

# Introduction to MARS and MIPS

## Requirements

1. Mac, Windows or Linux PC
2. Java standard edition (SE) version 9 or higher.
3. MARS Java program

## Introduction

In this lab, you will be introduced to the **MARS (MIPS Assembler and Runtime Simulator)** programming environment in which you will develop **MIPS** assembly language programs. MARS is a software simulator written using the Java programming language. The MARS simulator will allow you to edit, assemble, run and interact with MIPS assembly language programs. The assembly programs are text files with the extension “.asm”, for example “lab1.asm”.

## Basic MARS Use

Most Mac’s, Windows and Linux computers do not come with Java installed, good news is you can download it for free and install on your computer.

***MARS requires Java to be installed on your computer***, which is available here:

<https://www.oracle.com/technetwork/java/javase/downloads/index.html>

Oracle Technology Network / Java / Java SE / Downloads

Overview Downloads Documentation Community Technologies Training

### Java SE Downloads

 **DOWNLOAD**

Java Platform (JDK) 13

Java Platform, Standard Edition

**Java SE 13.0.1**  
Java SE 13.0.1 is the latest release for the Java SE Platform  
[Learn more](#)

- [Installation Instructions](#)
- [Release Notes](#)
- [Oracle JDK License](#)
- [Java SE Licensing Information User Manual](#)
  - Includes Third Party Licenses
- [Certified System Configurations](#)
- [Readme](#)

Oracle JDK **DOWNLOAD**

### Java SDKs and Tools

- [Java SE](#)
- [Java EE and Glassfish](#)
- [Java ME](#)
- [Java Card](#)
- [NetBeans IDE](#)
- [Java Mission Control](#)

### Java Resources


- [Java APIs](#)
- [Technical Articles](#)
- [Demos and Videos](#)
- [Forums](#)
- [Java Magazine](#)
- [Developer Training](#)
- [Tutorials](#)
- [Java.com](#)

MARS is available here:

<http://courses.missouristate.edu/KenVollmar/MARS/>

**Missouri State**  
UNIVERSITY

Search  
a b c d e f g h i j k l  
m n o p q r s t u v w x  
y z



**MARS (MIPS Assembler and Runtime Simulator)**  
*An IDE for MIPS Assembly Language Programming*

MARS is a lightweight interactive development environment (IDE) for programming in MIPS assembly language, intended for educational-level use with Patterson and Hennessy's *Computer Organization and Design*.

**100% FREE**  
NO SPYWARE  
NO ADWARE  
NO VIRUSES  
**SOFTPEDIA™**  
certified by www.softpedia.com

Feb. 2013: "MARS has been tested in the Softpedia labs using several industry-leading security solutions and found to be completely clean of adware/spyware components. ... Softpedia guarantees that MARS 4.3 is 100% FREE, which means it does not contain any form of malware, including spyware, viruses, trojans and backdoors."

[Download MARS from Softpedia](#) (version on Softpedia may lag behind the version on this page).

**Download MARS 4.5 software! (Aug. 2014)**

After you click on the link above you will be taken to the following screen



### Windows PC's

Depending on your computer's security settings you might see this screen, click on "Keep".

