EBook Gratis

APRENDIZAJE mips

Free unaffiliated eBook created from
Stack Overflow contributors.

#mips

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: mips

It is an unofficial and free mips ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official mips.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con mips

Observaciones

Esta sección proporciona una descripción general de qué es mips y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de Mips, y vincular a los temas relacionados. Dado que la Documentación para mips es nueva, es posible que deba crear versiones iniciales de los temas relacionados.

Examples

Instalación o configuración

Instrucciones detalladas sobre cómo configurar o instalar mips.

QtSpim para windows

1. descarga QtSpim desde aquí 32.6 MB
2. instálalo fácil instalación
3. haga su primer archivo de ensamblaje (.s) o use la muestra *C: \ Archivos de programa (x86) \ QtSpim \ helloworld.s*
4. ejecute el programa desde el acceso directo del escritorio o *C: \ Archivos de programa (x86) \ QtSpim \ QtSpim.exe*

Hay dos ventanas para el programa, la principal etiquetada como QtSpim. Aquí puede ver el programa que está ejecutando (texto etiquetado), la memoria (datos etiquetados), los valores de los registros (etiquetados Regulaciones FP para punto flotante e Int Regs para enteros) y el control del simulador.

la otra ventana etiquetada como consola es donde verá la salida e ingresará la entrada de su programa si hay alguna

5. carga el archivo usando Archivo -> Cargar archivo
6. puede usar click run (f5) para ver el resultado final o ir paso a paso (p10) para ver el estado del registro y la memoria mientras el programa se ejecuta para depurar

