

**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**GRADO EN INTELIGENCIA ARTIFICIAL**

**FUNDAMENTOS DE ARQUITECTURA DE COMPUTADORES**

**Segundo proyecto: Lenguaje ensamblador**

**Grupo 3 - Curso 2023/2024**

**García Pérez Pablo**

**Bravo Vivas Víctor**

**Humanes Cuadrado Jesús**

**Ramírez Ocaña Eneas**

**Higuera Herrero Jorge**

**ÍNDICE**

**Ejercicio 1.**

**1.1** Código empleado con explicación.

**1.2** Direcciones y tamaño en memoria.

**1.2.1** dirección de inicio y fin segmento text, tamaño del programa.

**1.2.2** direcciones y tamaños variables numéricas.

**1.2.3** direcciones y tamaños variables de caracteres.

**1.3** Introducciones de los datos de entrada y salidas de los resultados.

**1.4** Casos de prueba.

**Ejercicio 2.**

**2.1** Código empleado con explicación.

**2.2** Direcciones y tamaño en memoria.

**2.2.1** dirección de inicio y fin segmento text, tamaño del programa.

**2.2.2** direcciones y tamaños variables numéricas.

**2.2.3** direcciones y tamaños variables de caracteres.

**2.3** Introducciones de los datos de entrada y salidas de los resultados.

**2.4** Casos de prueba.

**Ejercicio 3.**

**3.1** Código empleado con explicación.

**3.2** Direcciones y tamaño en memoria.

**3.2.1** dirección de inicio y fin segmento text, tamaño del programa.

**3.2.2** direcciones y tamaños variables numéricas.

**3.2.3** direcciones y tamaños variables de caracteres.

**3.3** Introducciones de los datos de entrada y salidas de los resultados.

**3.4** Casos de prueba.

**Ejercicio 4.**

**4.1** Código empleado con explicación.

**4.2.1** dirección de inicio y fin segmento text, tamaño del programa.

**4.2.2** direcciones y tamaños variables numéricas.

**4.2.3** direcciones y tamaños variables de caracteres.

**4.3** Introducciones de los datos de entrada y salidas de los resultados.

**4.3** Explicación del uso del bucle (loop - while)

**4.5** Casos de prueba.

**Ejercicio 5.**

**5.1** Código empleado con explicación.

**5.2** Direcciones y tamaño en memoria.

**5.2.1** dirección de inicio y fin segmento text, tamaño del programa.

**5.2.2** direcciones y tamaños variables numéricas.

**5.2.3** direcciones y tamaños variables de caracteres.

**5.3** Introducciones de los datos de entrada y salidas de los resultados.

**5.4** Casos de prueba.

**Ejercicio 1: Expresión con enteros**

Se muestra a continuación el código fuente que hemos creado para resolver la ecuación planteada:

.data

x: .space 4 # espacio para la variable x

y: .space 4 # espacio para la variable y

z: .space 4 # espacio para la variable z

w: .space 4 # espacio para la variable w

frase\_x: .asciiz "Dame el valor de x: "

frase\_y: .asciiz "Dame el valor de y: "

frase\_z: .asciiz "Dame el valor de z: "

salida\_W: .asciiz "La solución es: "

.text

# Entrada de valores para x

la $a0, frase\_x

li $v0, 4

syscall

li $v0, 5

syscall

sw $v0, x

# Entrada de valores para y

la $a0, frase\_y

li $v0, 4

syscall

li $v0, 5

syscall

sw $v0, y

# Entrada de valores para z

la $a0, frase\_z

li $v0, 4

syscall

li $v0, 5

syscall

sw $v0, z

main:

lw $s0,x # $s0 x

lw $s1,y # $s1 y

mul $t0,$s1,$s1 # $t0 y\*y

lw $s2,z # $s2 z

mul $t1,$s2,3 # $t1 z\*3

add $t0,$t0,$t1 # $t0 y\*y+z\*3

add $t0,$s0,$t0 # x + y\*y + z\*3

sw $t0,w # w $s3

# Mostramos el valor final de w

la $a0, salida\_W

li $v0, 4

syscall

lw $a0, w

li $v0, 1

syscall

# Terminamos el programa

li $v0, 17 # código de servicio 17 para terminar la ejecución

syscall

Posteriormente, veamos ciertos datos de interés como la dirección de comienzo del programa y el número de bytes que ocupa, las direcciones de las variables usadas y sus respectivos bytes.

**Dirección de comienzo del programa dentro del segmento de texto**

Para la dirección del segmento de texto, empezamos desde 0x00400000 y duramos hasta 0x004000b8. Si restamos el tamaño final al tamaño inicial tenemos que el segmento de texto ocupa unos **1472 bits** que son **184 bytes.**

**Dirección de las variables usadas y sus respectivos bytes**

***Variables numéricas***

*Variable x → 0x10010000 y ocupa 4 bytes*

*Variable y → 0x10010004 y ocupa 4 bytes*

*Variable z → 0x10010008 y ocupa 4 bytes*

*Variable w → 0x1001000c y ocupa 4 bytes*

***Variables de caracteres***

*frase\_x → 0x10010010 y ocupa 19 bytes (18 bytes + 1 byte añadido por el carácter nulo)*

*frase\_y → 0x10010025 y ocupa 19 bytes (18 bytes + 1 byte añadido por el carácter nulo)*

*frase\_z → 0x1001003a y ocupa 19 bytes (18 bytes + 1 byte añadido por el carácter nulo)*

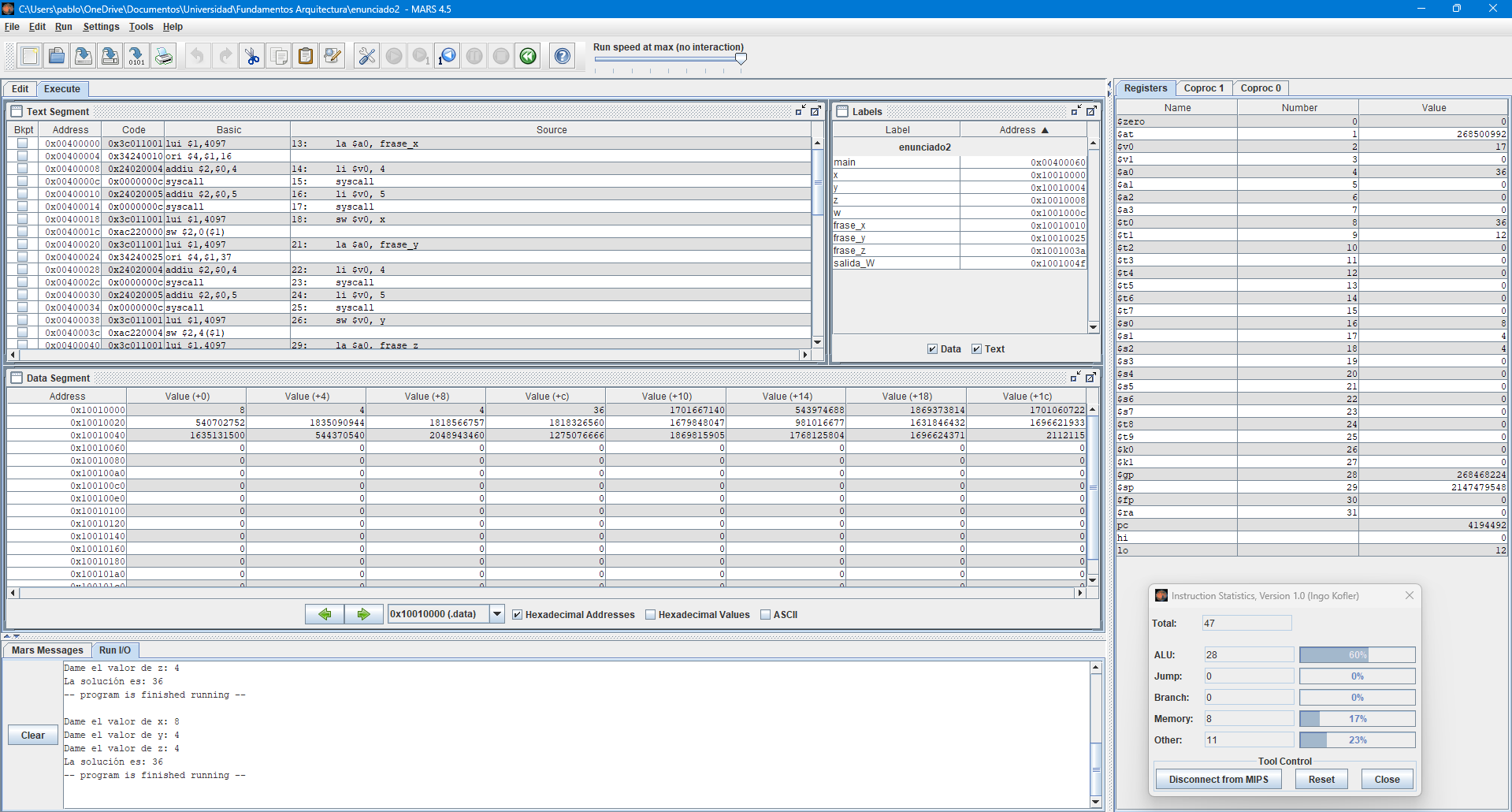
*salida\_W → 0x1001004f y ocupa 18 bytes (17 bytes + 1 byte de carácter nulo )*

