

Lab 1: Introduction to MARS Simulator

Objectives :

After completing this lab, you will:

- Get familiar with the MARS simulator
- Learn how to assemble, run, and debug a MIPS program

MARS may be downloaded from:

<https://courses.missouristate.edu/KenVollmar/mars/download.htm>

1. General Information

MARS, the **MIPS** Assembly and Runtime Simulator, is an integrated development environment (IDE) for programming in MIPS assembly language. It allows editing, assembling, debugging and simulating the execution of MIPS assembly language programs. MARS is written in Java.

There are two main windows in MARS, as shown in Figure 1.1.

- The *Edit* window: used to create and modify a MIPS program.
- The *Execute* window: used to run and debug a MIPS program.

To switch between the *Edit* and the *Execute* windows, use the tabs at the top. The *Execute* window contains three main panes:

1. *Text Segment*: shows the machine code and related addresses.
2. *Data Segment*: shows memory locations that hold variables in the data segment.
3. *Labels*: shows addresses of labeled items, i.e. variables and jump endpoints.

There are two tabbed message areas at the bottom of Figure 1.1:

1. The *Mars Messages* tab: Used for messages such as assembly or runtime errors and informational messages. You can click on assembly error messages to select the corresponding line of code in the editor.
2. The *Run I/O* tab: Used at runtime for displaying console output and entering console input as program execution progresses.

