

## Laboratorio 1 Introducción a MARS y al MIPS

### *Arquitectura de Ordenadores*

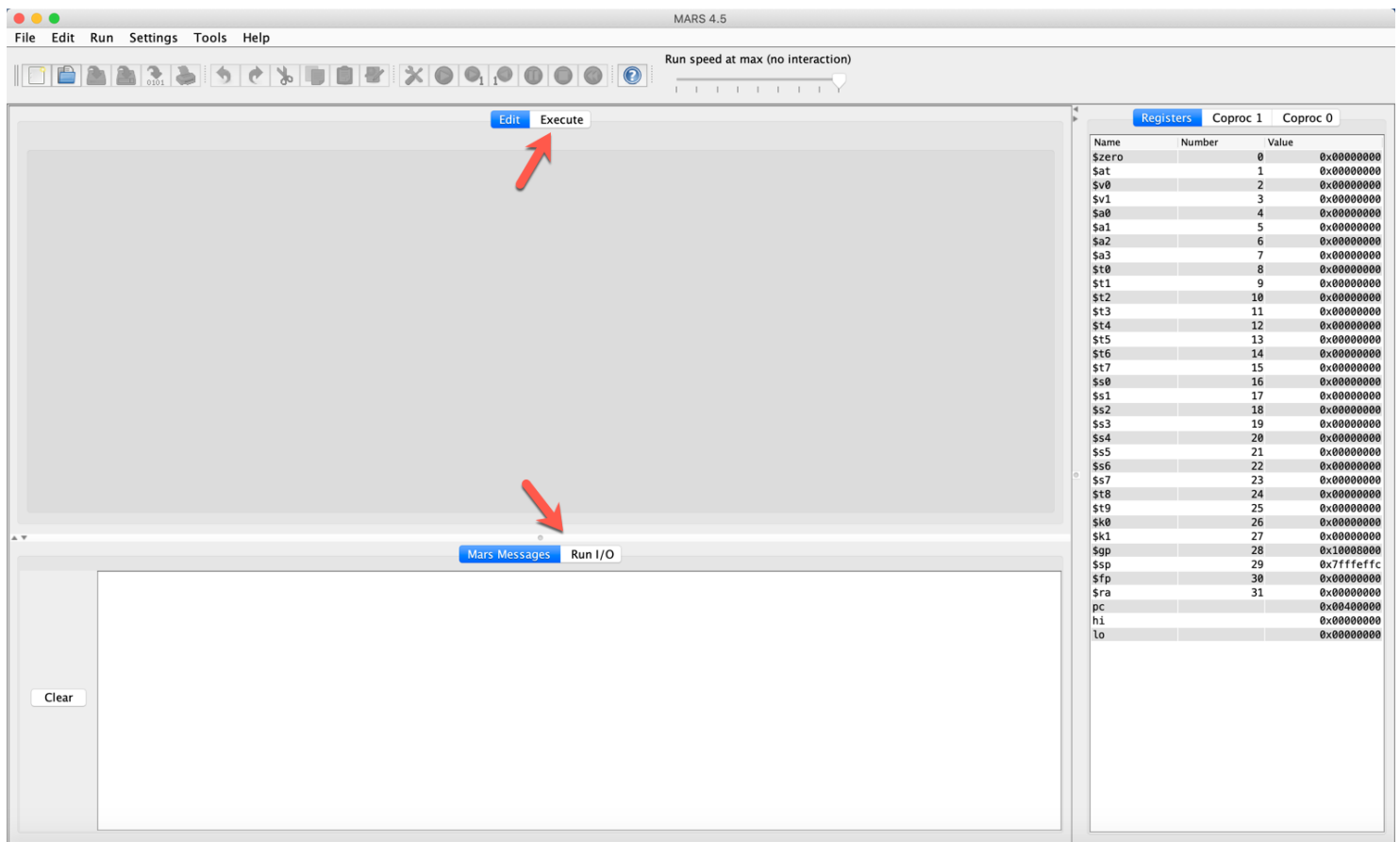
En este laboratorio se introducirá el entorno de programación MARS que permite el desarrollo de programas en lenguaje ensamblador MIPS. Puede descargar MARS utilizando el enlace que se especifica a continuación:

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

## Fundamentos de MARS

### Ejercicio 1:

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.
------	---------	---	------	----------

