



Universidad
Rey Juan Carlos

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: $w = x + y^2 + z \cdot 3$

```
.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

[illegible]

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

The screenshot displays the Mars MIPS simulator interface. The main window shows the assembly code for 'enunciado2' with instructions like 'li \$a0, frase_x', 'syscall', and 'sw \$v0, x'. The registers window on the right shows the state of registers \$zero through \$t6. The 'Mars Messages' window at the bottom shows the execution progress, including the completion of the program. The 'Instruction Statistics' window shows the usage of ALU, Jump, Branch, Memory, and Other instructions.

Ejercicio 2:

Código

```
.data
#Reservamos espacio para las variables n y suma.
n:      .space4          # Copia en $s0
suma:   .space4          # 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)
      #      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

The screenshot displays the MARS MIPS simulator interface. The main window is titled "C:\Users\usuario\Downloads\mips1.asm - MARS 4.5". The menu bar includes File, Edit, Run, Settings, Tools, and Help. The toolbar contains various icons for file operations, execution, and debugging.

The **Text Segment** panel shows the assembly code for the program, with the following instructions:

```
0x00400000: 0x3c011001 lui $1,4097
0x00400004: 0x34240006 ori $4,$1,8
0x00400008: 0x24020004 addiu $2,$0,4
0x0040000c: 0x0000000c syscall
0x00400010: 0x24020005 addiu $2,$0,5
0x00400014: 0x0000000c syscall
0x00400018: 0x3c011001 lui $1,4097
0x0040001c: 0x24020000 sw $2,0($1)
0x00400020: 0x24040001 addiu $10,$0,1
0x00400024: 0x24110001 addiu $17,$0,1
0x00400028: 0x24040003 addiu $5,$0,3
0x0040002c: 0x3c011001 lui $1,4097
0x00400030: 0x8c300000 lw $16,0($1)
0x00400034: 0x020802a1 slt $1,$16,$0
0x00400038: 0x1420000b bne $1,$0,$1
0x0040003c: 0x20010002 addi $1,$0,2
0x00400040: 0x71414805 mul $5,$10,$1
```

The **Labels** panel shows the following labels:

```
main 0x00400020
while 0x0040002c
if 0x00400044
end_if 0x00400058
inc 0x00400060
end_while 0x00400068
end_main 0x0040009c
n 0x10010000
suma 0x10010004
frase_x 0x10010008
salida_w 0x1001001d
```

The **Data Segment** panel shows the memory layout, with the following values:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	10	175	1701667140	543974698	1869373814	1701060722	540700192	543247360
0x10010004	1970040691	1852755235	909641066	32	0	0	0	0
0x10010008	0	0	0	0	0	0	0	0
0x1001000c	0	0	0	0	0	0	0	0
0x10010010	0	0	0	0	0	0	0	0
0x10010014	0	0	0	0	0	0	0	0
0x10010018	0	0	0	0	0	0	0	0
0x1001001c	0	0	0	0	0	0	0	0
0x10010020	0	0	0	0	0	0	0	0
0x10010024	0	0	0	0	0	0	0	0
0x10010028	0	0	0	0	0	0	0	0
0x1001002c	0	0	0	0	0	0	0	0
0x10010030	0	0	0	0	0	0	0	0
0x10010034	0	0	0	0	0	0	0	0
0x10010038	0	0	0	0	0	0	0	0
0x1001003c	0	0	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0	0	0