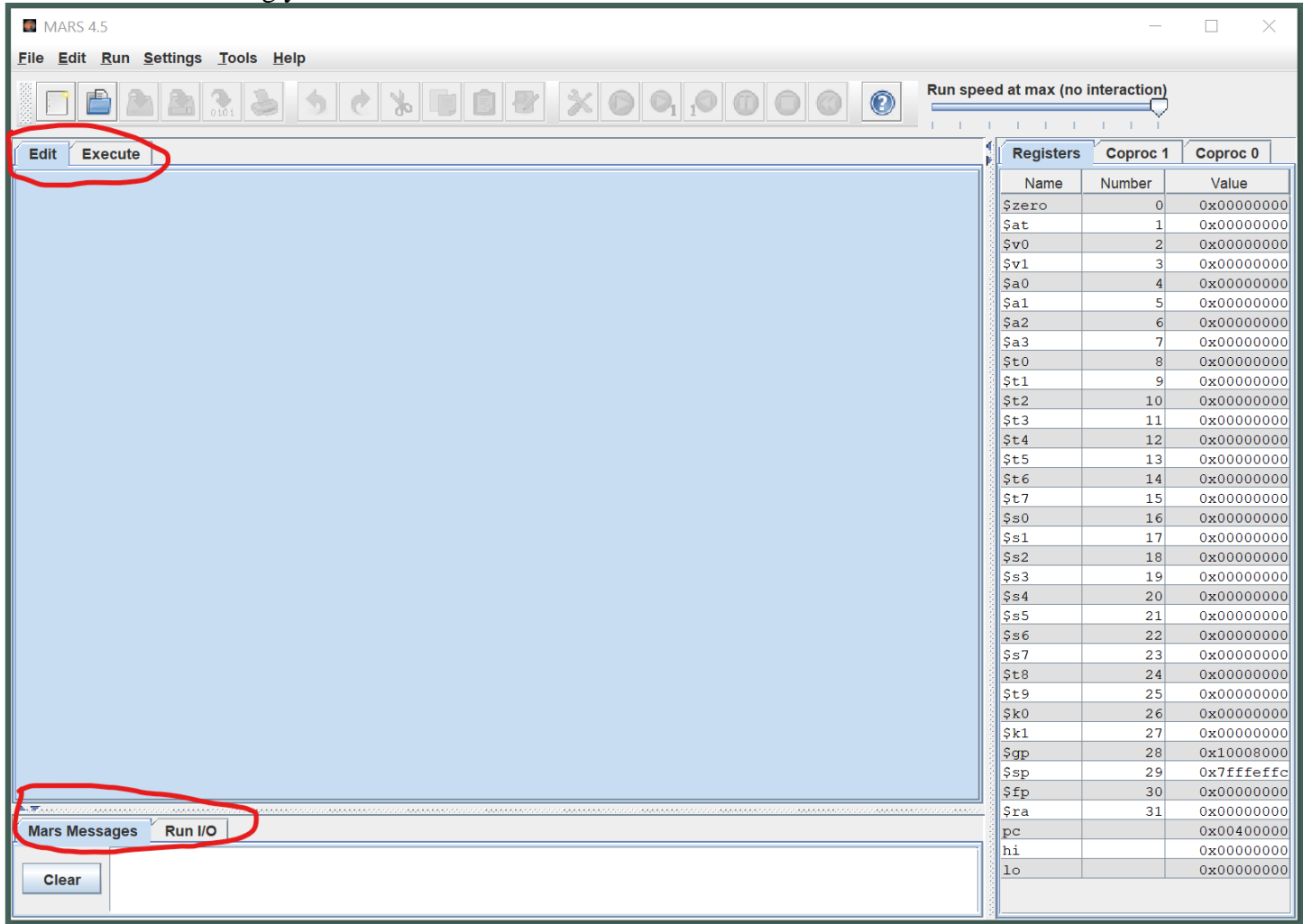


### MAC's

It is likely that Java is disabled from running on your MAC, click here for instructions on how to enable:  
<https://www.intertech.com/Blog/running-local-java-applets-on-a-mac/>

After you have installed Java you can run MARS by double clicking on the file “Mars4\_5.jar”.

Once MARS is running you will see this screen:

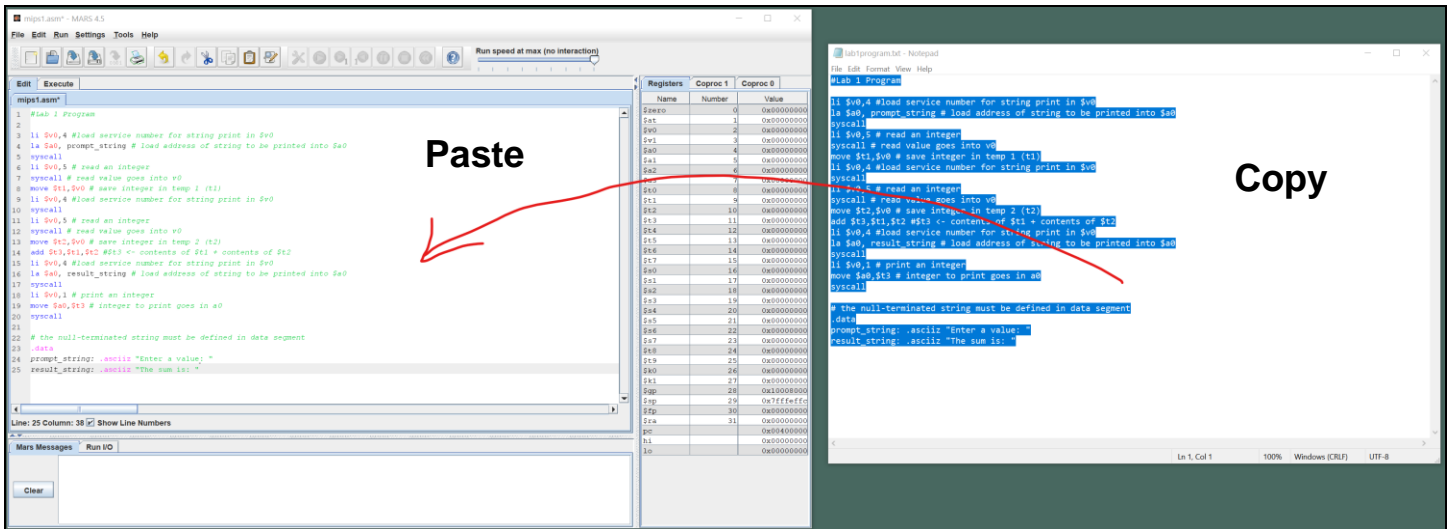


Notice the **Edit/Execute** tabs and the **Mars Messages** and **Run I/O** tabs. These are used to switch between the panels used for those purposes.

Also notice the **Registers** panel along the right-hand side of the screen. All 32 MIPS registers are labeled and numbered, in addition to 3 more listed at the bottom (the **pc**, or program counter, and **hi** and **lo** registers, which you will learn about later).


## Exercise 1:

- From the MARS screen, which registers have a non-zero value at this point?
- Similarly, explain what the value of the **pc** indicates:
- In MARS, Under **File** select **New**
- Copy** the **lab1program.txt** text that I posted in Course Studio and paste it (using the paste icon in the MARS command bar) into the **Edit** panel.

