

**Lab 1: Introduction to MARS MIPS**

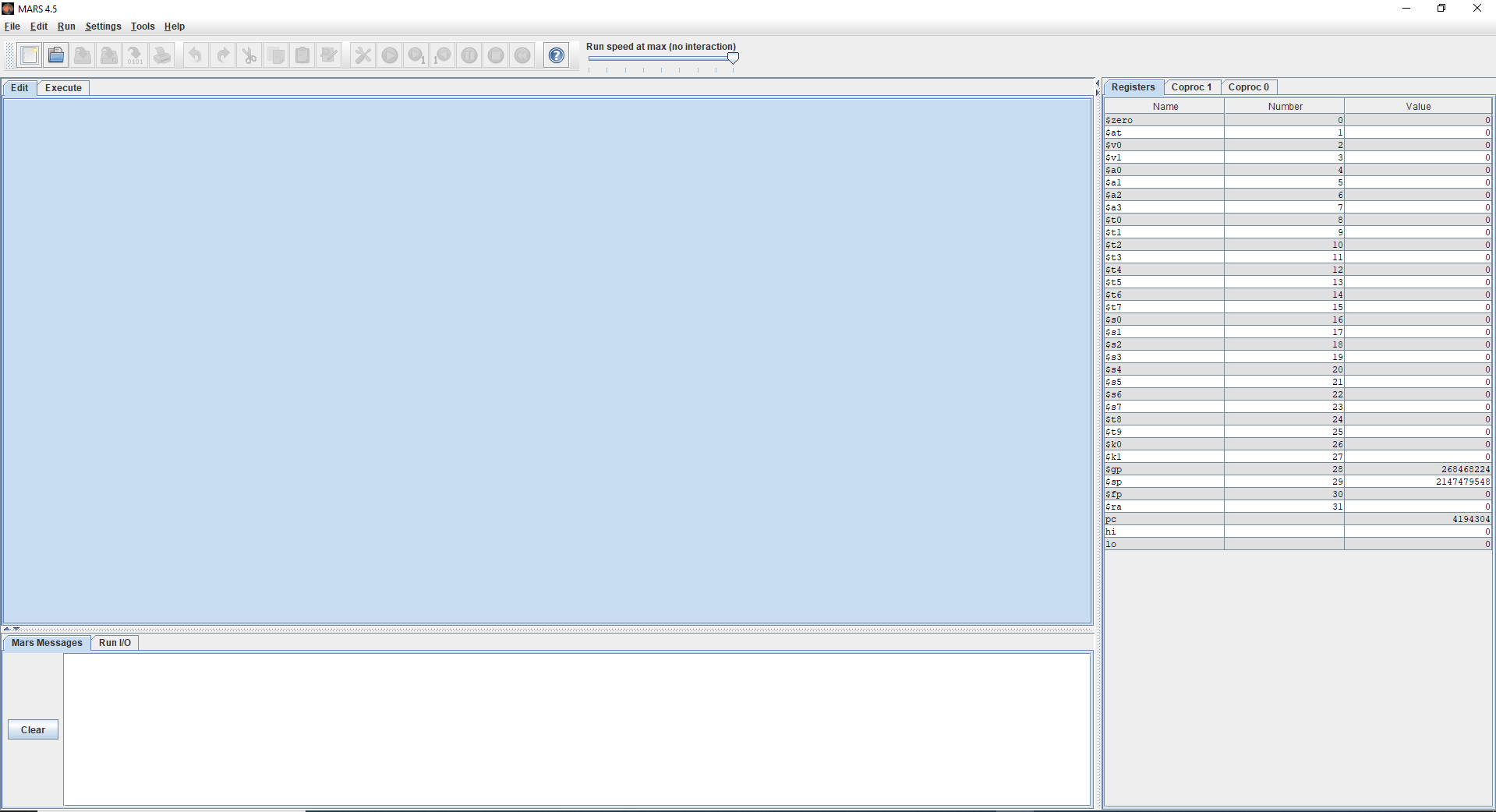
# Objectives

This lab introduces the student to Mars MIPS and the MIPS assembly language.