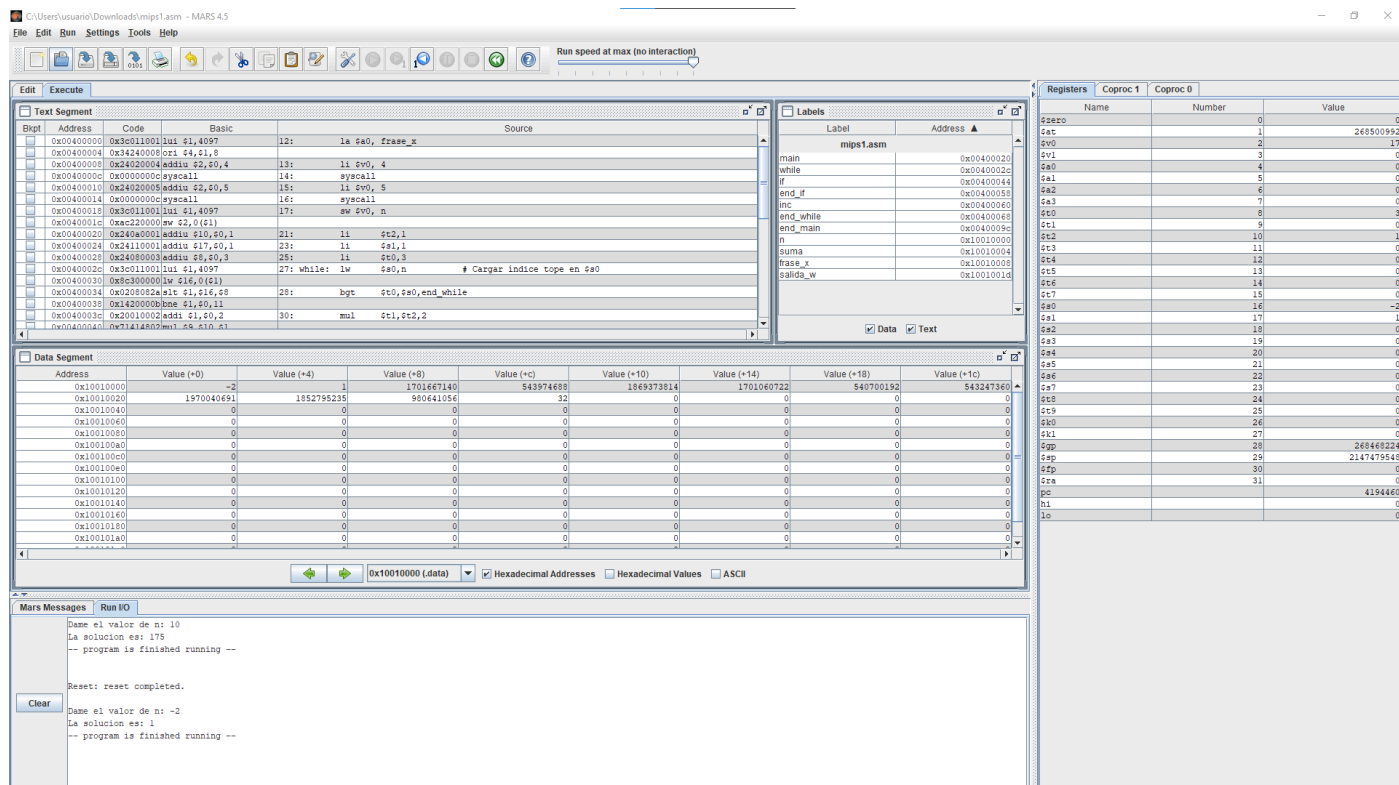
The **Registers** panel shows the state of the registers, with the following values:

Name	Number	Value
\$zero	0	0
\$at	1	268500992
\$v0	2	17
\$v1	3	0
\$a0	4	0
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	11
\$t1	9	86
\$t2	10	86
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$t8	16	10
\$s0	17	175
\$s1	18	0
\$s2	19	0
\$s3	20	0
\$s4	21	0
\$s5	22	0
\$s6	23	0
\$s7	24	0
\$s8	25	0
\$s9	26	0
\$k0	27	0
\$k1	28	0
\$gp	29	268460324
\$sp	30	2147479548
\$fp	31	0
\$ra	32	4194460
\$pc	33	0
\$hi	34	0
\$lo	35	5

The **Messages** panel shows the following output:

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


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

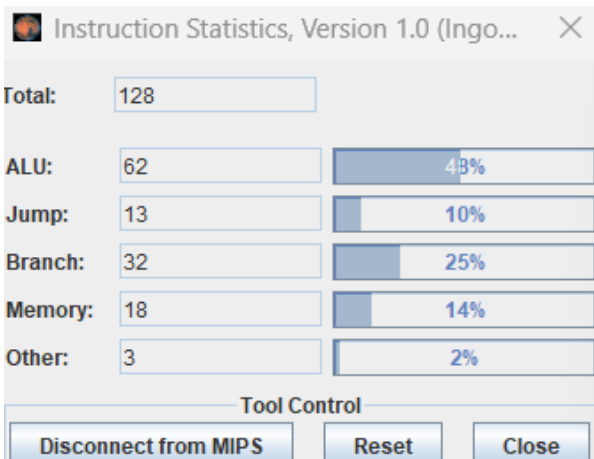
Este será el vector usado para el primer ejemplo

```
vector: .word 0,1,2,3
        .word 0,5,6,7
        .word 8,9,10,0
        .word 12,13,14,15
```

The screenshot displays the Mars debugger interface. The main window shows assembly code with columns for Step, Address, Code, Basic, and Source. The code is for a program that prints the value of a counter and a message. The registers window on the right shows the current state of registers, with \$a0 containing 0. The data segment window at the bottom shows the memory layout, including the vector array and the message string.