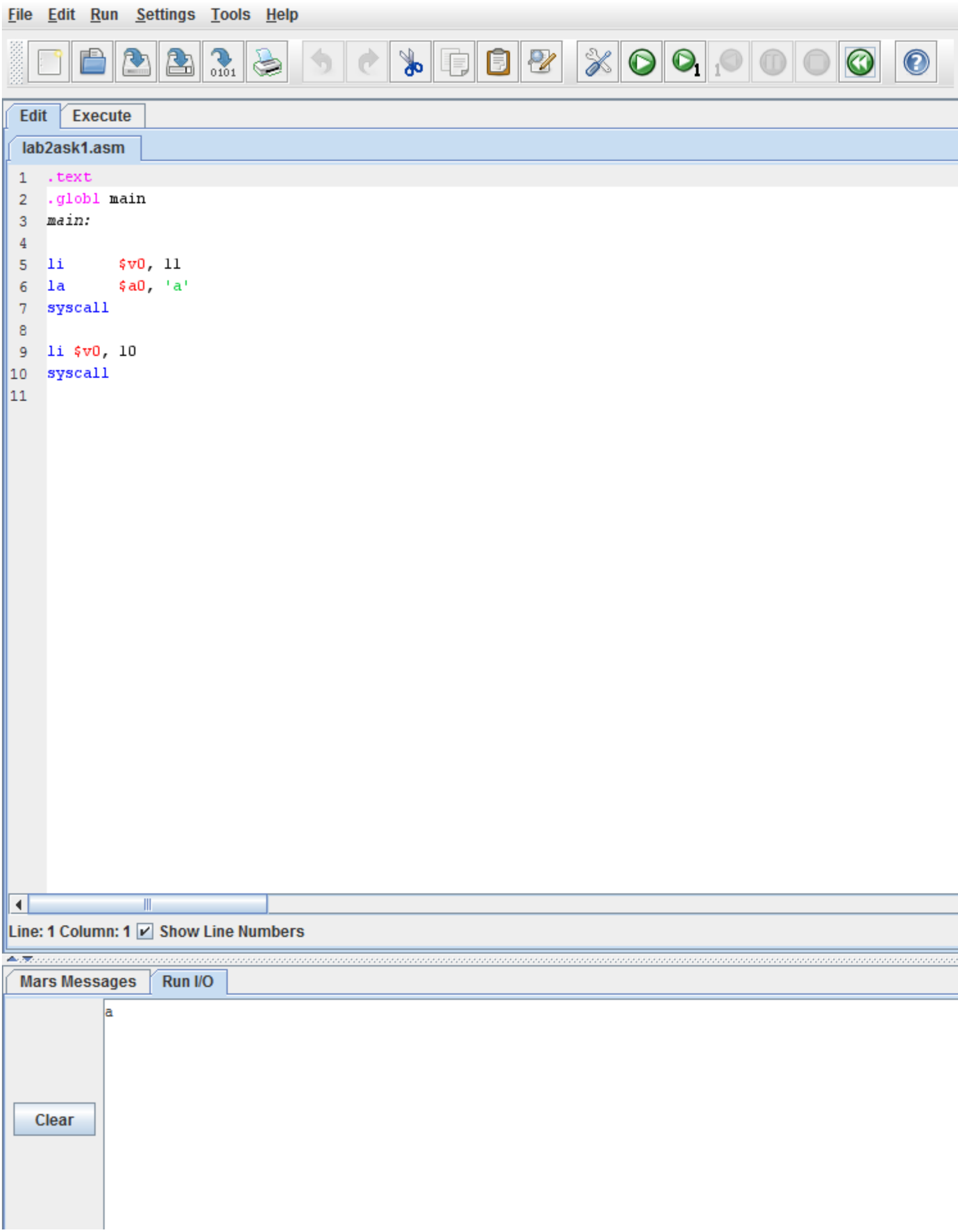
Simulador MARS MIPS

MARS MIPS simulator es un editor, ensamblador, simulador y depurador de lenguaje ensamblador para el procesador MIPS, desarrollado por Pete Sanderson y Kenneth Vollmar en la Universidad Estatal de Missouri (src).

Obtienes el MARS gratis aquí . En cuanto a la instalación de la versión 4.5, es posible que

necesite el SDK de Java adecuado para su sistema desde aquí.

Antes de ensamblar, el entorno de este simulador se puede dividir de manera simplista en tres segmentos: el *editor* en la parte superior izquierda donde se escribe todo el código, el compilador / salida justo debajo del editor y la *lista de registros* que representan la "CPU" para nuestro programa.



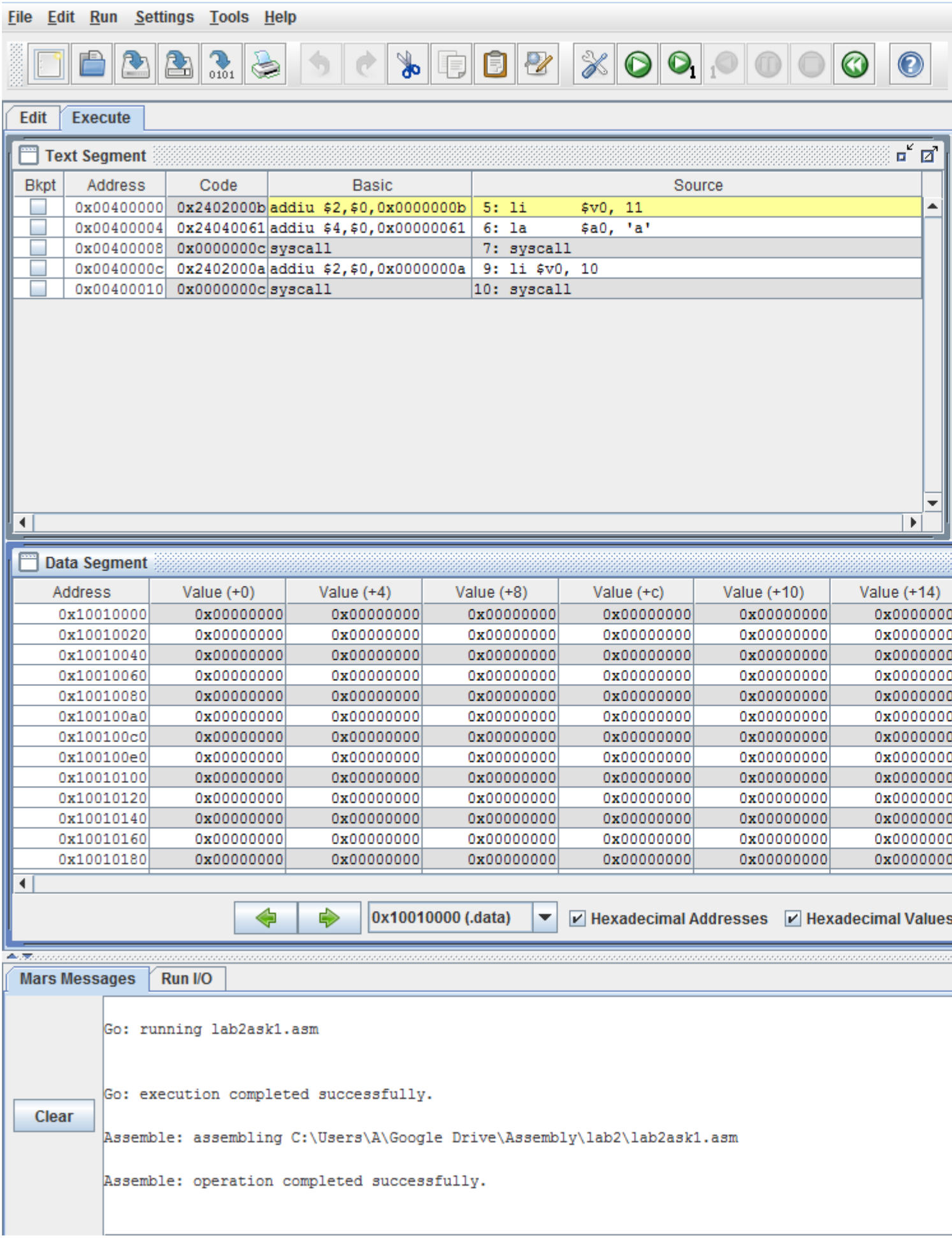
Después de ensamblar (simplemente presionando F3), el entorno cambia, con dos nuevos