**¿Cómo introducimos los datos de entrada y cómo pueden verse los resultados?**

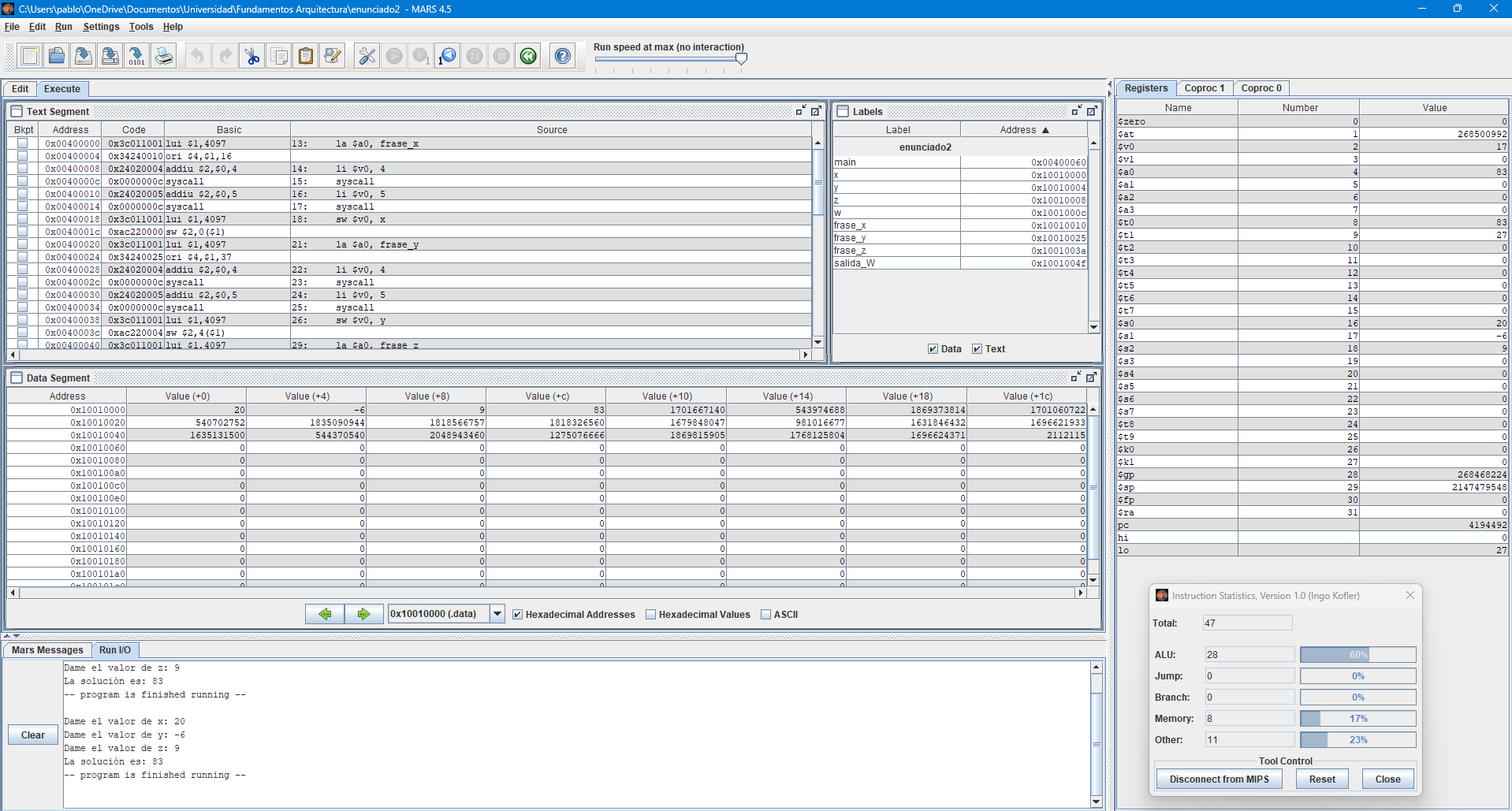
Para este primer ejercicio, hemos decidido en nuestra parte principal del programa la reserva de espacio para nuestras variables y asimismo, reservar espacio y asignar las frases que anteceden a la recolecta de datos que tendremos a posteriori. En nuestra sección **.text** solicitamos los valores de cada variable primero cargando el prompt anteriormente programado y luego cargando el valor que introduzca el usuario en un registro de 4 bytes de espacio.

**Casos de prueba**

*Prueba 1 con x = 8, y = 4, z = 4, total de instrucciones = 47*

**

*Prueba 2 con x = 20, y = -6, z = 9, total de instrucciones = 47*

**

**Ejercicio 2:**

**Código**

.data

#Reservamos espacio para las variables n y suma.

n: .space 4 # Copia en $s0

suma: .space 4 # Copia en $s1

# i: $t0 // ai: $t1 // ai\_1: $t2

frase\_x: .asciiz "Dame el valor de n: "

salida\_w: .asciiz "La solucion es: "

# Solicitar al usuario que introduzca el valor de n

.text

# Entrada de valores para n

la $a0, frase\_x

li $v0, 4

syscall

li $v0, 5

syscall

sw $v0, n

main:

#Le damos valores a nuestras variables

# Cargamos el valor ‘1’ en la variable ‘ai\_1’

li $t2,1

# Cargamos el valor ‘1’ en la variable ‘suma’

li $s1,1

# Cargamos el valor ‘2’ en la variable ‘i’

li $t0,3

# Inicializamos el bucle while

while: lw $s0,n # Cargar índice tope en $s0

bgt $t0,$s0,end\_while

# Multiplicamos la variable ‘ai\_1’ por ‘2’ y lo guardamos en ‘ai’

mul $t1,$t2,2

# if (i % 2 != 0)

if: rem $t3,$t0,2

beq $t3,$zero,end\_if

# Le sumamos a la variable ‘ai’ -1 y lo guardamos en ‘ai’

addi $t1,$t1,-1

end\_if:

# Le sumamos a la variable ‘suma’ la variable ‘ai’ y lo guardamos en ‘suma’

add $s1,$s1,$t1

# Movemos el contenido de ‘ai’ a ‘ai\_1’

move $t2,$t1

# Le sumamos a la variable ‘i’ 1i y lo guardamos en ‘i’

inc: addiu $t0,$t0,1

# Terminamos el bucle while

b while

end\_while:

sw $s1,suma

# Mostrar el resultado por pantalla

# Mostramos el valor final de suma

la $a0, salida\_w

li $v0, 4

syscall

lw $a0, suma

li $v0, 1

syscall

#

li $v0,17

li $a0,0

syscall

# Finalizamos el programa

end\_main:

Una vez inicializado nuestro programa, vayamos a ver ciertos datos de interés como la dirección de comienzo del programa y el número de bytes que ocupa, las direcciones de las variables usadas y sus respectivos bytes.

