1 Introduction to MARS

1.1 Objectives

After completing this lab, you will:

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

1.2 The MARS Simulator

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.

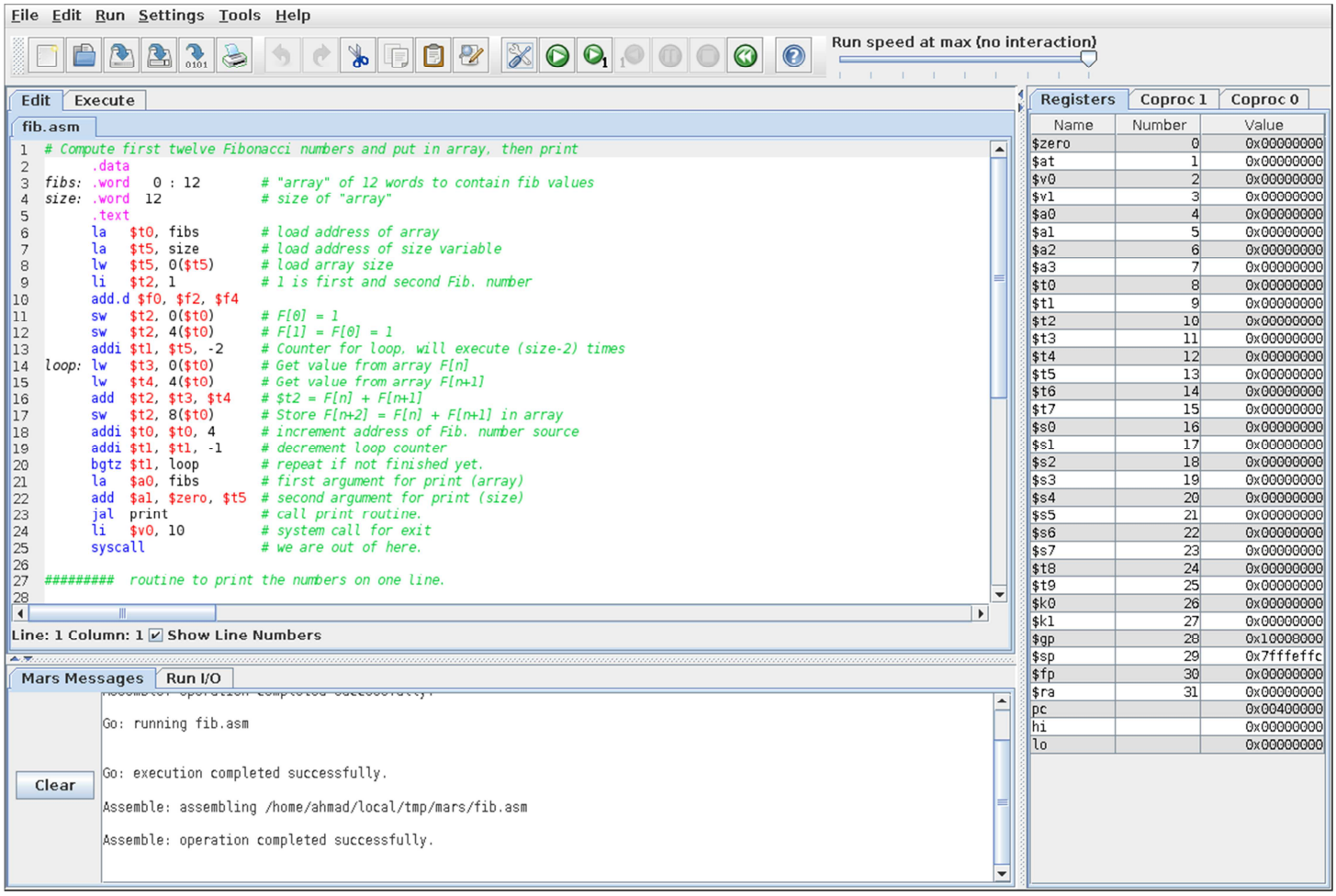


Figure 1.1: The MARS Integrated Development Environment (IDE)

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.