segmentos obteniendo la posición del editor: el *segmento de texto* donde

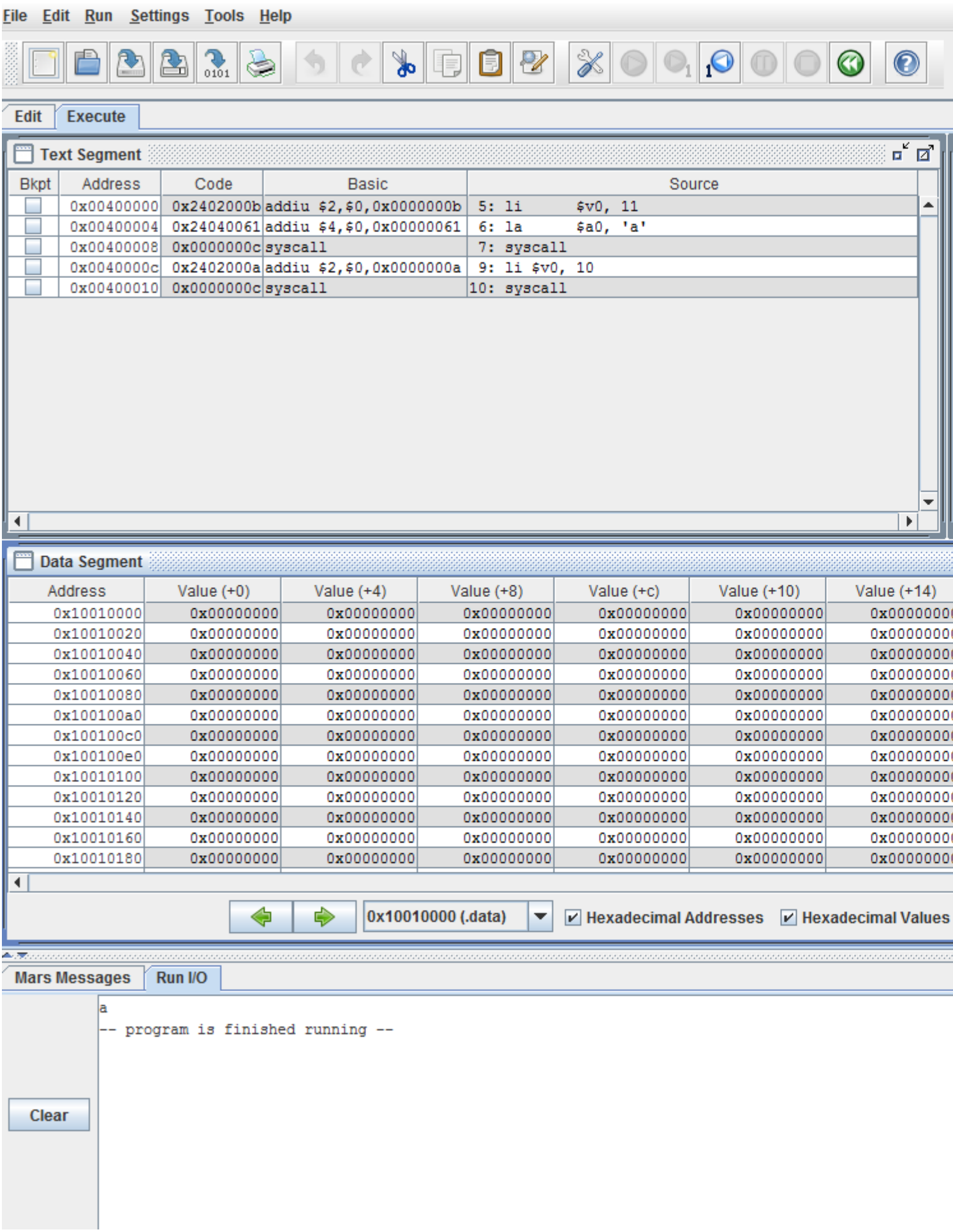
i) cada línea de código de ensamblaje se borra de "pseudoinstrucciones" (hablaremos de ellas en un segundo) en la columna "básica" y