**Dirección de comienzo del programa dentro del segmento de texto**

Para la dirección del segmento de texto, empezamos desde 0x00400000 y duramos hasta 0x00400098. Tenemos que el segmento de texto ocupa unos **1184 bits** que son **148 bytes.**

**Dirección de las variables usadas y sus respectivos bytes**

***Variables numéricas***

*Variable n → 0x10010000 y ocupa 4 bytes*

*Variable suma → 0x10010004 y ocupa 4 bytes*

***Variables de caracteres***

*frase\_x → 0x10010008 y ocupa 19 bytes (18 bytes + 1 byte añadido por el carácter nulo)*

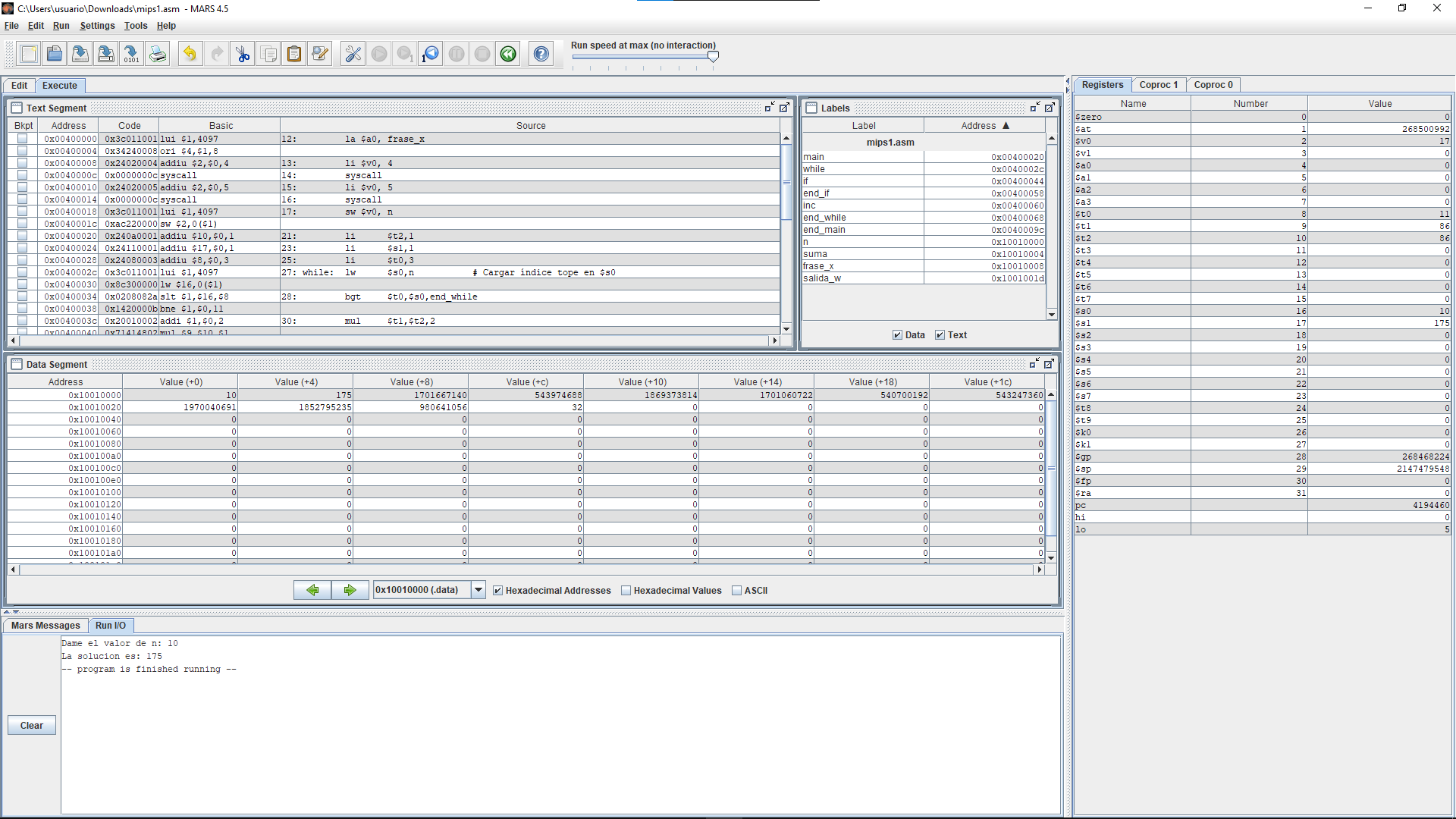
*salida\_W → 0x1001001d y ocupa 18 bytes (17 bytes + 1 byte de carácter nulo )*

**¿Cómo introducimos los datos de entrada y cómo pueden verse los resultados?**

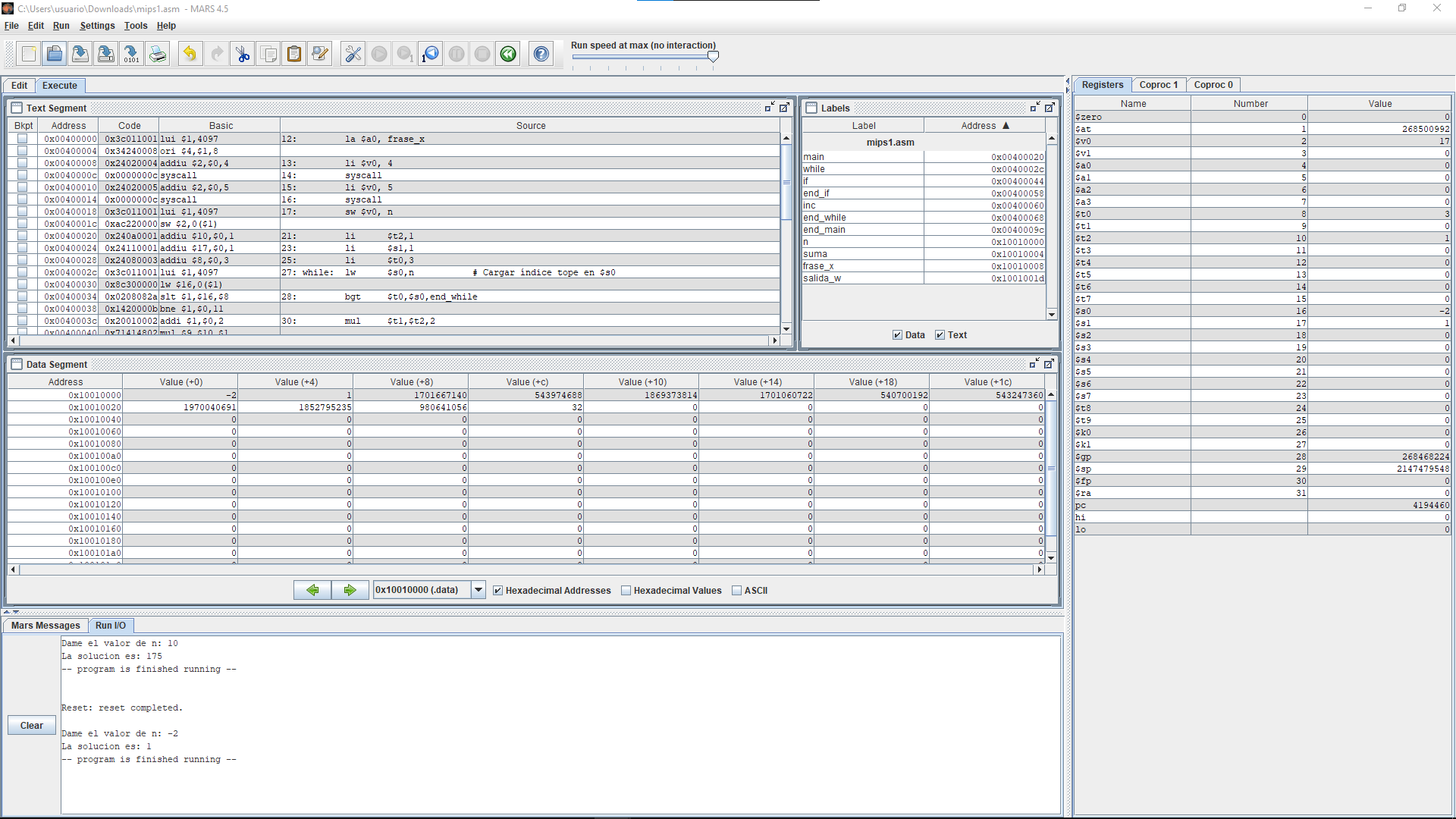
Hemos decidido en nuestro programa la reserva de espacio para nuestras variables y para reservar espacio y asignar las frases que anteceden a la recolecta de datos que tendremos a continuación. En nuestra sección **.text** solicitamos los valores de cada variable primero cargando el prompt anteriormente programado y luego cargando el valor que introduzca el usuario en un registro de 4 bytes de espacio.