Register	Value
\$zero	0x00000000
\$at	0x10010000
\$v0	0x00000004
\$v1	0x00000000
\$a0	0x00000000
\$a1	0x00000000
\$a2	0x00000000
\$a3	0x00000000
\$a4	0x00000000
\$a5	0x00000000
\$a6	0x00000000
\$a7	0x00000000
\$s0	0x00000000
\$s1	0x00000000
\$s2	0x00000000
\$s3	0x00000000
\$s4	0x00000000
\$s5	0x00000000
\$s6	0x00000000
\$s7	0x00000000
\$t0	0x00000000
\$t1	0x00000000
\$t2	0x00000000
\$t3	0x00000000
\$t4	0x00000000
\$t5	0x00000000
\$t6	0x00000000
\$t7	0x00000000
\$f0	0x00000000
\$f1	0x00000000
\$f2	0x00000000
\$f3	0x00000000
\$f4	0x00000000
\$f5	0x00000000
\$f6	0x00000000
\$f7	0x00000000

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
0x10010000	0x00000000	0x00000001	0x00000002	0x00000003	0x00000000	0x00000005	0x00000006	0x00000007
0x10010004	0x00000000	0x00000005	0x00000006	0x00000007	0x00000008	0x00000009	0x0000000A	0x0000000B
0x10010008	0x0000000C	0x0000000D	0x0000000E	0x0000000F	0x00000010	0x00000011	0x00000012	0x00000013
0x1001000C	0x00000014	0x00000015	0x00000016	0x00000017	0x00000018	0x00000019	0x0000001A	0x0000001B
0x1001000F	0x0000001C	0x0000001D	0x0000001E	0x0000001F	0x00000020	0x00000021	0x00000022	0x00000023
0x10010010	0x00000024	0x00000025	0x00000026	0x00000027	0x00000028	0x00000029	0x0000002A	0x0000002B
0x10010014	0x0000002C	0x0000002D	0x0000002E	0x0000002F	0x00000030	0x00000031	0x00000032	0x00000033
0x10010018	0x00000034	0x00000035	0x00000036	0x00000037	0x00000038	0x00000039	0x0000003A	0x0000003B
0x1001001C	0x0000003C	0x0000003D	0x0000003E	0x0000003F	0x00000040	0x00000041	0x00000042	0x00000043
0x10010020	0x00000044	0x00000045	0x00000046	0x00000047	0x00000048	0x00000049	0x0000004A	0x0000004B
0x10010024	0x0000004C	0x0000004D	0x0000004E	0x0000004F	0x00000050	0x00000051	0x00000052	0x00000053
0x10010028	0x00000054	0x00000055	0x00000056	0x00000057	0x00000058	0x00000059	0x0000005A	0x0000005B
0x1001002C	0x0000005C	0x0000005D	0x0000005E	0x0000005F	0x00000060	0x00000061	0x00000062	0x00000063
0x10010030	0x00000064	0x00000065	0x00000066	0x00000067	0x00000068	0x00000069	0x0000006A	0x0000006B
0x10010034	0x0000006C	0x0000006D	0x0000006E	0x0000006F	0x00000070	0x00000071	0x00000072	0x00000073
0x10010038	0x00000074	0x00000075	0x00000076	0x00000077	0x00000078	0x00000079	0x0000007A	0x0000007B
0x1001003C	0x0000007C	0x0000007D	0x0000007E	0x0000007F	0x00000080	0x00000081	0x00000082	0x00000083
0x10010040	0x00000084	0x00000085	0x00000086	0x00000087	0x00000088	0x00000089	0x0000008A	0x0000008B
0x10010044	0x0000008C	0x0000008D	0x0000008E	0x0000008F	0x00000090	0x00000091	0x00000092	0x00000093
0x10010048	0x00000094	0x00000095	0x00000096	0x00000097	0x00000098	0x00000099	0x0000009A	0x0000009B
0x1001004C	0x0000009C	0x0000009D	0x0000009E	0x0000009F	0x000000A0	0x000000A1	0x000000A2	0x000000A3