ii) el código de máquina para cada instrucción en la columna "código",

y el *segmento de datos* donde podemos ver una representación de la memoria de un procesador con orden little-endian .


File Edit Run Settings Tools Help
Edit Execute
Text Segment
Bkpt Address Code Basic Source
0x00400000 0x2402000b addiu $2,$0,0x0000000b 5: li $v0, 11
0x00400004 0x24040061 addiu $4,$0,0x00000061 6: la $a0, 'a'
0x00400008 0x0000000c syscall 7: syscall
0x0040000c 0x2402000a addiu $2,$0,0x0000000a 9: li $v0, 10
0x00400010 0x0000000c syscall 10: syscall
Data Segment
Address Value (+0) Value (+4) Value (+8) Value (+c) Value (+10) Value (+14)
0x10010000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x0000000
0x10010020 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x0000000
0x10010040 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x0000000
0x10010060 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x0000000
0x10010080 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x0000000
0x100100a0 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x0000000
0x100100c0 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x0000000
0x100100e0 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x0000000
0x10010100 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x0000000
0x10010120 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x0000000
0x10010140 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x0000000
0x10010160 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x0000000
0x10010180 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x0000000
0x10010000 (.data)
Hexadecimal Addresses
Hexadecimal Values
Mars Messages
Run I/O
Clear
Go: running lab2ask1.asm
Go: execution completed successfully.
Assemble: assembling C:\Users\A\Google Drive\Assembly\lab2\lab2ask1.asm
Assemble: operation completed successfully.

Después del ensamblaje, podemos ejecutar nuestro código de una vez (F5) o paso a paso (F7), así como rebobinar la ejecución varios pasos hacia atrás (F8).


File Edit Run Settings Tools Help
Edit Execute
Text Segment
Bkpt Address Code Basic Source
0x00400000 0x2402000b addiu $2,$0,0x0000000b 5: li $v0, 11
0x00400004 0x24040061 addiu $4,$0,0x00000061 6: la $a0, 'a'
0x00400008 0x0000000c syscall 7: syscall
0x0040000c 0x2402000a addiu $2,$0,0x0000000a 9: li $v0, 10
0x00400010 0x0000000c syscall 10: syscall
Data Segment
Address Value (+0) Value (+4) Value (+8) Value (+c) Value (+10) Value (+14)
0x10010000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010020 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010040 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010060 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010080 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x100100a0 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x100100c0 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x100100e0 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010100 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010120 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010140 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010160 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010180 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010000 (.data)
Hexadecimal Addresses
Hexadecimal Values
Mars Messages Run I/O
a
-- program is finished running --
Clear