El **Text Segment** contiene el código de la sección **.text** del programa (las instrucciones del programa). Explique, en su opinión, qué representan cada una de las columnas de este panel:

*Bkpt*: Si lo seleccionas se interrumpe el programa justo en esa línea

*Address*: Dirección de memoria donde se encuentra la instrucción

*Code*: Código de instrucción que entiende la máquina

*Basic*: Código Ensamblador

*Source*: Instrucción de código en lenguaje ensamblador

Conteste también a las preguntas siguientes:

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

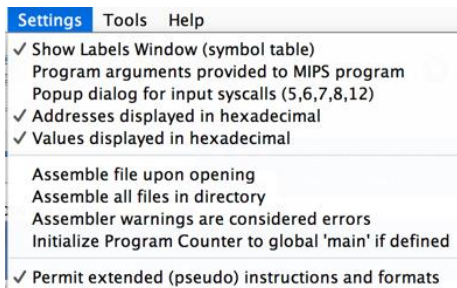
¿Cuál es la dirección de inicio del programa?

0x00400000

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**?

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?

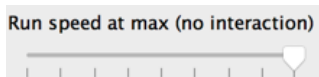
Porque son valores que se van a ir reescribiendo a medida que avance el programa



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.



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 --
```





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.


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? 268500996

Avance de una instrucción  para ejecutar la **add**. Examine de **\$t0** otra vez. ¿Cuánto vale ahora? 268500996 pero **\$t0** vale 2 ahora (antes 0)

Haga click en  para continuar la ejecución desde el breakpoint.

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:

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 "your code is: "
val:    .byte 8
```

Ejecute el programa y observe su salida (en caso de error en la línea `prompt: .asciiz "your code is"`, sustituir los quotes `" "`, con `" "`).

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):

**Data Segment**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+C)
0x1001000	0x72756f79	0x646f6320	0x73692065	0x0800203a

Asimismo, conteste las siguientes preguntas:

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**.

Address      Label    Data

0x100100f	val	/0
0x100100e		
0x100100c	:	
0x100100b	s	
0x100100a	i	
0x1001009		
0x1001008	e	
0x1001007	d	
0x1001006	o	
0x1001005	c	
0x1001004		
0x1001003	r	
0x1001002	u	
0x1001001	o	
0x1001000	Prompt	y

0x10010000

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

¿De qué viene el valor 08? \b

Ejercicio 3:

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  4,3,2,1
         .half  8,7,6,5
         .word  1,2,3,4
         .space 1
         .word  12
letters: .ascii "EFG"
         .ascii "efg"
negls:   .byte  -1,-1
         .word  15
```

**Antes** de ejecutar el programa, **prediga** el contenido de la memoria de datos (utilice valores hexadecimales, para ahorrar espacio omita el **0x** antes del número):

**Data Segment**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+C)	Value (+10)	Value (+14)	Value (+18)	Value (+1C)
<u>0x10010000</u>	0x01020304	0x00070008	0x00050006	0x00000001	0x00000002	0x00000003	0x00000004	0x00000000
<u>0x10010020</u>	0x0000000c	0x00474645	0xffff6665	0x0000000f	0x00000000	0x00000000	0x00000000	0x00000000

Asimismo, conteste las siguientes preguntas:





Ejercicio 4:

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.

¿Cuántos bytes del segmento de datos son reservados para el segment 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)?

10 bytes

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

**Data Segment**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+C)	Value (+10)	Value (+14)	Value (+18)	Value (+1C)
0x10010000	0x000000ff	0x02030405	0x00000002	0x00000003	0x00636261	0x000b000a	0x000d000c	0x000f000e

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,neg1byte								
la \$s1,oneword								
la \$s2,twowords								
la \$s3,smallstring								
lb \$t0,3(\$s1)			00000002					10010007:02
add \$s1,\$s1,\$t0								
lb \$t1,0(\$s1)				000000003				10010006:03
addi \$s1,\$s1,2								
add \$s2,\$s1,\$t1								
sb \$t0,0(\$s2)								
addi \$s1,\$s1,7								1001000b:02
add \$a0,\$s1,\$t0								
li \$v0,4								
syscall								
lw \$t0,0(\$s3)								
sh \$t0,-5(\$s2)								

Transcriba los valores **predichos** 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 describir la tabla entera):

## Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+C)	Value (+10)	Value (+14)	Value (+18)	Value (+1C)
0x10010000		0x62610405	0x20000002					