0x10010050	0x000000A4	0x000000A5	0x000000A6	0x000000A7	0x000000A8	0x000000A9	0x000000AA	0x000000AB
0x10010054	0x000000AC	0x000000AD	0x000000AE	0x000000AF	0x000000B0	0x000000B1	0x000000B2	0x000000B3
0x10010058	0x000000B4	0x000000B5	0x000000B6	0x000000B7	0x000000B8	0x000000B9	0x000000BA	0x000000BB
0x1001005C	0x000000BC	0x000000BD	0x000000BE	0x000000BF	0x000000C0	0x000000C1	0x000000C2	0x000000C3
0x10010060	0x000000C4	0x000000C5	0x000000C6	0x000000C7	0x000000C8	0x000000C9	0x000000CA	0x000000CB
0x10010064	0x000000CC	0x000000CD	0x000000CE	0x000000CF	0x000000D0	0x000000D1	0x000000D2	0x000000D3
0x10010068	0x000000D4	0x000000D5	0x000000D6	0x000000D7	0x000000D8	0x000000D9	0x000000DA	0x000000DB
0x1001006C	0x000000DC	0x000000DD	0x000000DE	0x000000DF	0x000000E0	0x000000E1	0x000000E2	0x000000E3
0x10010070	0x000000E4	0x000000E5	0x000000E6	0x000000E7	0x000000E8	0x000000E9	0x000000EA	0x000000EB
0x10010074	0x000000EC	0x000000ED	0x000000EE	0x000000EF	0x000000F0	0x000000F1	0x000000F2	0x000000F3
0x10010078	0x000000F4	0x000000F5	0x000000F6	0x000000F7	0x000000F8	0x000000F9	0x000000FA	0x000000FB
0x1001007C	0x000000FC	0x000000FD	0x000000FE	0x000000FF	0x00000100	0x00000101	0x00000102	0x00000103
0x10010080	0x00000104	0x00000105	0x00000106	0x00000107	0x00000108	0x00000109	0x0000010A	0x0000010B
0x10010084	0x0000010C	0x0000010D	0x0000010E	0x0000010F	0x00000110	0x00000111	0x00000112	0x00000113
0x10010088	0x00000114	0x00000115	0x00000116	0x00000117	0x00000118	0x00000119	0x0000011A	0x0000011B
0x1001008C	0x0000011C	0x0000011D	0x0000011E	0x0000011F	0x00000120	0x00000121	0x00000122	0x00000123
0x10010090	0x00000124	0x00000125	0x00000126	0x00000127	0x00000128	0x00000129	0x0000012A	0x0000012B
0x10010094	0x0000012C	0x0000012D	0x0000012E	0x0000012F	0x00000130	0x00000131	0x00000132	0x00000133
0x10010098	0x00000134	0x00000135	0x00000136	0x00000137	0x00000138	0x00000139	0x0000013A	0x0000013B
0x1001009C	0x0000013C	0x0000013D	0x0000013E	0x0000013F	0x00000140	0x00000141	0x00000142	0x00000143
0x100100A0	0x00000144	0x00000145	0x00000146	0x00000147	0x00000148	0x00000149	0x0000014A	0x0000014B
0x100100A4	0x0000014C	0x0000014D	0x0000014E	0x0000014F	0x00000150	0x00000151	0x00000152	0x00000153
0x100100A8	0x00000154	0x00000155	0x00000156	0x00000157	0x00000158	0x00000159	0x0000015A	0x0000015B
0x100100AC	0x0000015C	0x0000015D	0x0000015E	0x0000015F	0x00000160	0x00000161	0x00000162	0x00000163
0x100100B0	0x00000164	0x00000165	0x00000166	0x00000167	0x00000168	0x00000169	0x0000016A	0x0000016B
0x100100B4	0x0000016C	0x0000016D	0x0000016E	0x0000016F	0x00000170	0x00000171	0x00000172	0x00000173
0x100100B8	0x00000174	0x00000175	0x00000176	0x00000177	0x00000178	0x00000179	0x0000017A	0x0000017B
0x100100BC	0x0000017C	0x0000017D	0x0000017E	0x0000017F	0x00000180	0x00000181	0x00000182	0x00000183