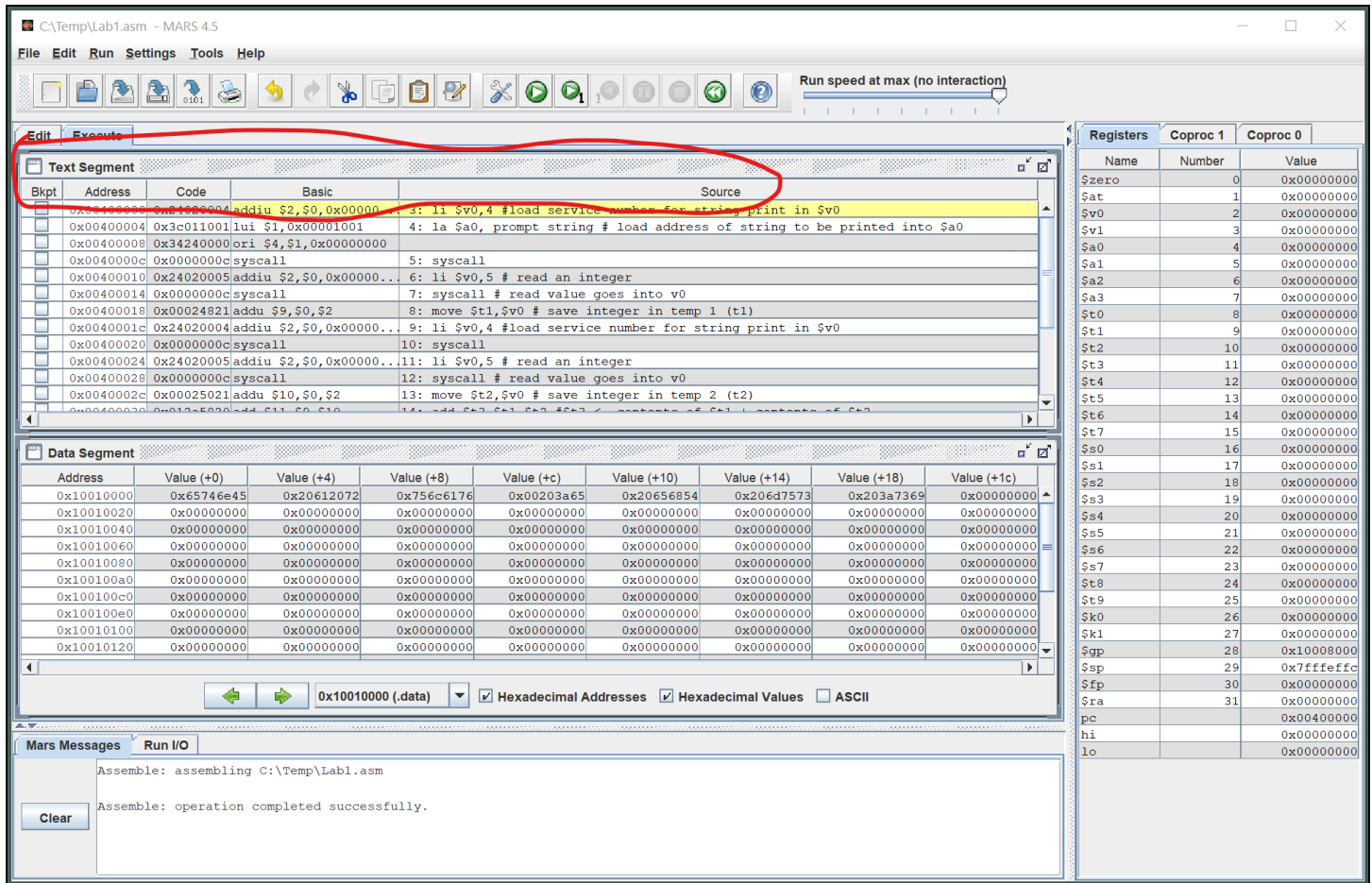


- Use **File...Save As** to save as **lab1.asm**, a MIPS program that adds two numbers.