**Casos de prueba**

Prueba 1 con n = 10



Prueba 2 con n = -2



## Ejercicio 3

En este ejercicio recorreremos un vector y contaremos la cantidad de ceros que hay en ese vector

### Código

.data

vector: .word 0,1,2,3

.word 0,5,6,7

.word 8,9,10,0

.word 12,13,14,15

cuenta: .word 0

i: .word 0

mensaje: .asciiz "El valor del contador es: "

.text

main:

la $t0, vector # $t0 = dirección de inicio del vector

la $t1, cuenta # $t1 = dirección de 'cuenta'

li $t2, 0 # $t2 = 0 (contador)

li $t3, 0 # $t3 = 0 (índice)

bucle:

lw $t4, 0($t0) # $t4 = vector[i]

beq $t4, $zero, cero # Si $t4 == 0, salta a 'cero'

j fin\_cero # Si $t4 != 0, salta a 'fin\_cero'

cero:

addi $t2, $t2, 1 # Incrementa el contador

fin\_cero:

addi $t0, $t0, 4 # Avanza al siguiente elemento del vector

addi $t3, $t3, 1 # Incrementa el índice

blt $t3, 16, bucle # Si aún no hemos procesado 16 elementos, repite el bucle

sw $t2, 0($t1) # Guarda el resultado final en 'cuenta'

# Imprime el mensaje

la $a0, mensaje # Carga la dirección del mensaje en $a0

li $v0, 4 # Código de servicio para imprimir una cadena

syscall # Llama al sistema para imprimir

# Imprime el valor del contador

lw $a0, 0($t1) # Carga el valor de 'cuenta' en $a0

li $v0, 1 # Código de servicio para imprimir un entero

syscall # Llama al sistema para imprimir

li $v0, 10 # Código de salida del sistema

syscall # Llama al sistema para terminar el programa

### Dirección de comienzo del programa dentro del segmento de texto

El segmento de Texto comienza en 0x00400000 y acaba en este caso en 0x0040005c y por lo tanto ocupará un total de 104 Bytes

### Dirección de las variables usadas y sus respectivos bytes

#### Variables numéricas

* Variable vector *→* 0x10010000 y ocupa 64 bytes
* Variable cuenta *→*  0x10010040 y ocupa 4 bytes
* Variable i  *→* 0x10010044 y ocupa 4 bytes

#### Variables de caracteres

* Variable mensaje *→* 0x10010048 y ocupa 27 bytes (26 bytes + 1 byte para el elemento nulo)

### ¿Cómo introducimos los datos de entrada y cómo pueden verse los resultados?

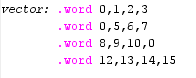
En este programa no introducimos datos en ningún momento si queremos cambiar el vector lo cambiamos directamente en el código. Por otro lado imprime el mensaje, el programa carga la dirección del mensaje "El valor del contador es: " en el registro $a0 y llama al sistema para imprimirlo. Esto se hace con el código de servicio 4.

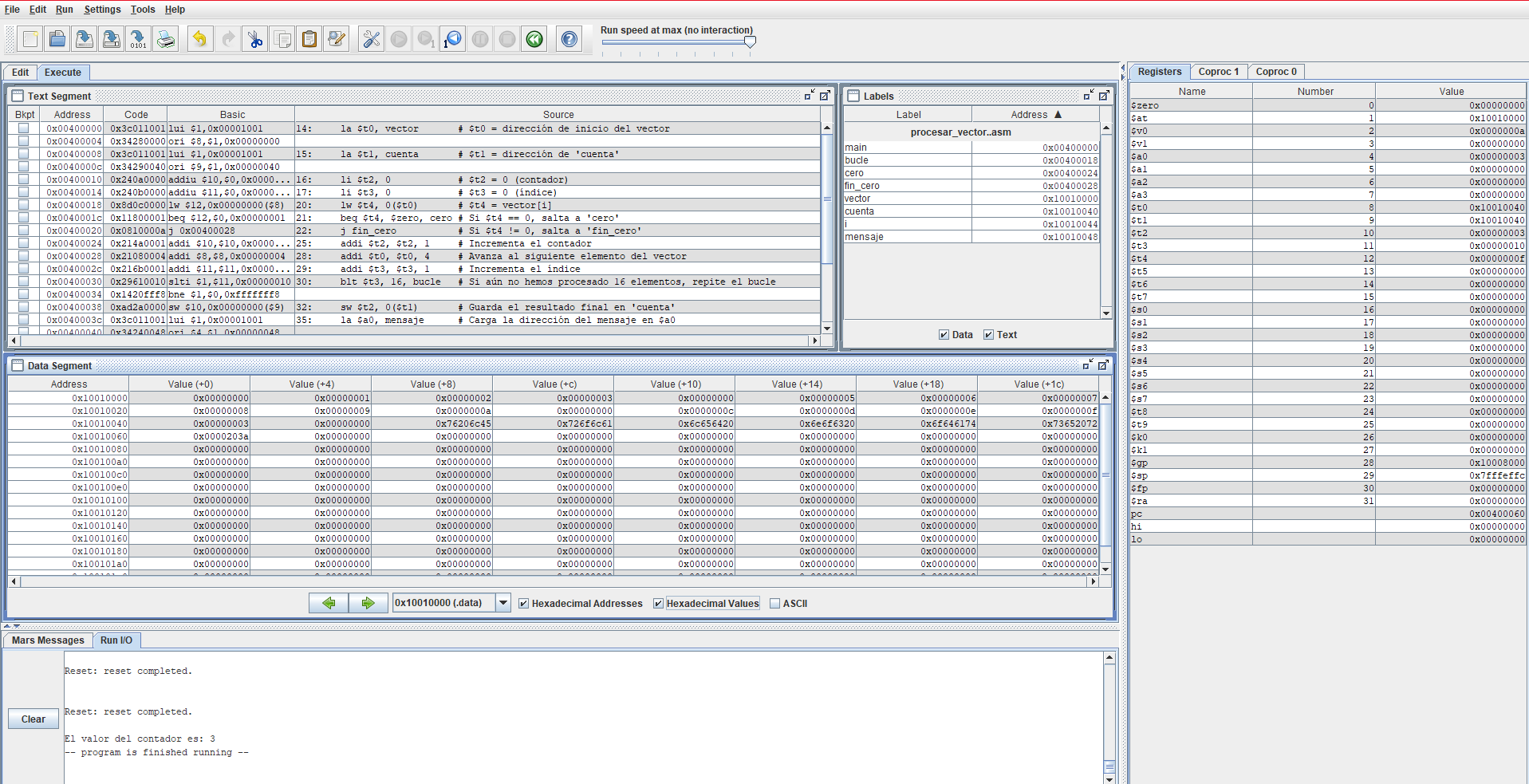
Imprime el valor del contador: Después de imprimir el mensaje, el programa carga el valor de cuenta en el registro $a0 y llama al sistema para imprimirlo. Esto se hace con el código de servicio 1.