Ahora, veamos el código de ejemplo de arriba y expliquemos cada línea:

```
.text
.globl main
main:            #main function

li    $v0, 11    #11=system code for printing a character, $v0=register that gets the system
code for printing as value
la    $a0, 'a'   #'a'=our example character, $a0=register that accepts the character for
printing
syscall          #Call to the System to execute our instructions and print the character at
the a0 register

li $v0, 10       #11=system code for terminating, $v0=register that gets the system code for
terminating (optional, but desirable)
syscall          #Call to the System to terminate the execution
```

MARS acepta y exporta archivos con el tipo de archivo .asm

Pero el código de arriba solo imprime un carácter, ¿qué pasa con el buen viejo "Hola mundo"? ¿Qué hay, dunno, añadiendo un número o algo? Bueno, podemos cambiar lo que teníamos un poco por eso:

```
.data               #data section
str: .asciiz "Hello world\n"
number: .word 256

.text                 #code section
.globl main
main:
li       $v0, 4               #system call for printing strings
la       $a0, str             #loading our string from data section to the $a0 register
syscall

la       $t0, number        #loading our number from data section to the $t0 register
lw       $s1, 0($t0)        #loading our number as a word to another register, $s1

addi     $t2, $s1, 8         #adding our number ($s1) with 8 and leaving the sum to register
$t2

sw       $t2, 0($t0)        #storing the sum of register $t2 as a word at the first place of
$t0

li       $v0, 10              # system call for terminating the execution
syscall
```

Antes de ilustrar los resultados a través de MARS, se necesita un poco más de explicación sobre estos comandos:

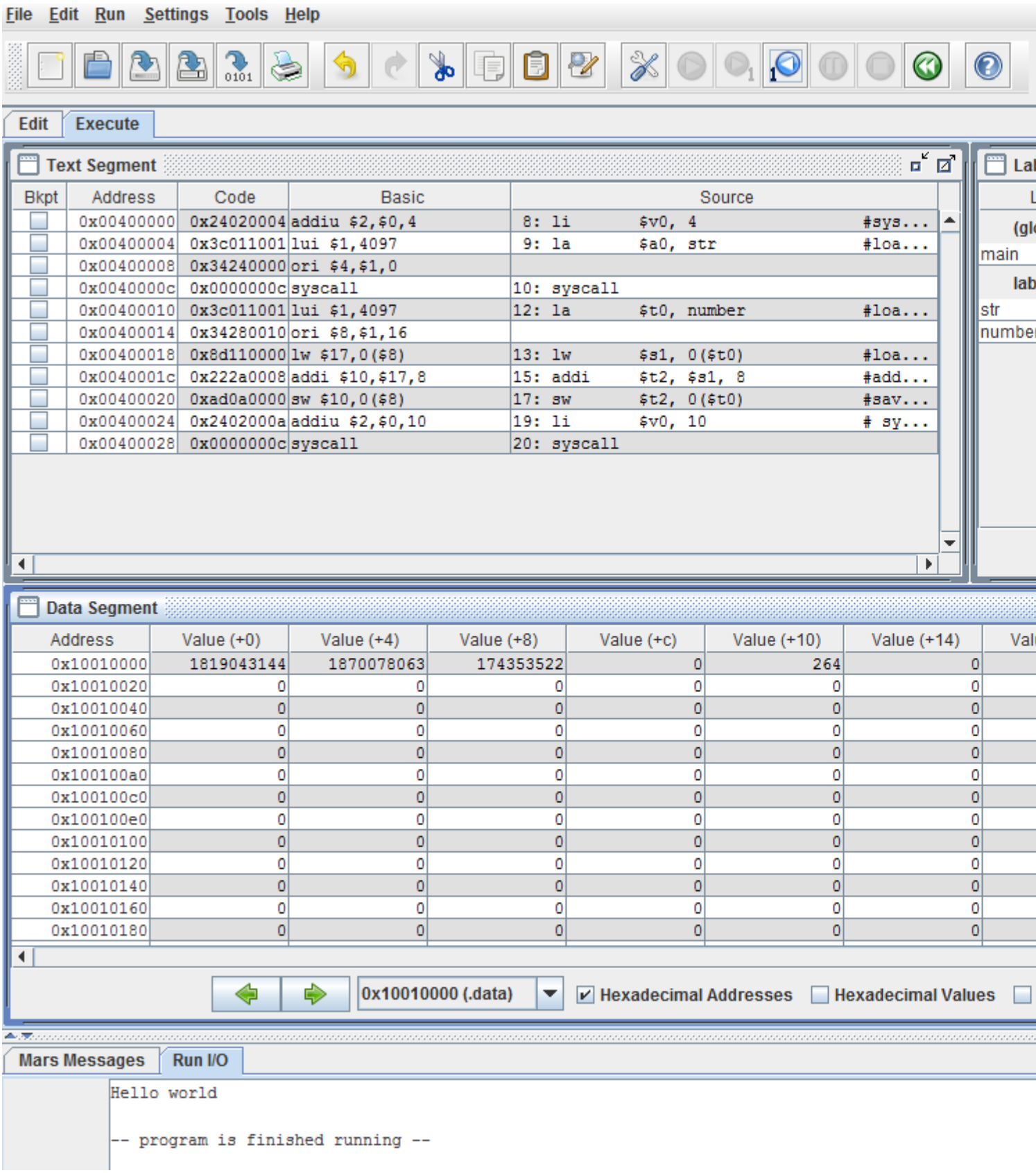
- ***Las llamadas al sistema*** son un conjunto de servicios provistos por el sistema operativo. Para usar una llamada al sistema, se necesita un *código de llamada* para ingresar a $ v0 para la operación necesaria. Si una llamada al sistema tiene argumentos, esos se colocan en los registros $ a0- $ a2. Aquí están todas las llamadas al sistema.

