Lab 1: MIPS Assembly Programming

Introduction

In this lab, you will be coding using MIPS assembly language using a free software called MARS (MIPS Assembler and Runtime Simulator). It is a lightweight interactive development environment (IDE) for programming in MIPS assembly language, intended for educational-level use. To code using MARS, it is assumed that you have the basic knowledge of MIPS assembly instruction. This lab guide will guide you through the code development process in two parts. The first part introduces the MARS software environment and the second guides you through the coding and debugging process. After completing this lab, you are required to do the designated homework.

1. MARS Software Environment

Objectives

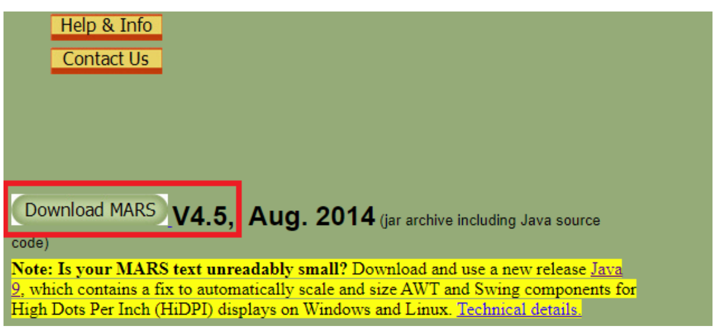
In the first part, you will go through the how to download, set-up, and familiarize MARS software interface and learn the basic settings and functionalities that you will need for the coding process.

A. Download and Set-up

>>To download MARS software go to this site:

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

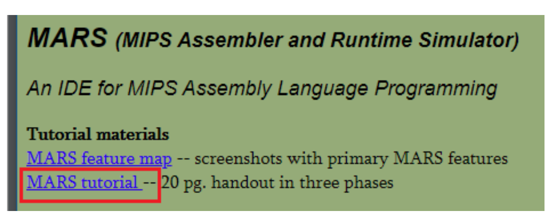
>>Then click the download button as indicated:



>>For more information on MARS simulator:

- Informative Tutorial: http://bits.usc.edu/files/ee109/documents/MARS_Tutorial.pdf
- Detailed Manual: https://courses.missouristate.edu/KenVollmar/mars/tutorial.htm

Then go to Tutorial Materials section. Click the link on MARS tutorial:



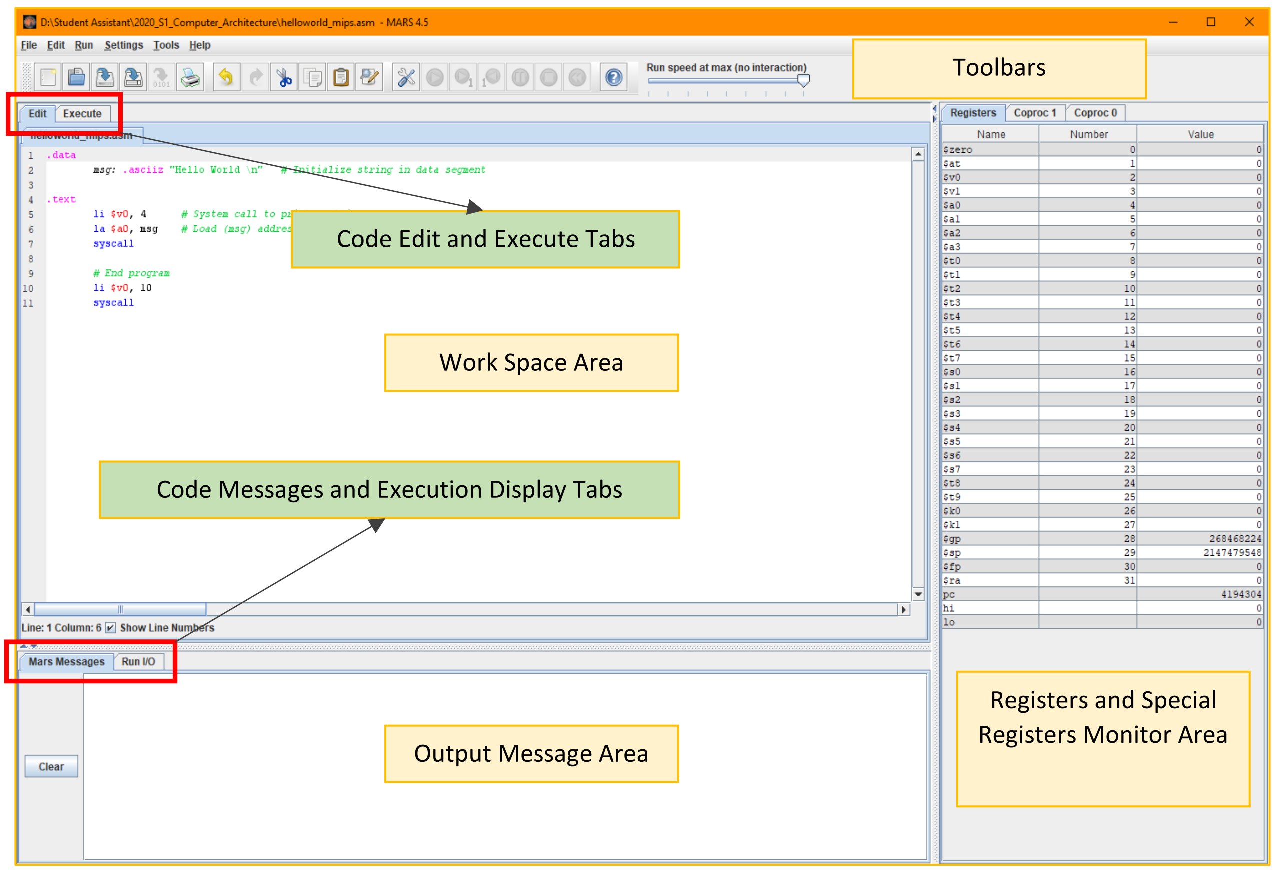
>>Note that MARS software does not need to be installed as this is already an executable Java program. Please download and install the latest version (9 and higher) of Java development kit (JDK) before executing the MARS software.