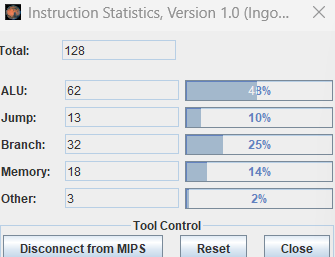
### Casos de prueba

#### Ejemplo 1

Este será el vector usado para el primer ejemplo

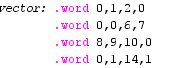


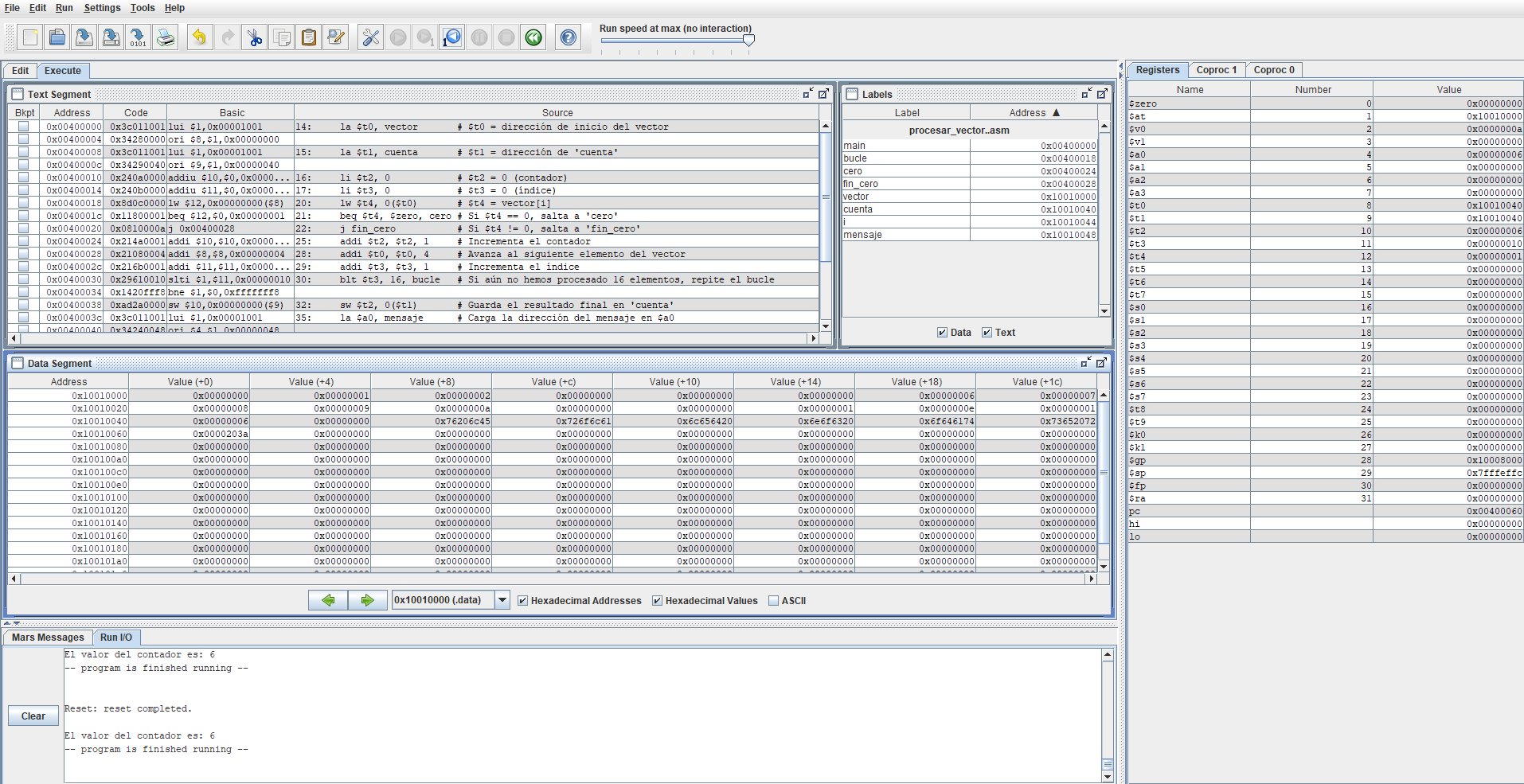


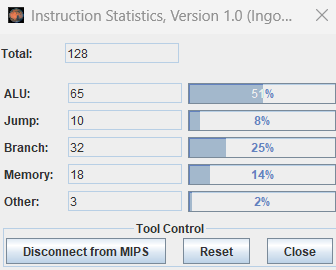


Ejemplo 2

Este es el vector del segundo ejemplo







## Ejercicio 4

En este apartado realizaremos un tratamiento de cadenas de texto, en el cual se convertirá una cadena minúscula a letra mayúscula. Y finalmente se contará el número de caracteres. Pediremos por consola la cadena y mostraremos por consola el resultado.

**Código:**

.data

prompt: .asciiz "Introduce una cadena de caracteres: "

output: .asciiz "La cadena en mayúsculas es: "

count\_msg: .asciiz "\nEl número de caracteres es: "

buffer: .space 256 # Espacio para almacenar la cadena de entrada

newline: .asciiz "\n"

.text

.globl main

main:

# Mostrar el prompt para la entrada de la cadena

li $v0, 4

la $a0, prompt

syscall

# Leer la cadena de entrada

li $v0, 8

la $a0, buffer

li $a1, 256

syscall

# Recorrer la cadena y convertir las minúsculas a mayúsculas

la $t0, buffer # Dirección de inicio de la cadena

li $t2, 0 # Inicializar contador de caracteres a 0

while\_loop:

lb $t1, 0($t0) # Cargar el carácter actual

beq $t1, 0, end\_while # Si es el terminador, salir del bucle

blt $t1, 'a', next\_char # Si el carácter es menor que 'a', pasar al siguiente

bgt $t1, 'z', next\_char # Si el carácter es mayor que 'z', pasar al siguiente

sub $t1, $t1, 32 # Convertir minúscula a mayúscula (restar 32)

sb $t1, 0($t0) # Almacenar el carácter convertido

next\_char:

addi $t0, $t0, 1 # Avanzar al siguiente carácter

addi $t2, $t2, 1 # Incrementar contador de caracteres

j while\_loop # Volver al inicio del bucle

end\_while:

# Restar 1 al contador de caracteres para ajustar

subi $t2, $t2, 1

# Mostrar la cadena en mayúsculas

li $v0, 4

la $a0, output

syscall

# Mostrar la cadena convertida

la $a0, buffer

li $v0, 4

syscall

# Salto de línea

li $v0, 4

la $a0, newline

syscall

# Mostrar el número de caracteres

li $v0, 4

la $a0, count\_msg

syscall

# Mostrar el valor del contador

move $a0, $t2

li $v0, 1

syscall

# Salto de línea

li $v0, 4

la $a0, newline

syscall

# Salir del programa

li $v0, 10

syscall

**Dirección de comienzo del programa dentro del segmento de texto**

Para la dirección del segmento de texto, empezamos desde 0x00400000 y duramos hasta 0x004000d0. Si restamos el tamaño final al tamaño inicial tenemos que el segmento de texto ocupa unos **3808** **bits** que son **476** **bytes.**

**¿Cómo funciona el bucle de este programa?**

El "loop" o bucle en este código es como una máquina que recorre cada letra de la cadena que el usuario ingresa. Cada vez que revisa una letra, la convierte a mayúscula si es minúscula y cuenta cuántas letras ha visto. Cuando llega al final de la cadena, la máquina deja de funcionar y muestra la cadena convertida junto con la cantidad total de letras que encontró.

**¿Cómo introducimos los datos de entrada y cómo pueden verse los resultados?**

Para ingresar los datos de entrada, el programa muestra un mensaje de prompt utilizando la instrucción `li $v0, 4` y `la $a0, prompt`, solicitando al usuario que introduzca una cadena de caracteres. Luego, utiliza la instrucción `li $v0, 8` para leer la cadena ingresada por el usuario en el espacio de datos reservado (`buffer`) utilizando la instrucción `la $a0, buffer`.