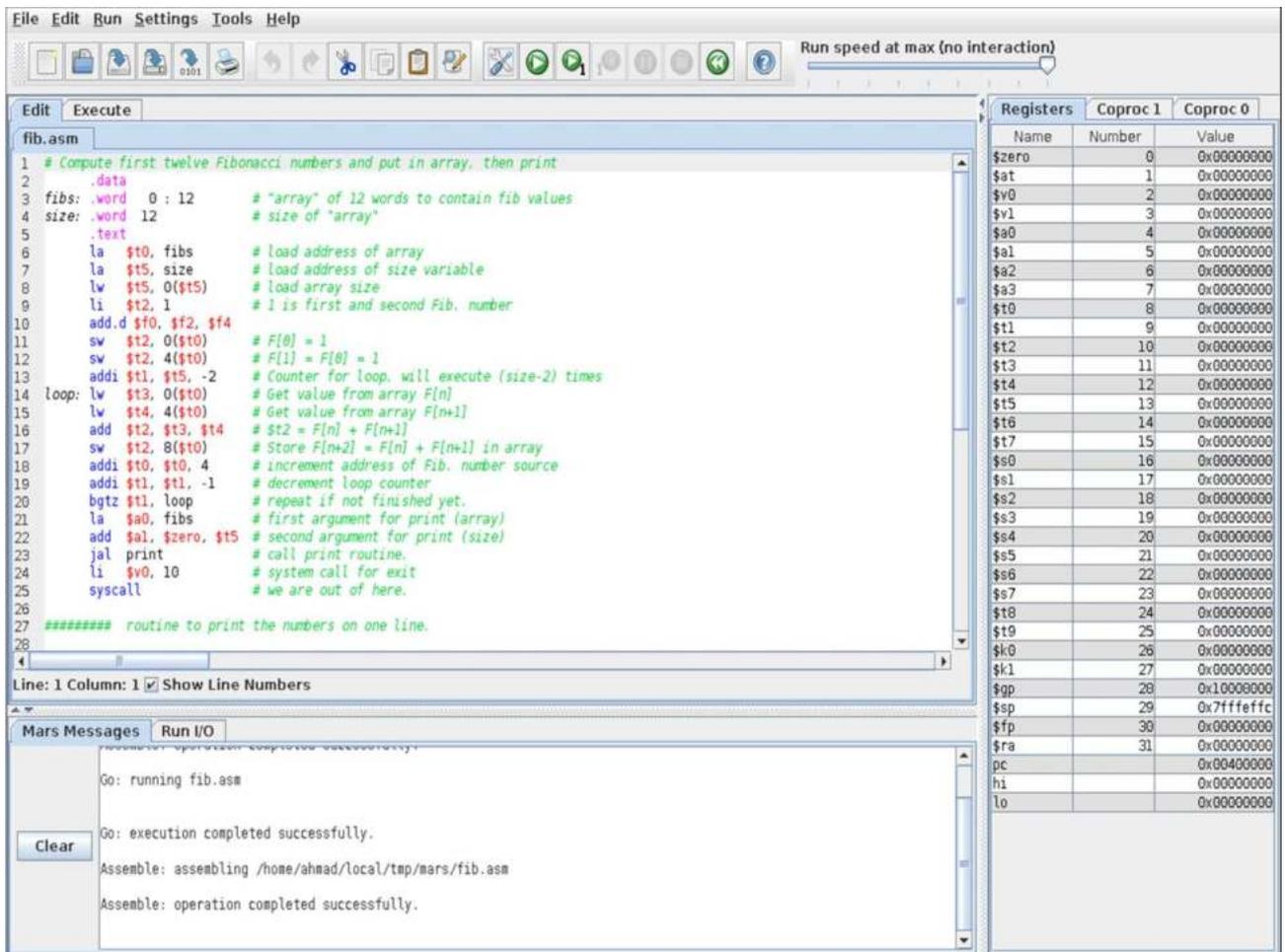


Figure 1.1: The MARS Integrated Development Environment (IDE)

Figure 1.2 shows the MARS Execute window's panes, and emphasizes the following features:

1. The Execute window's tab.
2. Assembly code displayed with addresses and machine code.
3. Values stored in the data segment. These are directly editable.
4. Controls for navigating the data memory area. Allows switching to view the stack segment.
5. Switching between decimal and hexadecimal addresses and values in memory and registers.
6. Labels and their corresponding memory addresses.

7. Values stored in registers. These are directly editable.
8. Checkboxes used to setup breakpoints for each MIPS instruction. Useful in debugging.
9. Execution speed selection. Useful in debugging.