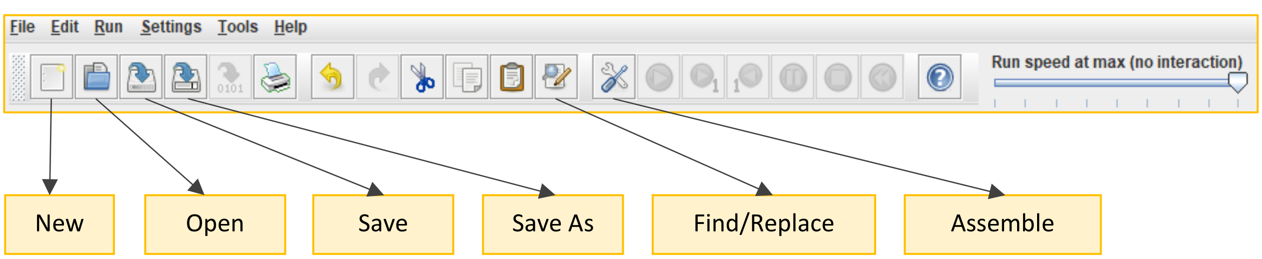
To download the latest JDK go to this site and find the compatible operating system:

https://www.oracle.com/java/technologies/javase-downloads.html

>>Then to start MARS just double click it in your file location.

B. Interface Environment





C. Basic Tool Functions

a. To create new <filename>.asm file click the ‘New’ icon in the toolbar or click File->New.

b. Save the file by clicking ‘Save As’ icon or click File->Save As then type a filename.