**NOTE:** All icons have menu bar equivalents; the remainder of these steps will use the icon whenever possible.

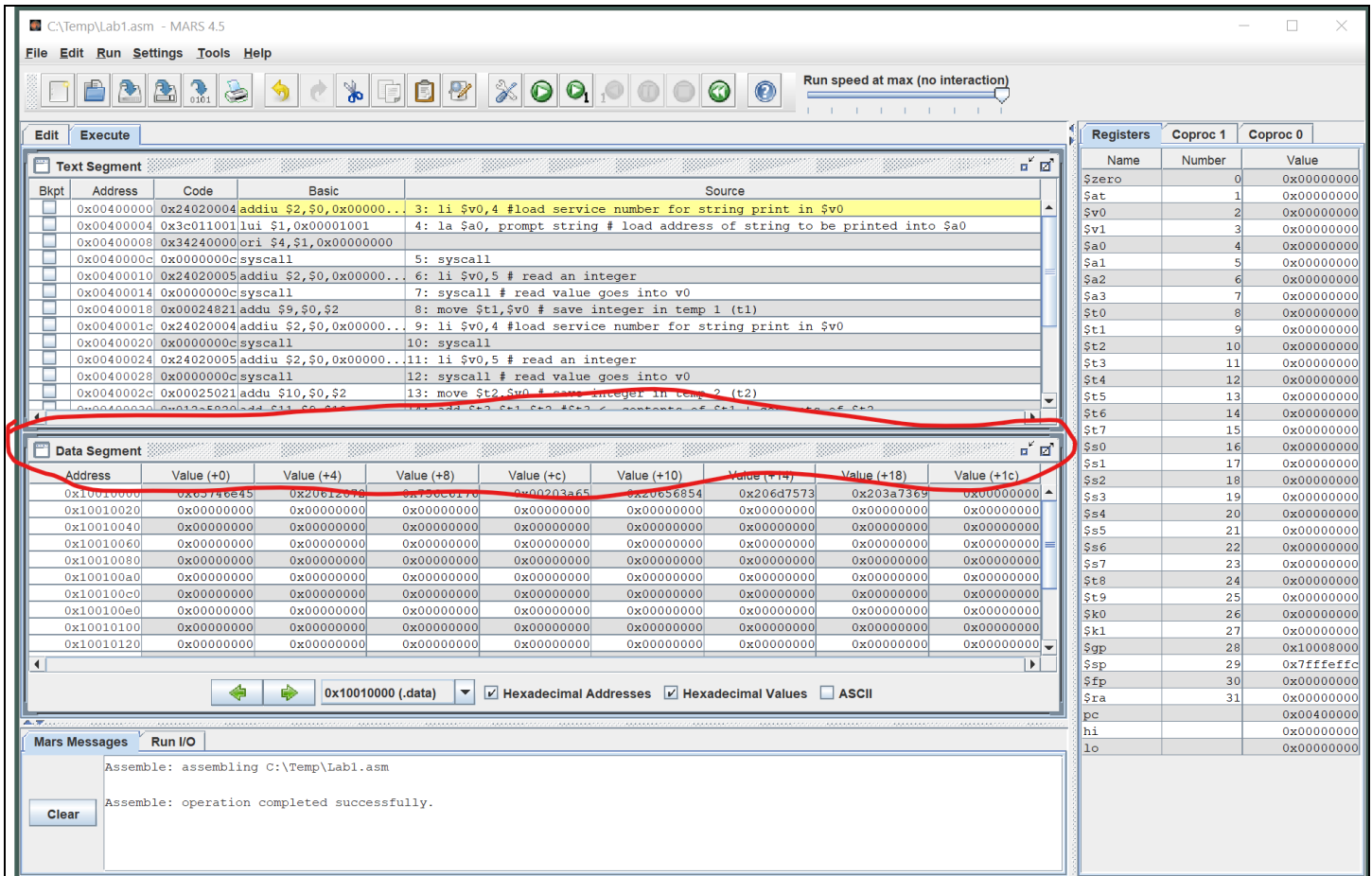
- Assemble the program using the icon  (also available from the **Run** menu).
- Examine the **Mars Messages** panel and notice that the message indicates the assembly was successful (hopefully).
- Also notice that the tab automatically changes from **Edit** to **Execute**, and that the **Text Segment** and **Data Segment** panels are now displayed (similar to the earlier screenshot).
- (Question) What does the **0x** notation mean which precedes the 8-digit numbers you see displayed in these panels?