0x100100C0	0x00000184	0x00000185	0x00000186	0x00000187	0x00000188	0x00000189	0x0000018A	0x0000018B
0x100100C4	0x0000018C	0x0000018D	0x0000018E	0x0000018F	0x00000190	0x00000191	0x00000192	0x00000193
0x100100C8	0x00000194	0x00000195	0x00000196	0x00000197	0x00000198	0x00000199	0x0000019A	0x0000019B
0x100100CC	0x0000019C	0x0000019D	0x0000019E	0x0000019F	0x000001A0	0x000001A1	0x000001A2	0x000001A3
0x100100D0	0x000001A4	0x000001A5	0x000001A6	0x000001A7	0x000001A8	0x000001A9	0x000001AA	0x000001AB
0x100100D4	0x000001AC	0x000001AD	0x000001AE	0x000001AF	0x000001B0	0x000001B1	0x000001B2	0x000001B3
0x100100D8	0x000001B4	0x000001B5	0x000001B6	0x000001B7	0x000001B8	0x000001B9	0x000001BA	0x000001BB
0x100100DC	0x000001BC	0x000001BD	0x000001BE	0x000001BF	0x000001C0	0x000001C1	0x000001C2	0x000001C3
0x100100E0	0x000001C4	0x000001C5	0x000001C6	0x000001C7	0x000001C8	0x000001C9	0x000001CA	0x000001CB
0x100100E4	0x000001CC	0x000001CD	0x000001CE	0x000001CF	0x000001D0	0x000001D1	0x000001D2	0x000001D3
0x100100E8	0x000001D4	0x000001D5	0x000001D6	0x000001D7	0x000001D8	0x000001D9	0x000001DA	0x000001DB
0x100100EC	0x000001DC	0x000001DD	0x000001DE	0x000001DF	0x000001E0	0x000001E1	0x000001E2	0x000001E3
0x100100F0	0x000001E4	0x000001E5	0x000001E6	0x000001E7	0x000001E8	0x000001E9	0x000001EA	0x000001EB
0x100100F4	0x000001EC	0x000001ED	0x000001EE	0x000001EF	0x000001F0	0x000001F1	0x000001F2	0x000001F3
0x100100F8	0x000001F4	0x000001F5	0x000001F6	0x000001F7	0x000001F8	0x000001F9	0x000001FA	0x000001FB
0x100100FC	0x000001FC	0x000001FD	0x000001FE	0x000001FF	0x00000200	0x00000201	0x00000202	0x00000203
0x10010100	0x00000204	0x00000205	0x00000206	0x00000207	0x00000208	0x00000209	0x0000020A	0x0000020B
0x10010104	0x0000020C	0x0000020D	0x0000020E	0x0000020F	0x00000210	0x00000211	0x00000212	0x00000213
0x10010108	0x00000214	0x00000215	0x00000216	0x00000217	0x00000218	0x00000219	0x0000021A	0x0000021B
0x1001010C	0x0000021C	0x0000021D	0x0000021E	0x0000021F	0x00000220	0x00000221	0x00000222	0x00000223
0x10010110	0x00000224	0x00000225	0x00000226	0x00000227	0x00000228	0x00000229	0x0000022A	0x0000022B
0x10010114	0x0000022C	0x0000022D	0x0000022E	0x0000022F	0x00000230	0x00000231	0x00000232	0x00000233
0x10010118	0x00000234	0x00000235	0x00000236	0x00000237	0x00000238	0x00000239	0x0000023A	0x0000023B
0x1001011C	0x0000023C	0x0000023D	0x0000023E	0x0000023F	0x00000240	0x00000241	0x00000242	0x00000243
0x10010120	0x00000244	0x00000245	0x00000246	0x00000247	0x00000248	0x00000249	0x0000024A	0x0000024B
0x10010124	0x0000024C	0x0000024D	0x0000024E	0x0000024F	0x00000250	0x00000251	0x00000252	0x00000253
0x10010128	0x00000254	0x00000255	0x00000256	0x00000257	0x00000258	0x00000259	0x0000025A	0x0000025B
0x1001012C	0x0000025C	0x0000025D	0x0000025E	0x0000025F	0x00000260	0x00000261	0x00000262	0x00000263
0x10010130	0x000							