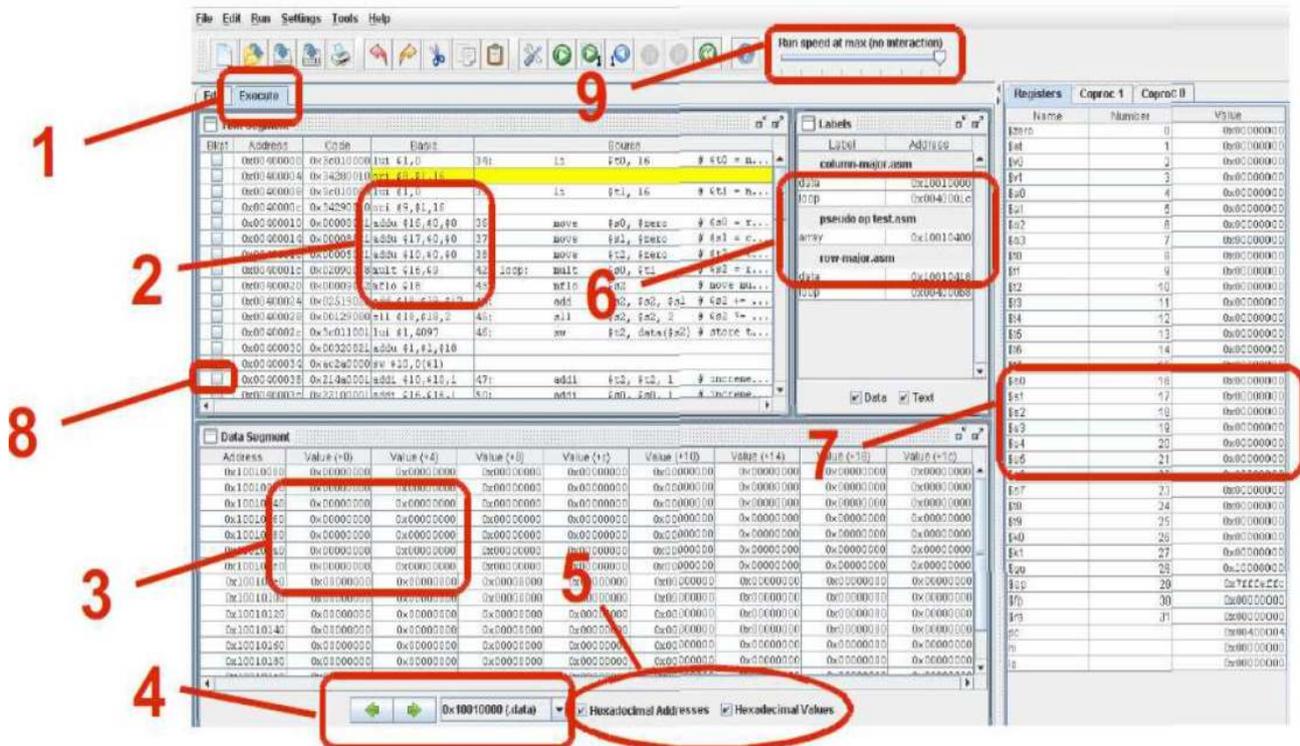


Figure 1.2: The MARS Execute Window

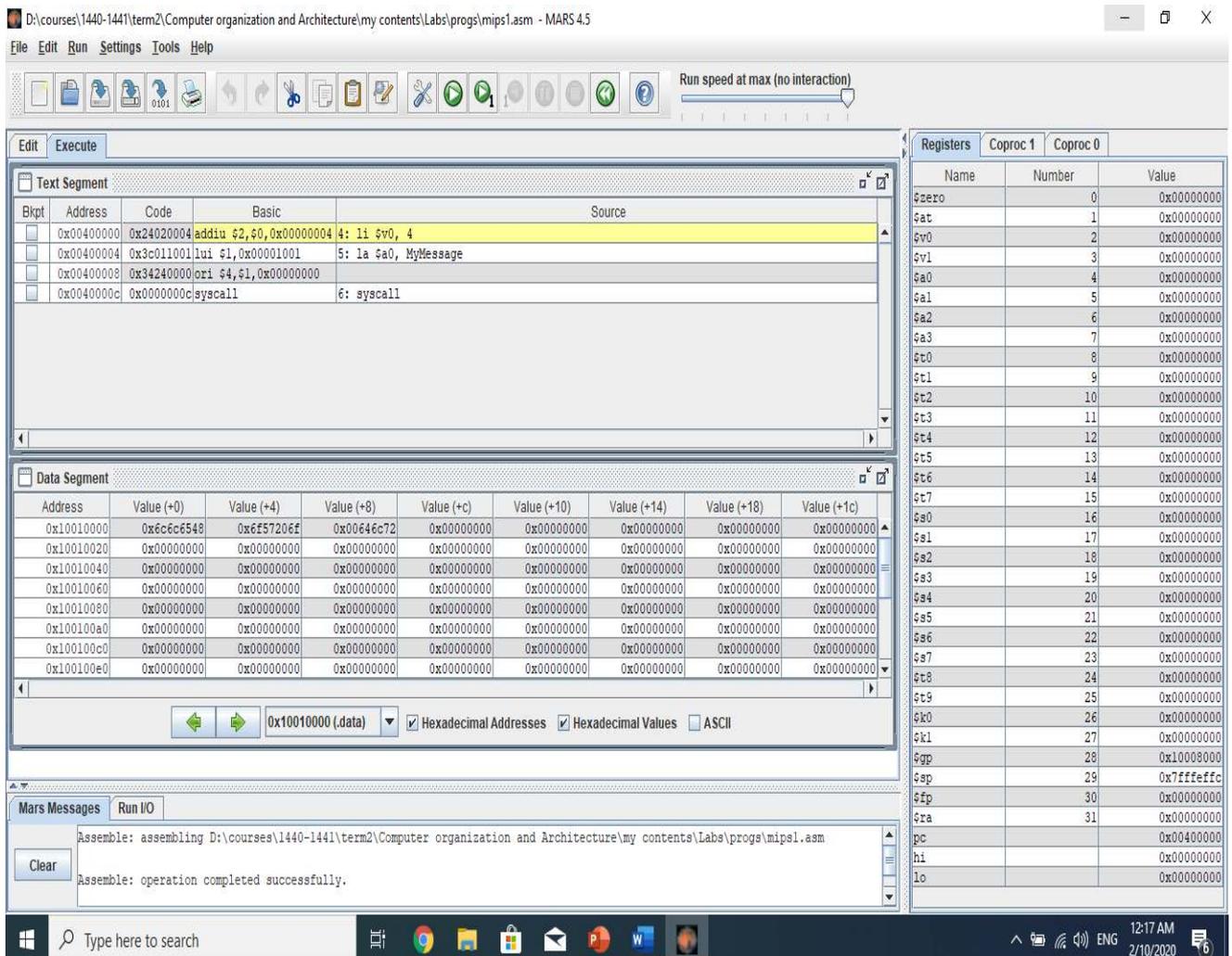


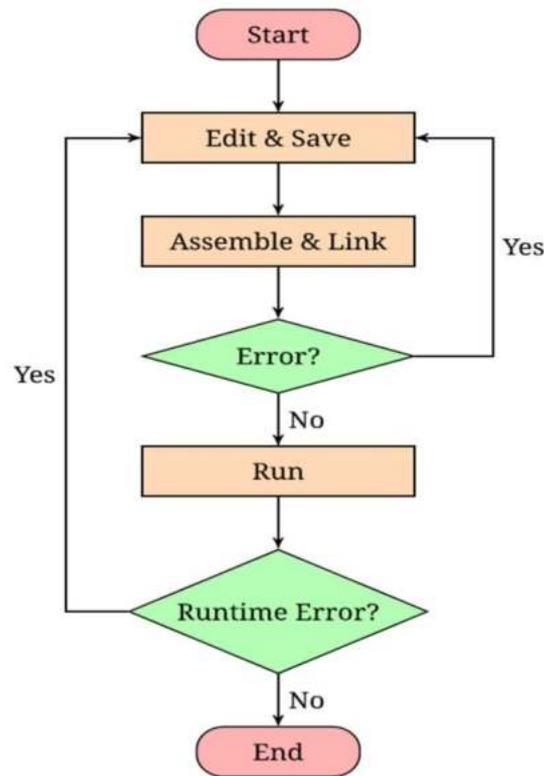
Figure 1.3: MARS screen after running the Assemble command

At all times, the MIPS register window appears on the right-hand side of the screen, even when you are editing and not running a program. While writing a program, this serves as a useful reference for register names and their use. Move the mouse over the register name to see the tool tips.

There are three register tabs:

- The Register File: integer registers **\$0** through **\$31**, **HI**, **LO**, and the Program Counter **PC**. Coprocessor 0: exceptions, interrupts, and status codes.
- Coprocessor 1: floating point registers.

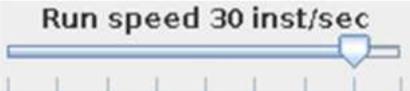
2. Edit, Save, Assemble, Run, and Debug a MIPS Program



- To edit a new program, select New from the File menu, or use the New toolbar icon.
- To save the program currently in the Edit tab, select Save as from the File menu, or use the Save as toolbar icon and write the name of the file with the extension **.asm**.
- To assemble the file currently in the Edit tab, select Assemble from the Run menu, or use the Assemble toolbar icon.
- If there are syntax errors in the program, they will appear in the Mars Messages tab at the bottom of the MARS screen. Each error message contains the line and column where the error occurred.