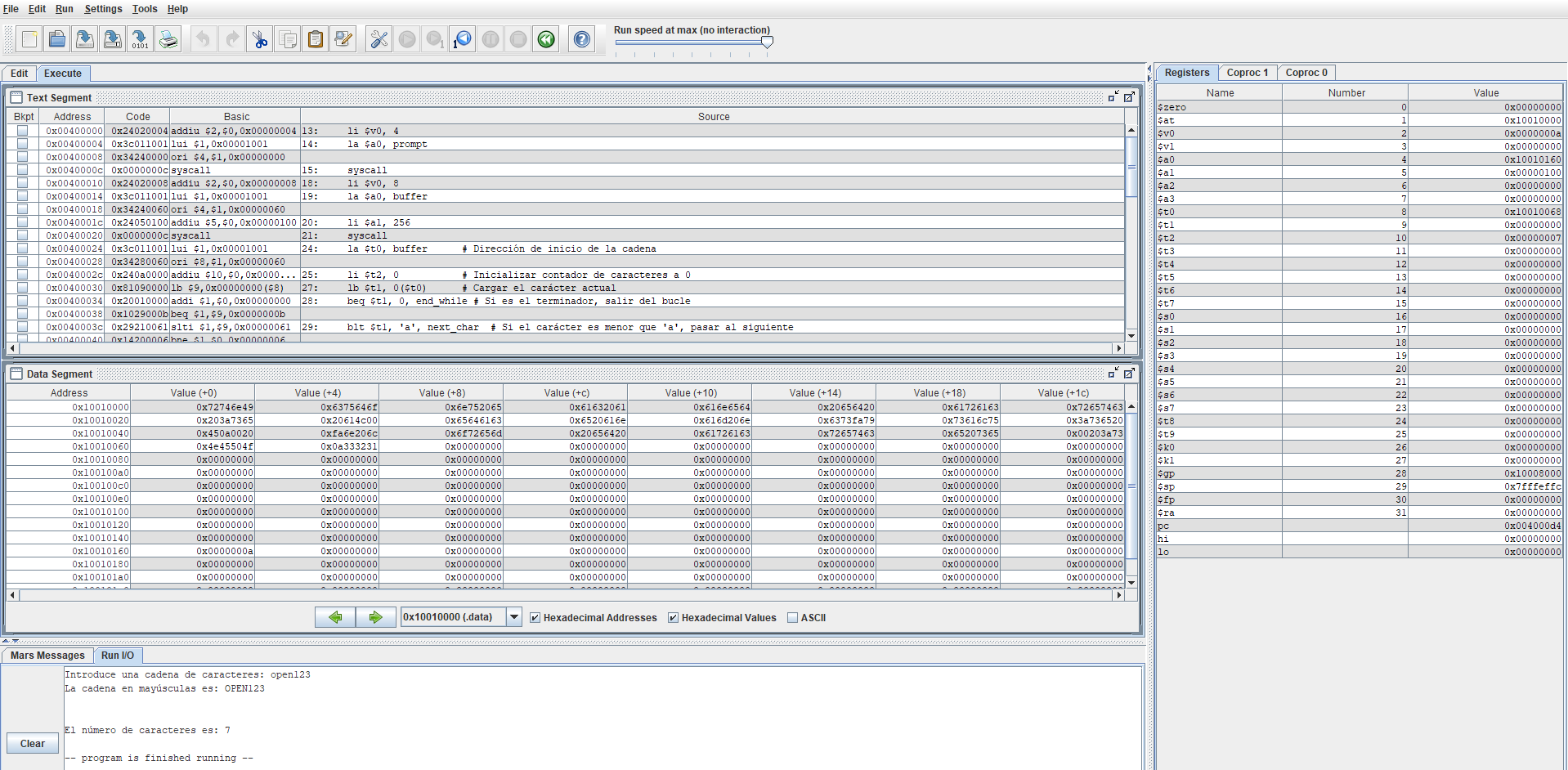
Para procesar la cadena, el programa utiliza un bucle que recorre cada carácter de la cadena. Dentro del bucle, verifica si el carácter es una letra minúscula y, en caso afirmativo, lo convierte a mayúscula utilizando las instrucciones `blt`, `bgt`, `sub`, y `sb`. Al mismo tiempo, cuenta el número de caracteres en la cadena utilizando la instrucción `addi $t2, $t2, 1`.

Una vez procesada la cadena, el programa muestra la cadena en mayúsculas utilizando la instrucción `li $v0, 4` y `la $a0, output`, y luego la cadena procesada utilizando `li $v0, 4` y `syscall`. Después, muestra el número de caracteres en la cadena utilizando las instrucciones `li $v0, 4`, `la $a0, count\_msg`, y `syscall`, seguido del valor del contador utilizando `move $a0, $t2`, `li $v0, 1`, y `syscall`.

Para terminar el programa muestra un salto de línea utilizando `li $v0, 4` y `la $a0, newline`, y luego finaliza con la instrucción `li $v0, 10` y `syscall`. Los resultados se muestran en la consola de salida.

**Caso de prueba**

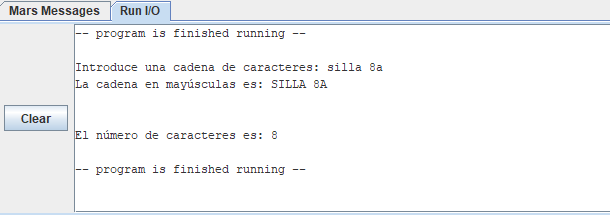
-Ejemplo 1



En el primer caso de uso, se introdujo por consola “open123” y se devolvió “OPEN123” y 7.

“OPEN123” siendo el texto en Mayúsculas y 7 la cuenta de caracteres, incluyendo 123.

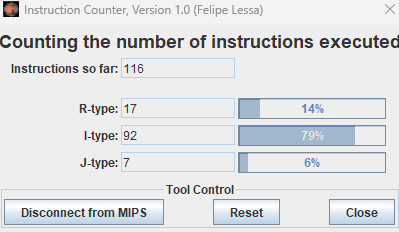
-Ejemplo 2



En este caso de uso, se introdujo por consola “silla 8a” y se devolvió “SILLA 8A” y 8.

“SILLA 8A” siendo el texto en Mayúsculas y 8 la cuenta de caracteres, incluyendo 8 y el espacio como carácter .

**Número de instrucciones**



Podemos apreciar un número total de 116 instrucciones en un ejemplo de 6 caracteres, ya que dependiendo de las veces que se ejecute el bucle se ejecutan más instrucciones.

**EJERCICIO 5:**

Se muestra a continuación el código fuente que hemos creado para resolver la ecuación planteada: para números en coma flotante.