Ejemplo 2

Este es el vector del segundo ejemplo

```
vector: .word 0,1,2,0
        .word 0,0,6,7
        .word 8,9,10,0
        .word 0,1,14,1
```

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Execute

Text Segment

Byte	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x3e011001	lui \$1,0x00000100	14: la \$t0, vector # \$t0 = dirección de inicio del vector
<input type="checkbox"/>	0x00400004	0x34200000	ori \$2,\$1,0x00000000	
<input type="checkbox"/>	0x00400008	0x3e011001	lui \$1,0x00000100	15: la \$t1, cuenta # \$t1 = dirección de 'cuenta'
<input type="checkbox"/>	0x0040000c	0x34200040	ori \$9,\$1,0x00000040	
<input type="checkbox"/>	0x00400010	0x24a00000	addiu \$10,\$0,0x00000000	16: li \$t2, 0 # \$t2 = 0 (contador)
<input type="checkbox"/>	0x00400014	0x24180000	addiu \$11,\$0,0x00000000	17: li \$t3, 0 # \$t3 = 0 (índice)
<input type="checkbox"/>	0x00400018	0x8d0c0000	lw \$t4,0(\$t0)	20: lw \$t4, 0(\$t0) # \$t4 = vector[\$i]
<input type="checkbox"/>	0x0040001c	0x11800001	beg \$t2,\$0,0x00000001	21: beg \$t2, \$zero, cero # Si \$t4 == 0, salta a 'cero'
<input type="checkbox"/>	0x00400020	0x0810000a	0x00400028	22: j fin_cero # Si \$t4 != 0, salta a 'fin_cero'
<input type="checkbox"/>	0x00400024	0x21a00001	addi \$t2,\$t2,1	25: addi \$t2, \$t2, 1 # Incrementa el contador
<input type="checkbox"/>	0x00400028	0x21080004	addi \$t2,\$t2,4	26: addi \$t2, \$t2, 4 # Avanza al siguiente elemento del vector
<input type="checkbox"/>	0x0040002c	0x21e00001	addi \$t1,\$t1,1	29: addi \$t1, \$t1, 1 # Incrementa el índice
<input type="checkbox"/>	0x00400030	0x24e10010	slti \$t3,\$t3,1	30: blt \$t3, 16, bucle # Si aún no hemos procesado 16 elementos, repite el bucle
<input type="checkbox"/>	0x00400034	0x1400ffff	bne \$t3,\$0,0xffffffff	
<input type="checkbox"/>	0x00400038	0xad2a0000	sw \$t0,0(\$t1)	32: sw \$t2, 0(\$t1) # Guarda el resultado final en 'cuenta'
<input type="checkbox"/>	0x0040003c	0x3c011001	lui \$1,0x00000100	35: la \$a0, mensaje # Carga la dirección del mensaje en \$a0
<input type="checkbox"/>	0x00400040	0x34200040	ori \$2,\$1,0x00000040	

Labels

Label	Address
main	0x00400000
bucle	0x00400018
cero	0x00400024
fin_cero	0x00400028
vector	0x10010000
cuenta	0x10010040
mensaje	0x10010044

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x0000000a
\$v1	3	0x00000000
\$a0	4	0x00000006
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x10010040
\$t1	9	0x10010040
\$t2	10	0x00000006
\$t3	11	0x00000010
\$t4	12	0x00000001
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$t8	16	0x00000000
\$t9	17	0x00000000
\$s0	18	0x00000000
\$s1	19	0x00000000
\$s2	20	0x00000000
\$s3	21	0x00000000
\$s4	22	0x00000000
\$s5	23	0x00000000
\$s6	24	0x00000000
\$s7	25	0x00000000
\$s8	26	0x00000000
\$s9	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7ffffc00
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x0040003c
\$l		0x00000000
10		0x00000000

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000001	0x00000002	0x00000000	0x00000000	0x00000000	0x00000006	0x00000007
0x10010004	0x00000000	0x00000009	0x0000000a	0x00000000	0x00000000	0x00000001	0x0000000a	0x00000001
0x10010008	0x00000004	0x00000000	0x7420e645	0x72ef6e61	0x6c656420	0x66ef6320	0x66f46174	0x73652072
0x1001000c	0x0000001a	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010010	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010014	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010018	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001001c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010024	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010028	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001002c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010030	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010034	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010038	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001003c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

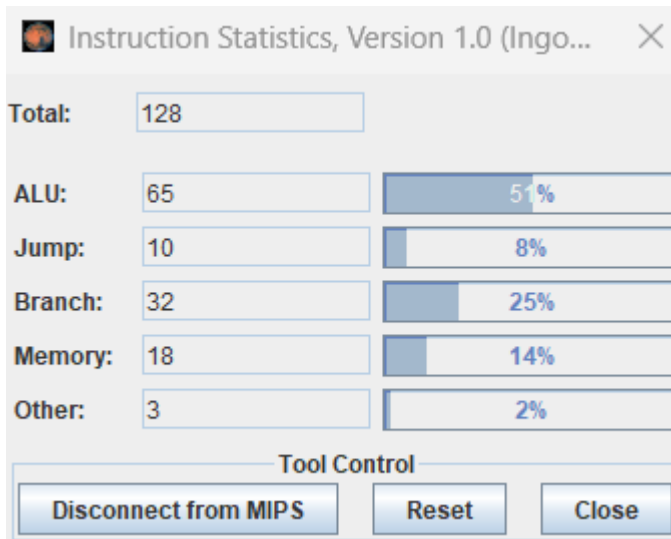
Mars Messages

Run IO

El valor del contador es: 6
-- program is finished running --

Reset: reset completed.