- `li` (carga inmediata) es una pseudo-instrucción (hablaremos de eso más adelante) que

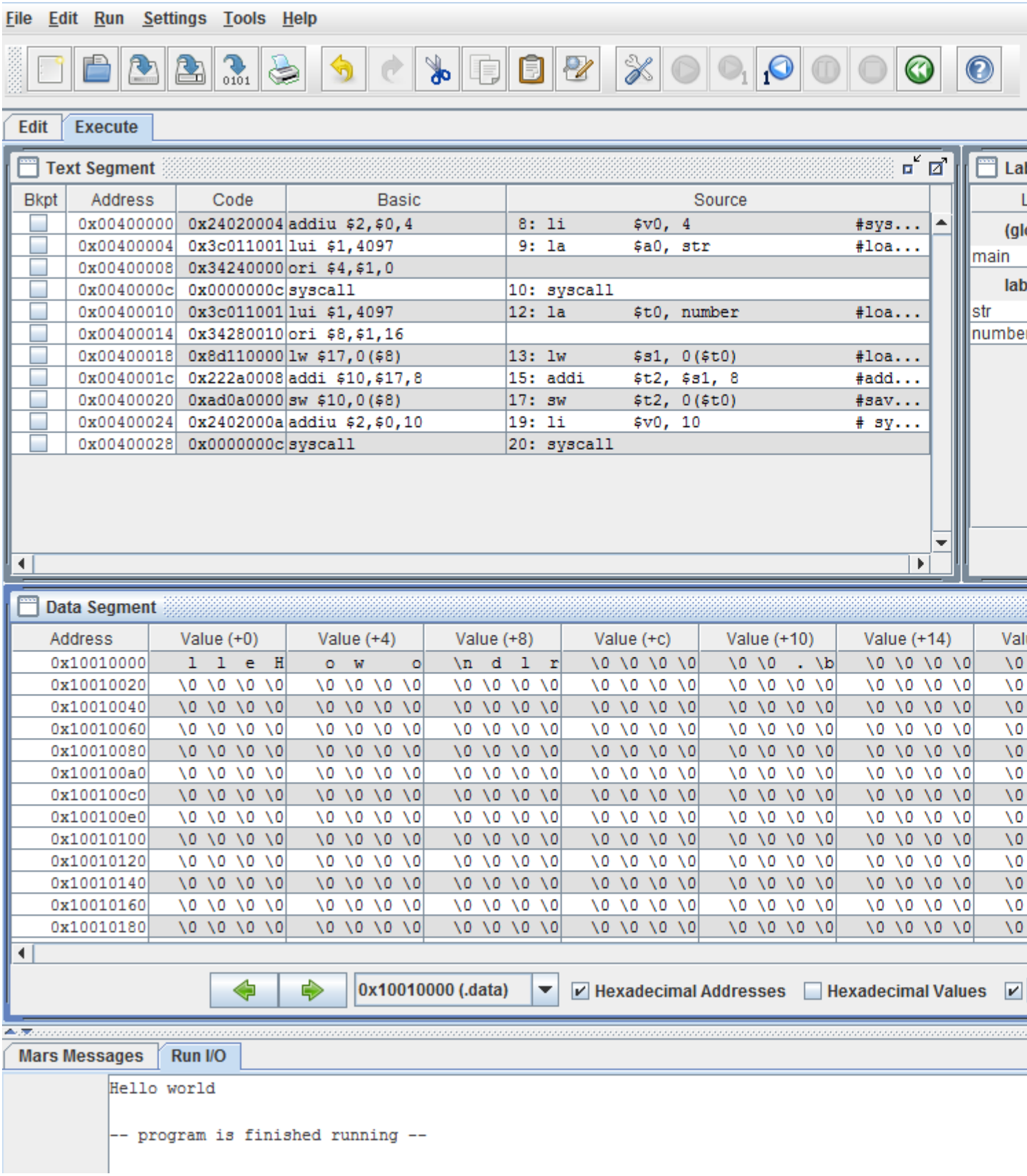
carga instantáneamente un registro con un valor. `la` (dirección de carga) también es una pseudo-instrucción que carga una dirección en un registro. Con `li $v0, 4` el registro $ v0 tiene ahora `4` como valor, mientras que `la $a0, str` carga la cadena de `str` en el registro `$a0` .