***.data***

***x: .float 0.0***# espacio para la variable x

***y: .float 0.0***# espacio para la variable y

***z: .float 0.0***# espacio para la variable z

***w: .float 0.0***# espacio para la variable w

***three: .float 3.0***# espacio para el número 3

***frase\_x: .asciiz "Dame el valor de x: "*** # Guardamos la cadena “Dame el valor de x: “ en frase\_x

***frase\_y: .asciiz "Dame el valor de y: "*** # Guardamos la cadena “Dame el valor de y: “ en frase\_y

***frase\_z: .asciiz "Dame el valor de z: "*** # Guardamos la cadena “Dame el valor de z: “ en frase\_z

***salida\_W: .asciiz "La solución es: "***  # Guardamos la cadena “La solución es: “ en salida\_W

***.text***

# Entrada de valores para x

***la $a0, frase\_x*** # Carga la dirección de la cadena "Dame el valor de x: " en el registro $a0

***li $v0, 4*** # Carga el valor 4 en el registro $V0, código de servicio para imprimir una cadena

***syscall***# Llamada al sistema para imprimir la cadena

***li $v0, 6***# Cargar el valor 6 en el registro $v0, código de servicio para leer un número de punto flotante

***syscall***# LLamada al sistema para leer el número de coma flotante

***s.s $f0, x***# Almacena el valor de x en el registro de coma flotante $f0

# Entrada de valores para y

***la $a0, frase\_y***# Carga la dirección de la cadena "Dame el valor de y: " en el registro $a0

***li $v0, 4***# Carga el valor 4 en el registro $V0, código de servicio para imprimir una cadena

***syscall***# Llamada al sistema para imprimir la cadena

***li $v0, 6***# Cargar el valor 6 en el registro $v0, código de servicio para leer un número de punto flotante

***syscall*** *#* LLamada al sistema para leer el número de coma flotante

***s.s $f0, y***# Almacena el valor de y en el registro de coma flotante $f0

# Entrada de valores para z

***la $a0, frase\_z***# Carga la dirección de la cadena "Dame el valor de z: " en el registro $a0

***li $v0, 4***# Carga el valor 4 en el registro $V0, código de servicio para imprimir una cadena

***syscall***# Llamada al sistema para imprimir la cadena

***li $v0, 6***# Cargar el valor 6 en el registro $v0, código de servicio para leer un número de punto flotante

***syscall***# LLamada al sistema para leer el número de coma flotante

***s.s $f0, z***# Almacena el valor de z en el registro de coma flotante $f0

***main:***

***l.s $f0,x***# carga el valor de x en el registro coma flotante $f0

***l.s $f1,y***# carga el valor de y en el registro coma flotante $f1

***mul.s $f2,$f1,$f1***# Realiza la operación y\*y, almacena el resultado en $f2 registro de coma flotante

***l.s $f3,z***# carga el valor de z en el registro coma flotante $f3

***l.s $f5,three***# carga el valor de three en el registro coma flotante $f5

***mul.s $f4,$f3,$f5***# Realiza la operación 3\*z, almacena el resultado en $f4 registro de coma flotante

***add.s $f2,$f2,$f4***# Realiza la operación y\*y + 3\*z, almacena el resultado en $f2 registro de coma flotante

***add.s $f2,$f0,$f2***# Realiza la operación x + y\*y + 3\*z, almacena el resultado en $f4

***s.s $f2,w***# Almacena el valor de w en el registro coma flotante $f2

# Mostramos el valor final de w

***la $a0, salida\_W***# Carga la dirección de la cadena "La solución es: " en el registro $s0

***li $v0, 4***# Carga el valor 4 en el registro $V0, código de servicio para imprimir una cadena

***syscall*** *#* Llamada al sistema para imprimir la cadena

***l.s $f12, w*** *#* Carga el valor de w en el registro coma flotante $f12

***li $v0, 2*** *#* Carga el valor 2 en el registro $v0, código de servicio para imprimir un número en coma flotante

***syscall*** *#* LLamada al sistema para imprimir el número en coma flotante

# Terminamos el programa

***li $v0, 17*** *#* Carga el valor 17 en el registro $v0, código de servicio para terminar la ejecución

***syscall*** *#* LLamada al sistema para terminar la ejecución

Tras analizar el código, vamos a prestar cierto interés a datos que podemos extraer en el apartado “Execute” dentro del programa MARS. Así como la dirección de comienzo del programa y el número de bytes que ocupa, las direcciones de las variables usadas y sus respectivos Bytes.

**Dirección de comienzo del programa dentro del segmento de texto**

Para la dirección del segmento de texto, empezamos desde la dirección 0x00400000 con la instrucción la $a0, frase\_x, y este segmento del programa se ejecuta hasta la dirección 0x004000bc, donde justamente finaliza el programa. Para calcular el número total de Bytes tendremos que multiplicar cada instrucción por su tamaño y finalmente sumar todo. Todas las instrucciones que componen nuestro programa ocupan 4 Bytes, y tenemos un total de 37 instrucciones (28 instrucciones y 9 declaraciones), al multiplicarlo por 4 obtenemos 148 Bytes.

Y teniendo en cuenta el espacio que ocupa cada variable (desarrollado en el próximo apartado) obtenemos 100 Bytes más. Por ende, 100+148 = 248 Bytes de memoria ocupa el programa.

**Dirección de las variables usadas y sus respectivos bytes**

***Variables numéricas***

Las variables con la instrucción *.float* ocupan un total de 4 Bytes. Por ello todas nuestras variables con instrucción *.float* van a ocupar un espacio en memoria equivalente a 4 Bytes

*Variable x → 0x10010000*

*Variable y → 0x10010004*

*Variable z → 0x10010008*

*Variable w → 0x1001000c*

*Variable three → 0x10010010*

***Variables de caracteres***

Las variables con la instrucción *.asciiz* ocupan un total de bytes que se calcula con el número total de caracteres de la cadena + 1 Byte por el carácter nulo de final de cadena.

Por ello y como todas nuestras variables *.asciiz* están compuestas por la misma cadena, a excepción de la variable: “Dame el valor de (variable): “, todas ellas a excepción de la variable salida\_W ocupan el mismo espacio en memoria. Un total de 21 Bytes.

*frase\_x → 0x10010014*

*frase\_y → 0x10010029*

*frase\_z → 0x1001003e*

*salida\_W → 0x10010053* Ocupa un total de 17 Bytes (16 de la cadena + 1 del carácter nulo)

**¿Cómo introducimos los datos de entrada y cómo pueden verse los resultados?**

Para introducir los datos de entrada y mostrar los resultados, primero debemos declarar las variables y

cadenas de texto en el apartado *.data* .

Después, en el apartado *.text* se realiza la entrada de datos, y se hace de la siguiente forma:

1º Cargamos la dirección de la cadena de texto *frase\_x* en el registro *$a0*.

2º Realizamos una llamada al sistema con el *código de servicio 4* para imprimir la cadena de texto.

3º Realizamos otra llamada al sistema, esta vez con el *código de servicio 6* para leer un número en coma flotante mediante la entrada del usuario.

4º Finalmente, guardamos el número leído en la variable x mediante la instrucción *s.s*.

5º Repetimos este proceso para las variables *y z*.

En la función main cargamos los valores de *x, y, z* en los registros *$f0, $f1 y $f3* y realizamos las operaciones mediante las instrucciones pertinentes.

Por último, para mostrar el resultado:

1º Cargamos la dirección de la cadena de texto *salida\_W* en el registro *$a0*.

2º Realizamos una llamada al sistema con el *código de servicio 4* para imprimir la cadena de texto.

3º Cargamos el valor de la variable w en el registro *$f12*.

4º Realizamos una llamada al sistema con el *código de servicio 2* para imprimir el número en coma flotante.

5º Realizamos la última llamada al sistema con el *código de servicio 17* para terminar la ejecución del programa.

**Casos de prueba**

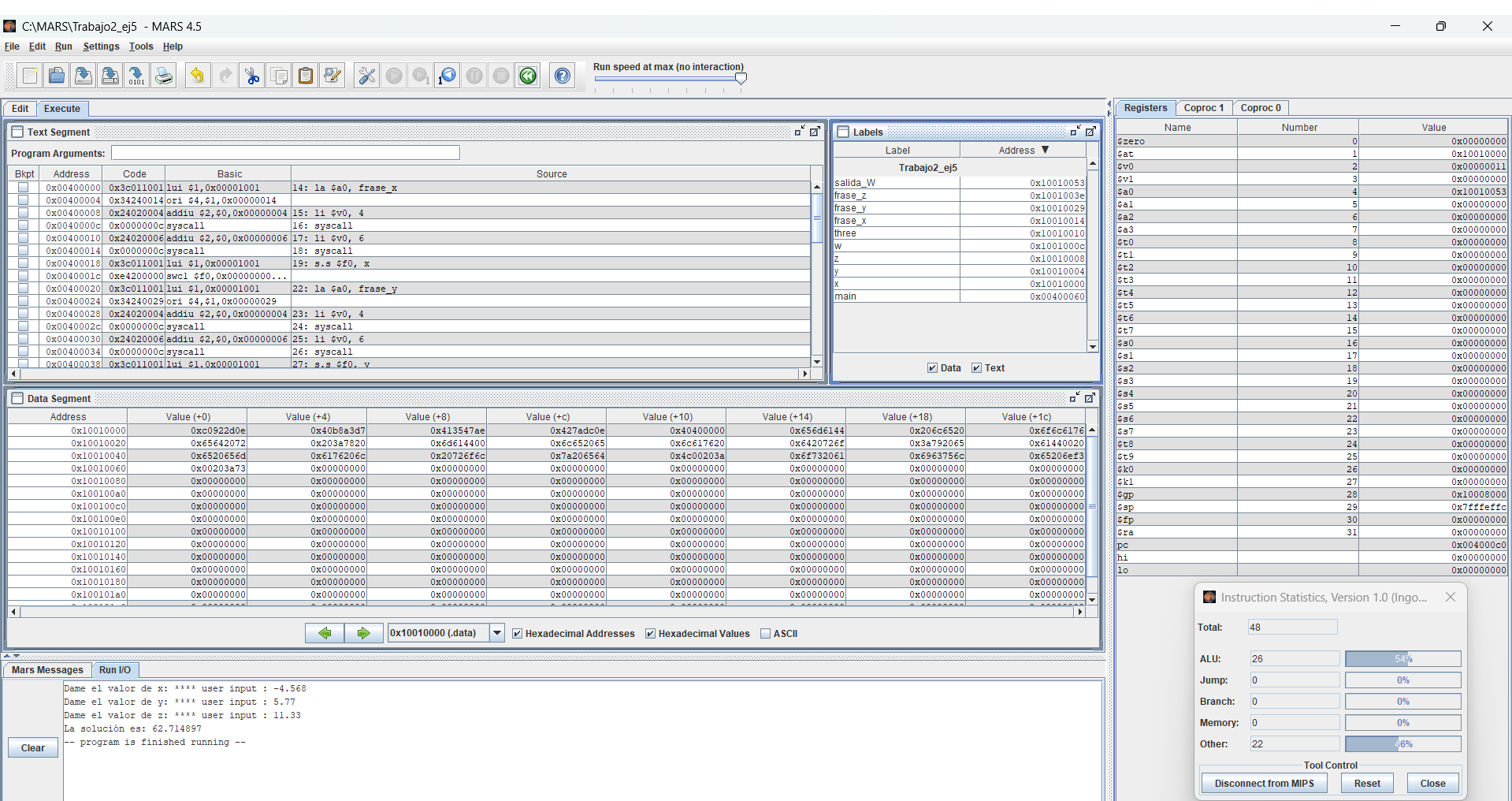
A continuación vamos a ver varios ejemplos del desarrollo del programa con las tablas que incluyen el segmento texto y dato, ventana de etiquetas, banco de registros y banco de registros de coma flotante.