El valor del contador es: 6
-- program is finished running --



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

The screenshot displays a debugger window with the following components:

- Text Segment:** A list of assembly instructions with their addresses and comments. For example, at address 0x24020004, there is an instruction `addiu $2,$0,0x00000004` with comment "13: li \$v0, 4".
- Data Segment:** A table showing memory values at various addresses. The first row shows address 0x10010000 with value 0x72746e49.
- Registers:** A table on the right showing the state of registers. The `$zero` register has value 0, and `$t0` has value 0x10010000.
- Mars Messages / Run I/O:** A section at the bottom showing the interaction between the program and the user. It displays the prompt "Introduce una cadena de caracteres: open123" and the response "La cadena en mayúsculas es: OPEN123". It also shows "El número de caracteres es: 7" and "program is finished running --".

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

The screenshot shows the "Mars Messages / Run I/O" window with the following text:

```
-- program is finished running --

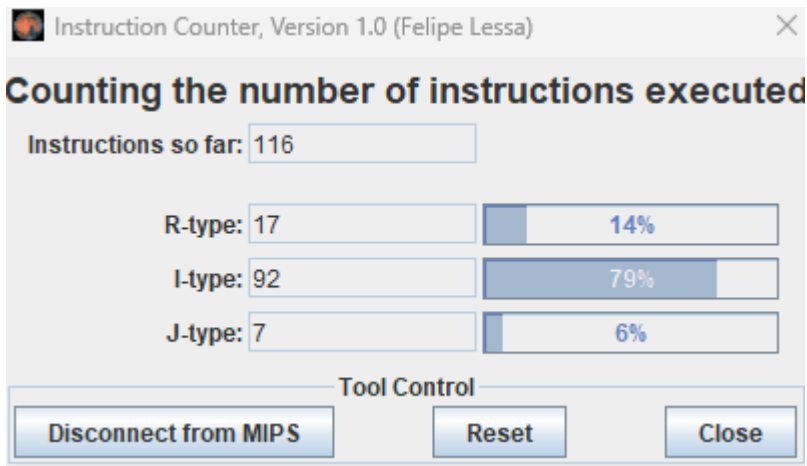
Introduce una cadena de caracteres: silla 8a
La cadena en mayúsculas es: SILLA 8A

El número de caracteres es: 8

-- program is finished running --
```

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: $w = x + y^2 + z \cdot 3$ 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* 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 $\{ x = -4.568 \quad y = 5.77 \quad z = 11.33 \}$

Resultado { 62.714897 }

Instrucciones{ 48 }

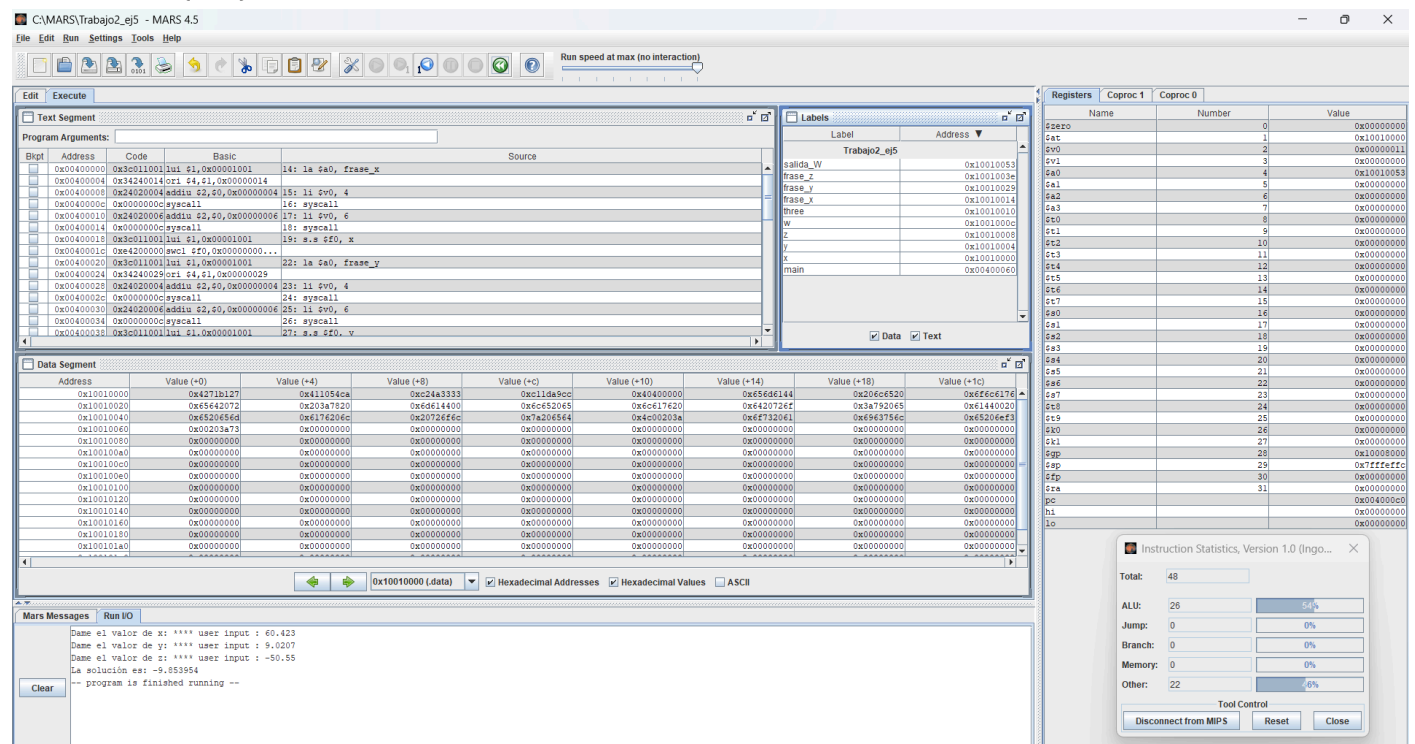
[illegible]