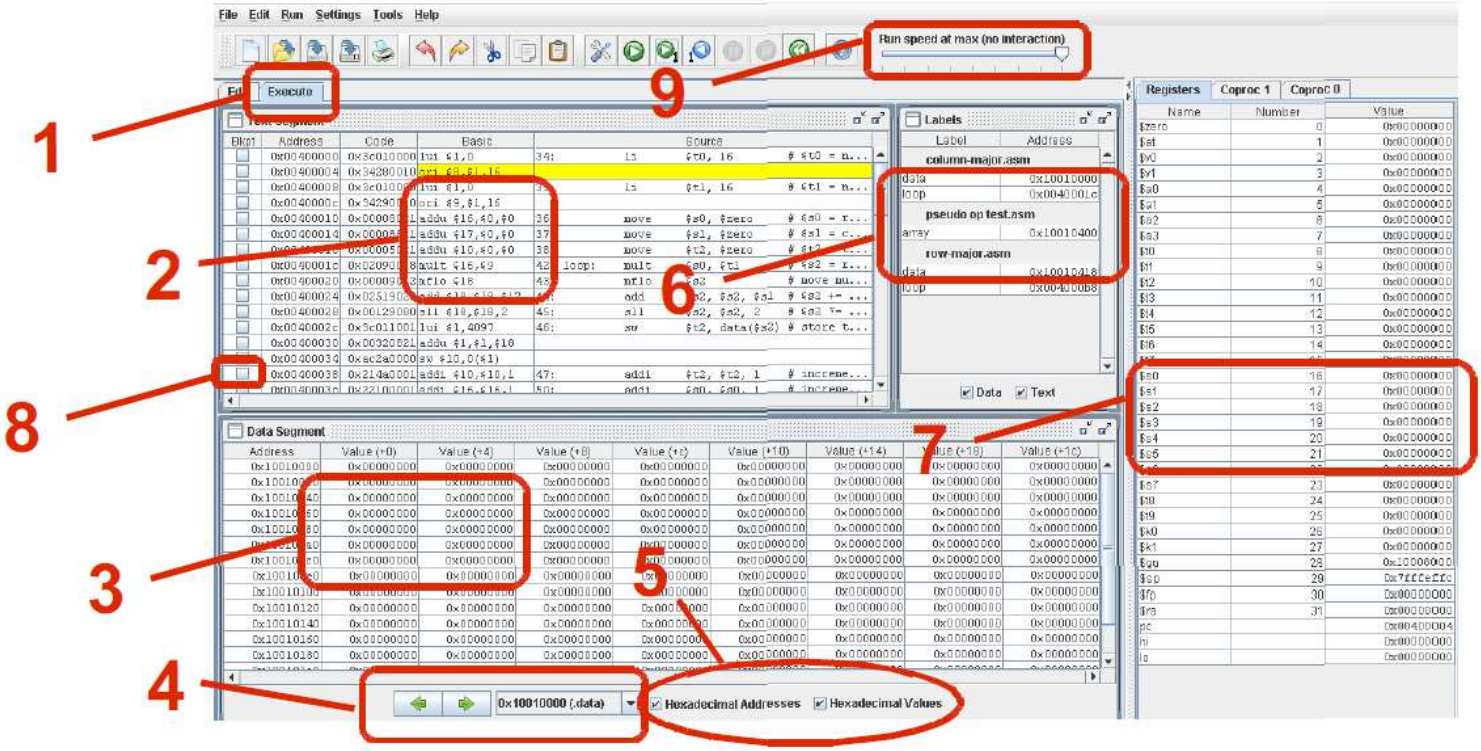


Figure 1.2: The MARS Execute Window

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.

1.3 Assemble, Run, and Debug a MIPS Program

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.

Once a MIPS program assembles successfully, the registers are initialized, and the Text Segment and the Data Segment are filled, as shown in Figure 1.3.