## CS10 – Lab 1, Introduction to MARS and MIPS

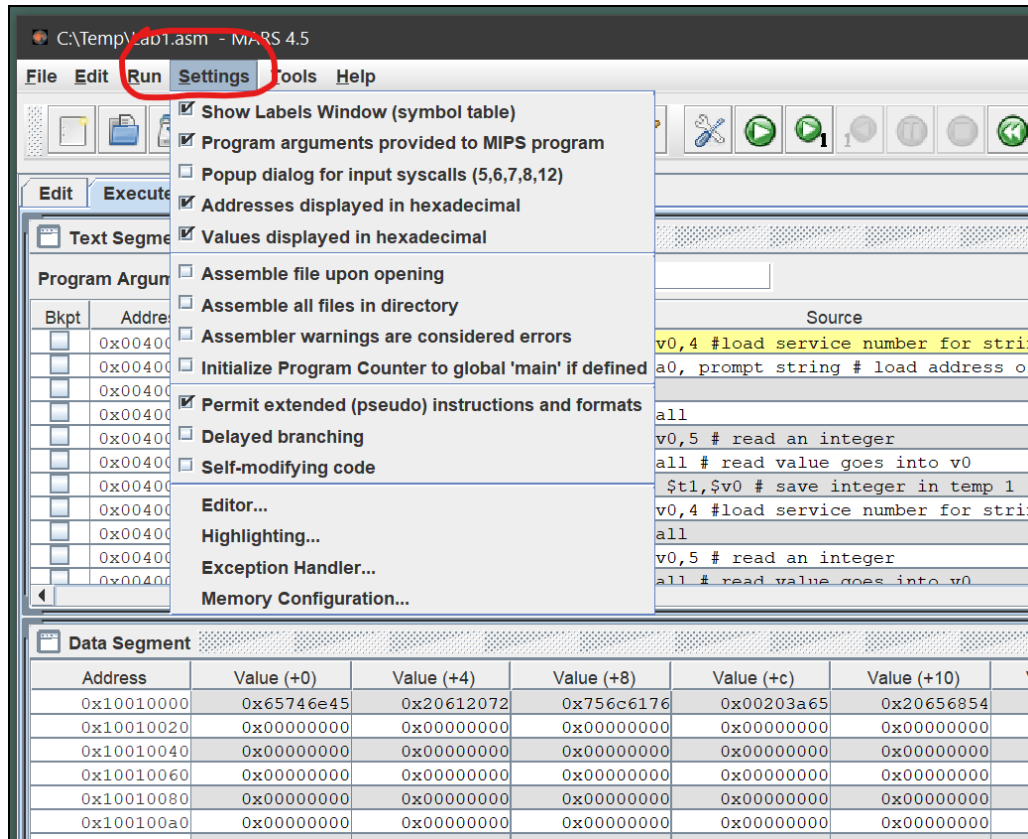


10. The **Text Segment** contains the code from the **.text** section of the program (the program instructions). Explain what you think each column in this panel is used for:
  11. Bkpt: \_\_\_\_\_
  12. Address: \_\_\_\_\_
  13. Code: \_\_\_\_\_
  14. Basic: \_\_\_\_\_
  15. Source: \_\_\_\_\_
16. What is the starting address of the program? \_\_\_\_\_
17. The **Data Segment** contains the code from the **.data** section of the program (the variables and constants defined in the program).