Registers	Coproc 1	Coproc 0
Name	Float	Double
%f0	0xc0922d0e	0x40b8a3d7c0922d0e
%f1	0x40b8a3d7	
%f2	0x427adc0e	0x413547ae427adc0e
%f3	0x413547ae	
%f4	0x4207f5c2	0x404000004207f5c2
%f5	0x40400000	
%f6	0x00000000	0x0000000000000000
%f7	0x00000000	
%f8	0x00000000	0x0000000000000000
%f9	0x00000000	
%f10	0x00000000	0x0000000000000000
%f11	0x00000000	
%f12	0x427adc0e	0x00000000427adc0e
%f13	0x00000000	
%f14	0x00000000	0x0000000000000000
%f15	0x00000000	
%f16	0x00000000	0x0000000000000000
%f17	0x00000000	
%f18	0x00000000	0x0000000000000000
%f19	0x00000000	
%f20	0x00000000	0x0000000000000000
%f21	0x00000000	
%f22	0x00000000	0x0000000000000000
%f23	0x00000000	
%f24	0x00000000	0x0000000000000000
%f25	0x00000000	
%f26	0x00000000	0x0000000000000000
%f27	0x00000000	
%f28	0x00000000	0x0000000000000000
%f29	0x00000000	
%f30	0x00000000	0x0000000000000000
%f31	0x00000000	

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

Resultado{ -9.853954 }

Instrucciones{ 48 }



Registers	Coproc 1	Coproc 0
Name	Float	Double
\$f0	0x4271b127	0x411054ca4271b127
\$f1	0x411054ca	
\$f2	0xc11da9cc	0xc24a3333c11da9cc
\$f3	0xc24a3333	
\$f4	0xc317a666	0x40400000c317a666
\$f5	0x40400000	
\$f6	0x00000000	0x0000000000000000
\$f7	0x00000000	
\$f8	0x00000000	0x0000000000000000
\$f9	0x00000000	
\$f10	0x00000000	0x0000000000000000
\$f11	0x00000000	
\$f12	0xc11da9cc	0x00000000c11da9cc
\$f13	0x00000000	
\$f14	0x00000000	0x0000000000000000
\$f15	0x00000000	
\$f16	0x00000000	0x0000000000000000
\$f17	0x00000000	
\$f18	0x00000000	0x0000000000000000
\$f19	0x00000000	
\$f20	0x00000000	0x0000000000000000
\$f21	0x00000000	
\$f22	0x00000000	0x0000000000000000
\$f23	0x00000000	
\$f24	0x00000000	0x0000000000000000
\$f25	0x00000000	
\$f26	0x00000000	0x0000000000000000
\$f27	0x00000000	
\$f28	0x00000000	0x0000000000000000
\$f29	0x00000000	
\$f30	0x00000000	0x0000000000000000
\$f31	0x00000000	

ejemplo 3:

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

Resultado{ 502.47998 }

Instrucciones{ 48 }