- Una ***palabra*** es (tanto como estamos hablando de MIPS) una secuencia de 32 bits, siendo el bit 31 el Bit más significativo y el bit 0 el Bit menos significativo.
- `lw` (palabra de carga) se transfiere de la memoria a un registro, mientras que `sw` (palabra de almacenamiento) se transfiere de un registro a la memoria. Con el comando `lw $s1, 0($t0)` , cargamos a `$s1` registrando el valor que estaba en el LSB del registro `$t0` (eso es lo que el `0` simboliza aquí, el desplazamiento de la palabra), también conocido como `256` . `$t0` aquí tiene la dirección, mientras que `$s1` tiene el valor. `sw $t2, 0($t0)` hace el trabajo opuesto.
- MARS usa el Little Endian , lo que significa que el LSB de una palabra se almacena en la dirección de byte más pequeña de la memoria.
- MIPS usa **direcciones de bytes** , por lo que una dirección es aparte de su anterior y siguiente en 4.

Al ensamblar el código de antes, podemos comprender mejor cómo se intercambian la memoria y los registros, deshabilitando "Valores hexadecimales" del segmento de datos:



o habilitar "ASCII" desde el segmento de datos:



Comience así

```
$ java -jar Mars4_5.jar
```

Crea este archivo y guárdalo.

```
    .text
main:
    li    $s0,0x30
loop:
    move    $a0,$s0         # copy from s0 to a0

    li    $v0,11         # syscall with v0 = 11 will print out
    syscall             # one byte from a0 to the Run I/O window

    addi    $s0,$s0,3    # what happens if the constant is changed?

    li    $t0,0x5d
    bne    $s0,$t0,loop
    nop             # delay slot filler (just in case)

stop:    j    stop         # loop forever here
    nop             # delay slot filler (just in case)
```

Presione F3 para ensamblarlo y luego presione Ejecutar. Ahora estás empezando a compilar y ejecutar código MIPS.

Lea Empezando con mips en línea: https://riptutorial.com/es/mips/topic/7049/empezando-con-mips

Creditos

S. No	Capítulos	Contributors
1	Empezando con mips	Community, Coursal, Dac Saunders, robert