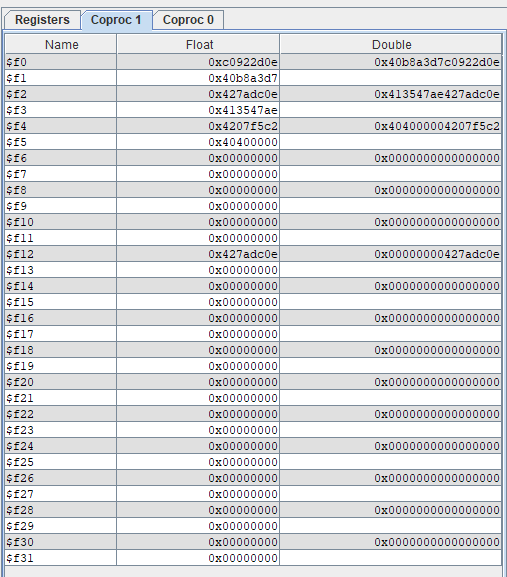
*ejemplo 1:*

Valores de las variables { x = -4.568 y = 5.77 z = 11.33 }

Resultado { 62.714897 }

Instrucciones{ 48 }



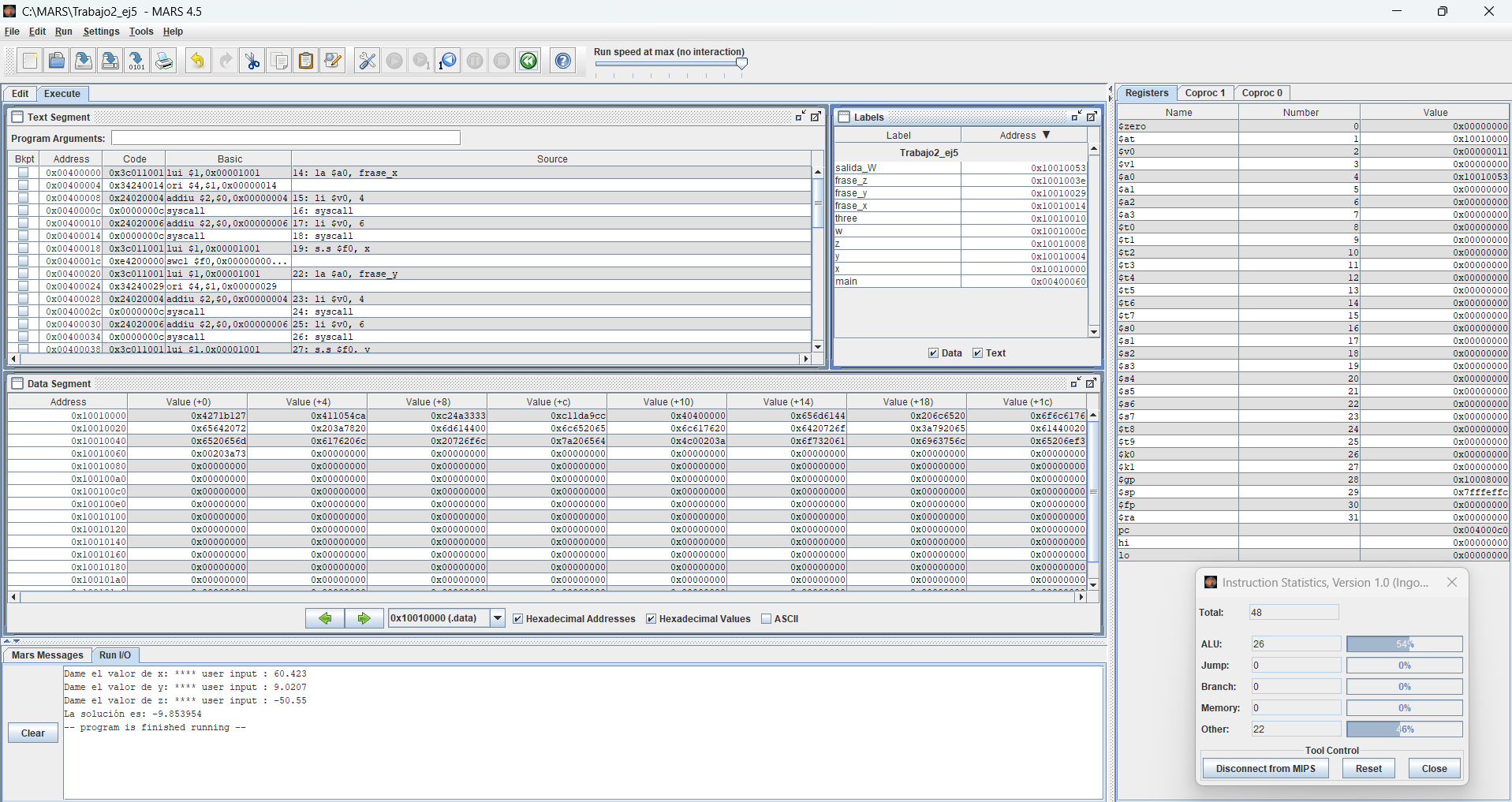


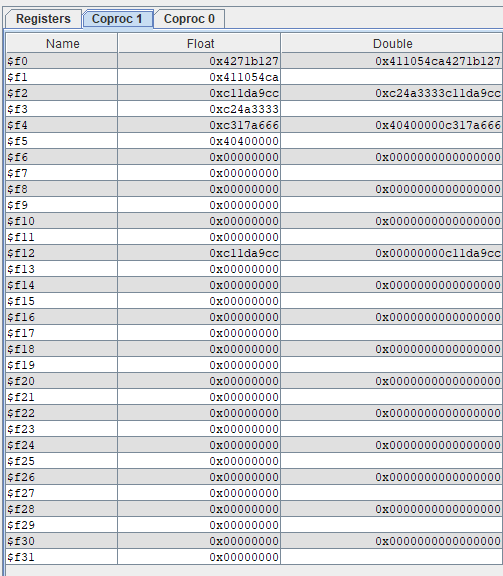
*ejemplo 2:*

Valores de las variables { x = 60.423 y = 9.0207 z = -50.55 }

Resultado{ -9.853954 }

Instrucciones{ 48 }





*ejemplo 3:*

Valores de las variables { x = -5023.26 y = 76.543 z = -111.0303 }

Resultado{ 502.47998 }

Instrucciones{ 48 }