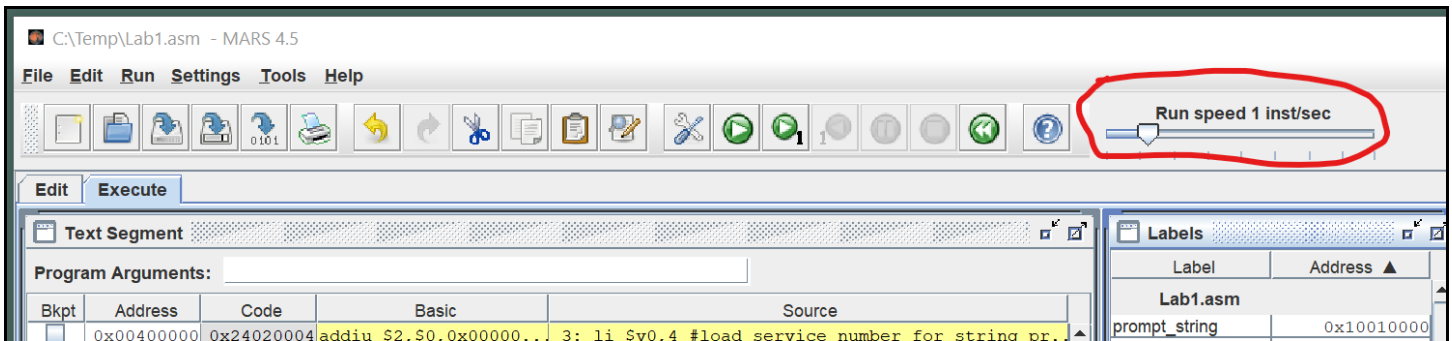




18. What is the starting address of the **Data Segment**? \_\_\_\_\_
19. Each row in the **Data Segment** lists the contents of 8 words in memory, each of which contains 32 bits, or 4 bytes, of data. Notice that the first 7 words in the **Data Segment** contain non-zero values.
20. Why are these non-zero for this program? \_\_\_\_\_
21. Use the Settings menu to configure the MARS displays. The settings will be retained for the next MARS session.
22. The Labels display contains the addresses of the assembly code statements with a label, but the default is to *not* show this display. Select the checkbox from the Settings menu.
23. Select the checkbox to allow pseudo-instructions (programmer-friendly instruction substitutions and shorthand).
24. Select the startup display format of addresses and values to be hexadecimal.



25. Use the slider bar to change the run speed to 1 instruction per second. This allows us to “watch the action” instead of the assembly program finishing directly.



There are several ways to execute the program:



The icon runs the program to completion. Using this icon, you should observe the yellow highlight showing the program’s progress in the **Text Segment**, and green highlight showing the registers being modified in the **Registers** panel. When there are changes to the **Data Segment**, they are also highlighted.










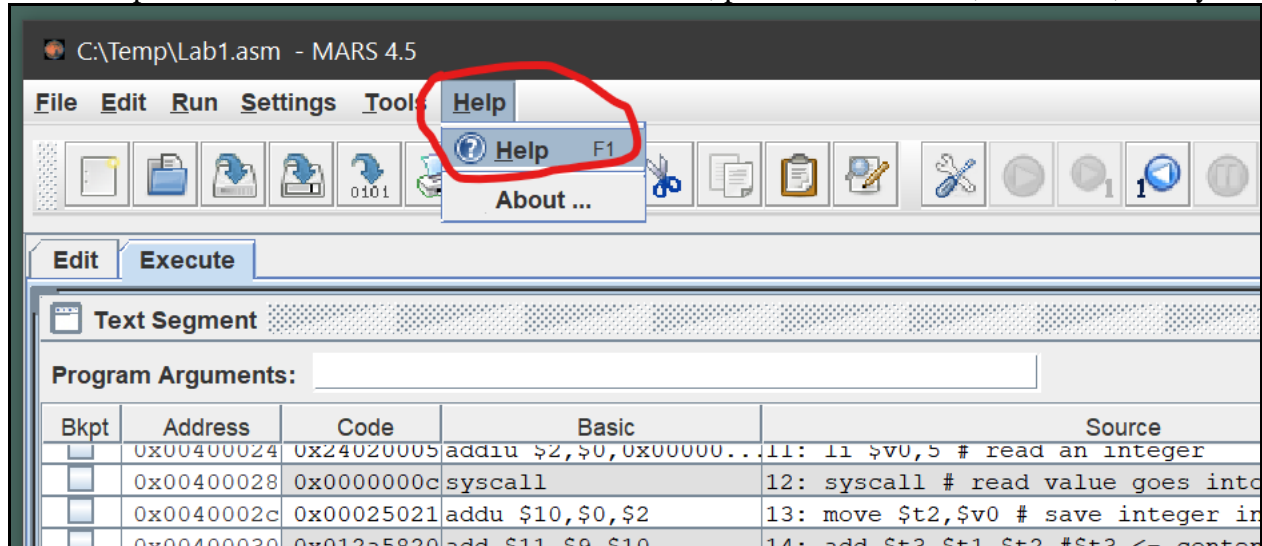
The icon resets the program and simulator to initial values. Memory contents are those specified within the program, and register contents are generally zero.