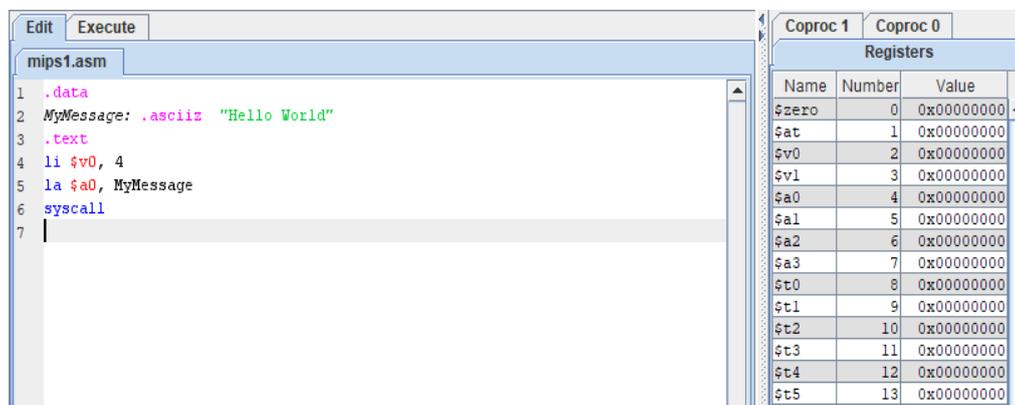
After running the Assemble command, you can now execute the program. The Run menu and the toolbar contain the following execution options:

Menu Item	Icon	Action
<i>Run > Assemble</i>		Assemble the program.

<i>Run > Go</i>		Run the program to completion, or until the next breakpoint.
<i>Run > Reset</i>		Reset the program and simulator to initial values. Allows restarting program execution.
<i>Run > Step</i>		Single-step execution: execute one instruction at a time. Allows debugging the program by inspecting register and memory after executing each single instruction.
<i>Run > Backstep</i>		Single-step backwards: “unexecute” the last executed instruction.
		The Run Speed Slider allows running the program at full speed or slowing it down so you can watch the execution. Affects normal execution only, not single-step execution.

3. Apply the precedent steps to run a program

- a. Type the program shown in the following figure.



```

1  .data
2  MyMessage: .asciiz "Hello World"
3  .text
4  li $v0, 4
5  la $a0, MyMessage
6  syscall
7

```

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000

- b. Find out how to show and hide line numbers.
- c. Assemble and run the program.
- d. What output does the program produce? and where does it appear?