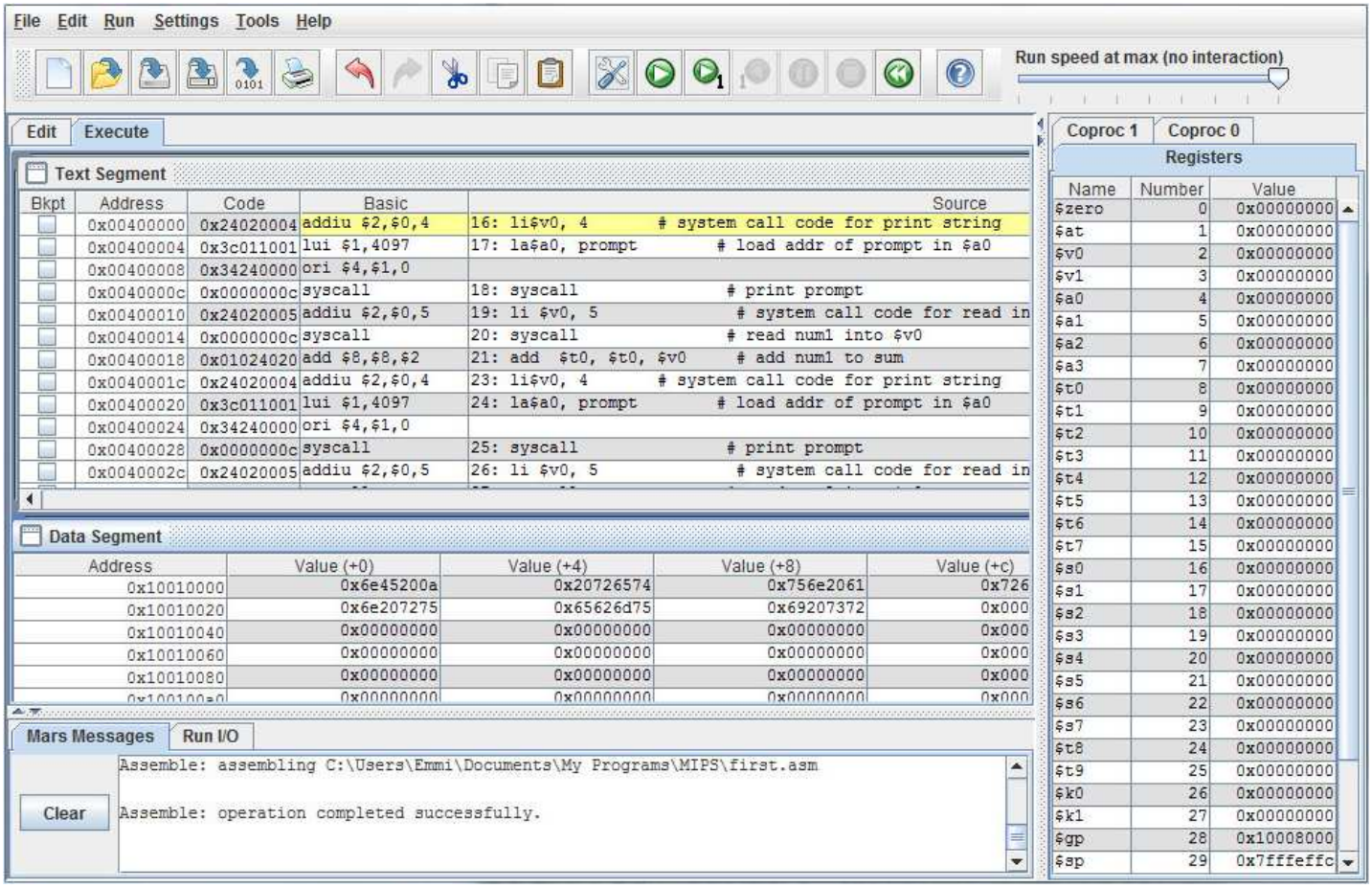
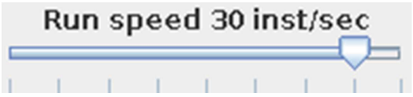


Figure 1.3: MARS screen after running the Assemble command

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
Run > Assemble		Assemble the program.
Run > Go		Run the program to completion, or until the next breakpoint.
Run > Reset		Reset the program and simulator to initial values. Allows restarting program execution.
Run > Step		Single-step execution: execute one instruction at a time. Allows debugging the program by inspecting register and memory after executing each single instruction.
Run > Backstep		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.

You can set a breakpoint at any instruction by checking the checkbox in front of the instruction in the text segment pane.

During execution, the instruction being executed is highlighted in yellow, and the register that was last modified is highlighted in green. Also, the variable that was last updated in the data segment is highlighted in blue. It's usually only possible to see the highlighting when you are stepping or running at less than full speed.

For more details about the MARS simulator, refer to the MARS documentation at the following link: http://courses.missouristate.edu/KenVollmar/MARS/

1.4 In-Lab Tasks

1. Test a simple MIPS program. Consider the following program shown below:

   a) Type the program shown in the Figure above.

   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?

2. Explore the MARS simulator:

   a) Download and assemble the **`Fibonacci.asm`** program from the MARS website.

   b) Identify the locations and values of the initialized data.

   c) Toggle the display format between decimal and hexadecimal.

   d) Run the program at a speed of 3 instructions per second or less.

   e) Single-step through the program and watch how register and memory values change.

f) Observe the output of the program in the *Run I/O* display window.

g) Set a breakpoint at the first instruction that prints results. What is the address of this instruction?

h) Run the program at full speed and watch how it stops at the breakpoint.

i) Change the line:

```
space: .asciiz " "    # space to insert between numbers
```

to:

```
space: .asciiz "\n"   # space to insert between numbers
```

Run the program again. What do you notice?