The icon is “single-step.” Its complement is , “single-step backwards” (undoes each operation).



26. Run the program to completion, using the very slow 1 second per instruction speed. You will need to enter two values in the **Run I/O** panel, to be used for the addition. Upon completion, the following will be displayed:
- Enter a value:2
  - Enter a value:3
  - The sum is:5
  - program is finished running –
27. Reset  and run the program one instruction at a time, using the single step . Examine the registers after each instruction to verify that you understand any new values.
28. Set a breakpoint at address 0x400030 by clicking on the checkbox at the left of the instruction.
29. Reset  and run  the program again, the program stops at the breakpoint, before executing the instruction.
30. Examine the value of **\$t3** at this point. What is it? \_\_\_\_\_
31. Perform a single step is  to execute the **add** instruction.
32. Examine the value of **\$t3** again. What is it now? \_\_\_\_\_
33. Click  to continue from the breakpoint. Note that you could modify register or memory values directly at a breakpoint before continuing, if it was necessary to do so for testing purposes.
34. Open the Help  for information on MIPS instructions, pseudo instructions, directives, and syscalls.



The screenshot shows the MARS 4.5 Help window. The 'MIPS' tab is selected and circled in red. Below the tabs, a green box titled 'Operand Key for Example Instructions' lists:
 

- label, target: any textual label
- \$t1, \$t2, \$t3: any integer register
- \$f2, \$f4, \$f6: even-numbered floating point register
- \$f0, \$f1, \$f3: any floating point register

 The 'Syscalls' tab in the navigation bar is also circled in red. The main content area is titled 'SYSCALL functions available in MARS' and includes an 'Introduction' section, a list of steps for using SYSCALL services, an example code snippet, and a 'Table of Available Services'.

**SYSCALL functions available in MARS**

**Introduction**

A number of system services, mainly for input and output, are available for use by your MIPS program. They are described in the table below.

MIPS register contents are not affected by a system call, except for result registers as specified in the table below.

**How to use SYSCALL system services**

Step 1. Load the service number in register \$v0.  
 Step 2. Load argument values, if any, in \$a0, \$a1, \$a2, or \$f12 as specified.  
 Step 3. Issue the SYSCALL instruction.  
 Step 4. Retrieve return values, if any, from result registers as specified.

**Example: display the value stored in \$t0 on the console**

```
li $v0, 1          # service 1 is print integer
add $a0, $t0, $zero # load desired value into argument register $a0, using pseudo-op
syscall
```

**Table of Available Services**

Service	Code in \$v0	Arguments	Result
print integer	1	\$a0 = integer to print	
print float	2	\$f12 = float to print	

Close

**Exercise 2:**

Now that you have seen the basic operation of MARS, try and write your own program!

1. Modify the add program so that it prompts you for your name and age, and outputs a message that greets you and tells you how old you will be in 4 years.

```
#to read in a string, do the following. The string will be stored in memory at location "answer"
li $v0,8          #system code for read string
la $a0,answer     #put address of answer string in $a0
lw $a1,length     #put length of string in $a1
syscall
```

```
#you also need the following definitions in your .data section for this to work:
answer: .space 51  #will hold up to 50 characters, so the name must be 50 characters or less
alength: .word 50
```

2. When you run your program, your console should look like the following:

```
What is your name? Harry Potter
What is your age? 11
Hello, Harry Potter
You will be 15 years old in four years
```

3. Format and comment your program appropriately in MARS.
4. Save your program with the name “**lab1ex2yourname.asm**” and **upload to Canvas**.

**Exercise 3:**

Write a MIPS program that plays the first dozen or so notes of your new hit single (or if you don’t have your own hit single, then your favorite!)

1. Go to the help screen from Exercise 1, step 34.
2. Experiment with system call 31, 32 and 33.
3. Put the notes to your song in a MIPS “array”
4. Format and comment your program appropriately in MARS.
5. Save your program with the name “**lab1ex3yourname.asm**” and **upload to Canvas**. If there is time, I will play a few of them in class.