**. Si un valor almacenado en memoria es modificado por una instrucción, transcriba también aquella dirección y su nuevo valor. Asuma que todos los registros estén inicializado al valor 0.

	$\$s0$	$\$s1$	$\$s2$	$\$s3$	$\$t0$	$\$t1$	$\$a0$	addr:
val:								
la $\$s0, \text{neglbyte}$	0x1001000							
la $\$s1, \text{oneword}$		0x1001004						
la $\$s2, \text{twowords}$			0x1001008					
la $\$s3, \text{smallstring}$				0x10010010				
lb $\$t0, 3(\$s1)$					0x00000002			
add $\$s1, \$s1, \$t0$		0x1001006						
lb $\$t1, 0(\$s1)$						0x00000003		
addi $\$s1, \$s1, 2$		0x1001008						
add $\$s2, \$s1, \$t1$			0x1001000b					
sb $\$t0, 0(\$s2)$						0x00000002		
addi $\$s1, \$s1, 7$		0x1001000f						
add $\$a0, \$s1, \$t0$							0x10010011	
li $\$v0, 4$								
syscall								
lw $\$t0, 0(\$s3)$					0x00636261			
sh $\$t0, -5(\$s2)$								

Transcriba los valores de los bytes almacenados en la sección de datos **después** que el programa se ha ejecutado (transcriba **sólo** los nuevos valores en memoria, en la posición correcta; no hace falta rescribir la tabla entera):

Data Segment

<u>Address</u>	<u>Value (+0)</u>	<u>Value (+4)</u>	<u>Value (+8)</u>	<u>Value (+C)</u>	<u>Value (+10)</u>	<u>Value (+14)</u>	<u>Value (+18)</u>
<u>Value (+1C)</u>							
<u>0x10010000</u>	0x0000ff	0x62610405	0x020002	0x000003	0x00636261	0x000b000a	0x000d000c
0x000f000e							