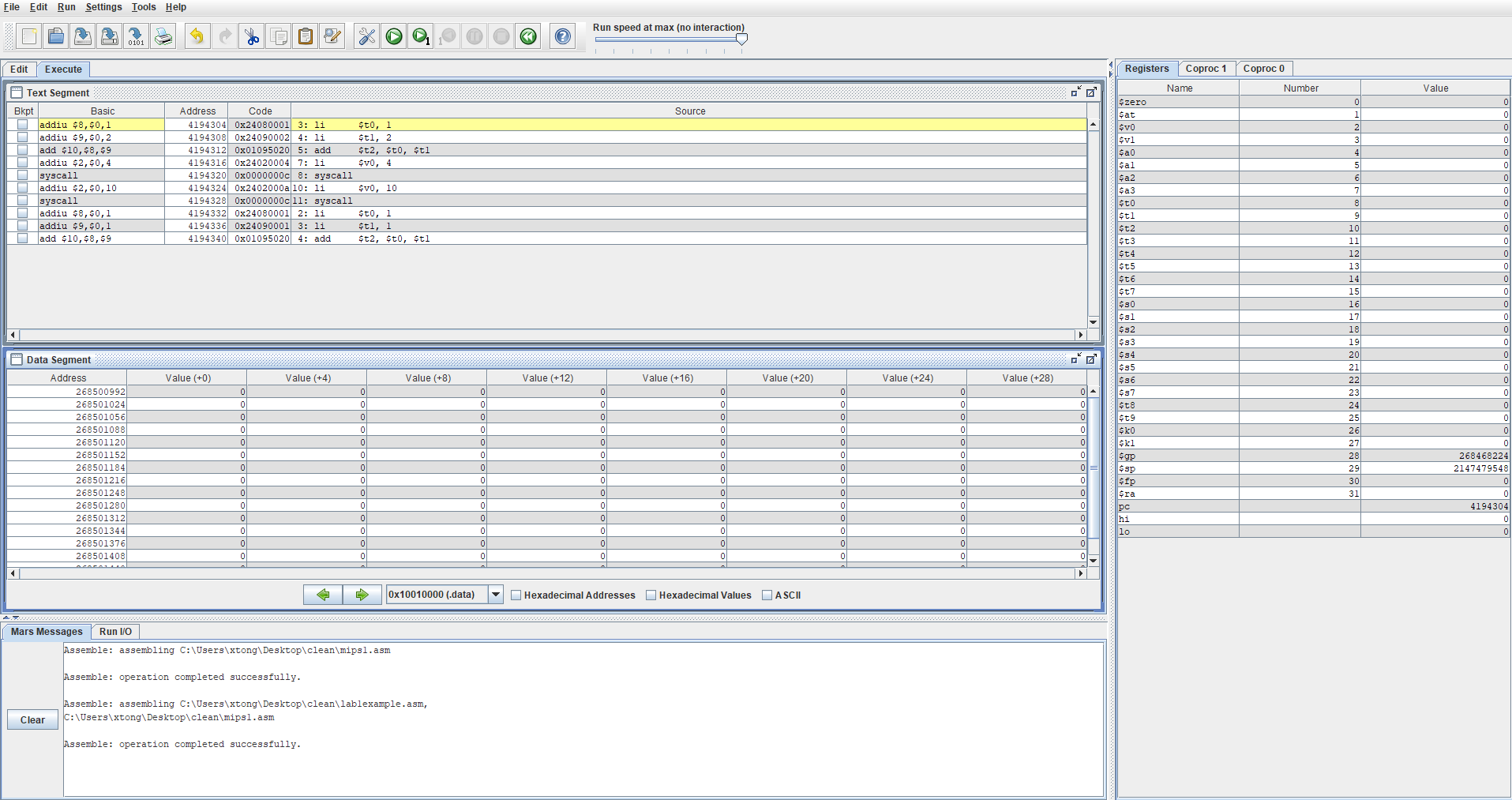
# Discussion and deliverable

Mars MIPS is an integrated development environment (IDE) for MIPS. It is a simulator designed to introduce students to MIPS assembly language and what is happening under the hood when the code is executed. Although Mars simulator is designed for educational purposes, MIPS chipset is used in many different fields in the real world.

Unzip lab 1 and move the .jar file and the .asm file into one folder. To run mars mips, run the .jar file (you may need to update your java runtime environment in order to run Mars MIPS). Figure 1 shows the starting window.

  
*Figure 1*

Open the .asm file that was placed in the same folder as the .jar file. Assemble the file by hitting the  icon or F3 or through the Run pull down menu. After assembling, the main window will be automatically toggled to the executed window (as shown in figure 3).

  
*Figure 3*

The text segment window shows each line of instruction code along with the memory address associated with the instruction (Assembly is interpreted line by line, with each line as an instruction). You can add break points using the 1st column in the text segment window.

The data segment window shows the contents of data segment of memory. In this window, you can also toggle through the different segments of memory using the drop down menu. In this window, you can also toggle between how the data is represented (hex, dec, or ascii).

The vertical rectangular window on the right with three tabs is the register status window. Under the registers tab is a table of 32 registers. In this table, the 1st and 2nd columns are the name and number of the registers. In your code, you can refer to them using either one. For example, if you wanted to load a number into a register $t0, you can either use “li $t0, number” or “li $8, number” (note: here, “number” stands for some integer. When using LI instruction, the second operand must be some integer). The 3rd column is the contents of the particular register. This value may change during runtime, as each instruction is executed.

  
*Figure 4*

Hit the step play is a step button once (this is the play button with the number 1 in figure 4). This by step execution of the instructions. Once the first instruction has been executed, look at register $t0 in the register table. The value in the third column should have changed to 1 once the first line was executed. Hit the step play button once again. Contents of register $t1 should be a 2 now.

Hit the reset button (this is the backward button show in figure 4). This resets the program and all of the register values and memory. Select a break point at the 5th instruction (the first syscall instruction). After the break point has been selected by checking the box in the bkpt column, go ahead and run the program by hitting the play button (this is the bigger play button). The program should have ran though instructions 1-4 and stopped at the 5th instruction. Registers $t0 should have the value 1, $t1 should have the value 2, $t2 should have the value 3, and $v0 should have the value 1. So far this program has completed the task of 1+2, and is now attempting to printing out the sum. Once syscall is executed, the system reads the code $v0 which tells the system that our program is requesting an input/output action. Our program is requesting the usage of a peripheral component, in this case, displaying a number on the monitor. Hit the step play button. The syscall line should have been executed. Notice the output in the window at the bottom of the screen. It did not print out the sum, but instead printed out 0. This is because we did not update the argument that is needed for the print integer syscall.

Hit the button with the question mark (see figure 4). Hit the syscall tab. Here is the table of all the syscalls available for you to use. Scroll down to the first listing. Notice that print integer syscall needs an argument in addition to the service number. In the editor add the following line in the space before the 1st syscall: move $a0, $t2. Save and reassemble. Now hit the play button. The program should execute all the instructions, printed out a 3, and then exited.

Answer the multiple questions on FSO under week 1 for lab 1.