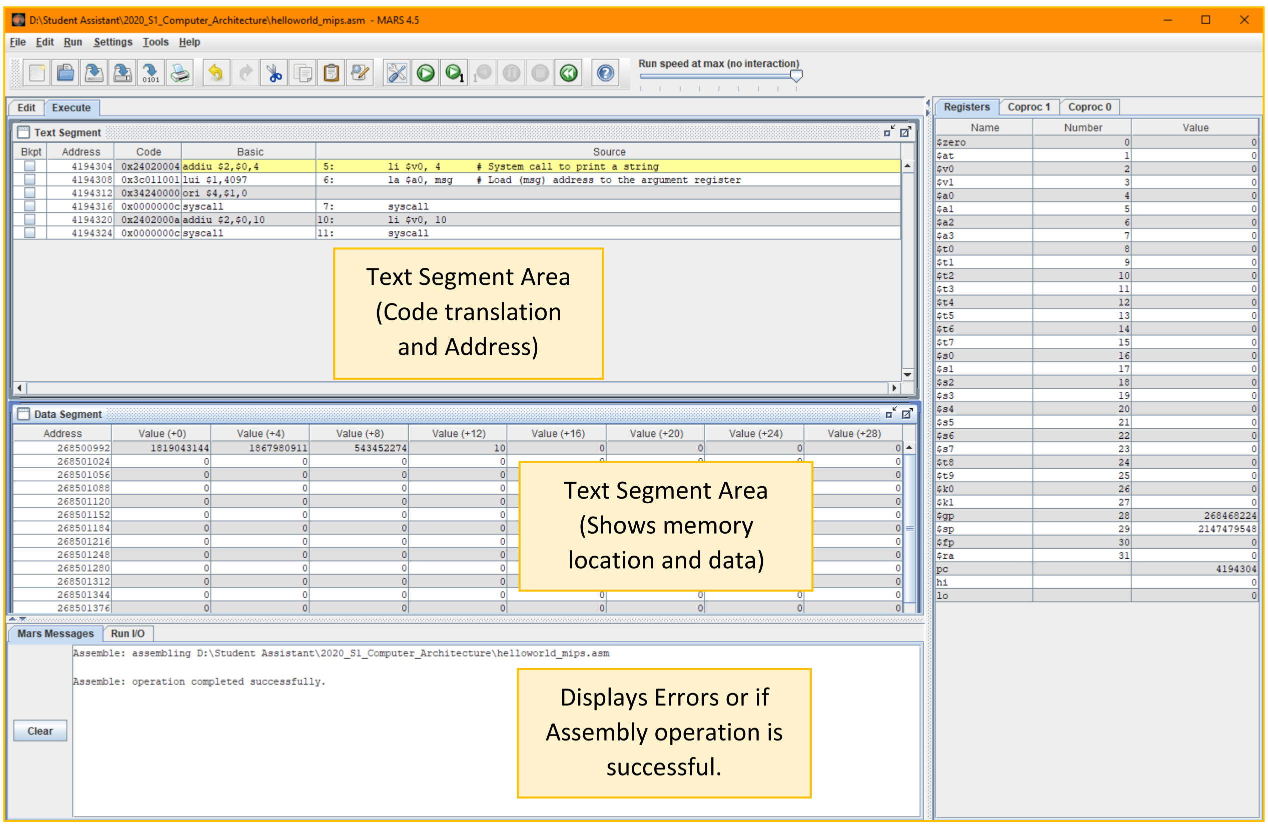
c. To execute the assembly file, click the tab of the assembly code you want to execute and save it first by clicking the ‘Save’ icon or File->Save then click the ‘Assemble’ icon or click Run->Assemble.

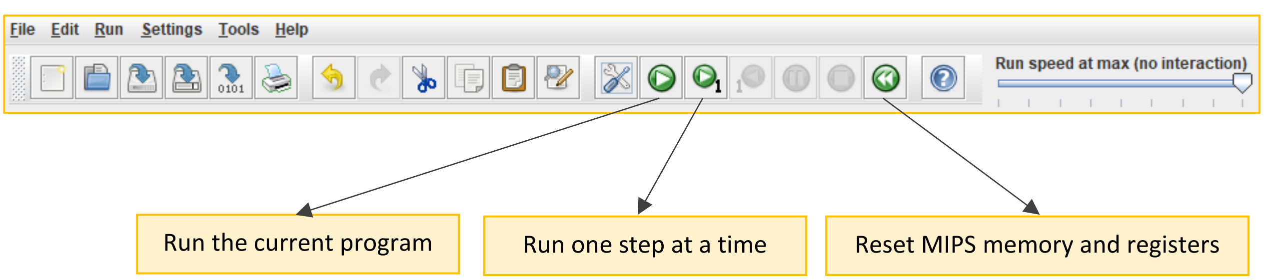
d. The register display values can be changed either to hexadecimal display or integer display. To change it, click Settings->values displayed in hexadecimal.

e. To close the assembly code file, click the tab of the assembly code then click File->Close.

D. Execute Functions

After successfully assembling the MIPS assembly code. The execution tab and execution functions will be displayed.





2. Coding MIPS Assembly Program

Objectives

In the second part, you will learn the coding and debugging process and also some of the MIPS assembler directives and system call functions to make a working program.

- MIPS Assembly Language Guide: http://www.cs.uni.edu/~fienup/cs041s08/lectures/lec20_MIPS.pdf

A. Hello World in MIPS

This programs prints a “Hello World” statement using the MIPS system call and makes use of the .asciiz assembler directive.

1. Create a new .asm file and name it “helloworld_mips.asm”. (File->New, then File->Save as-><filename>)

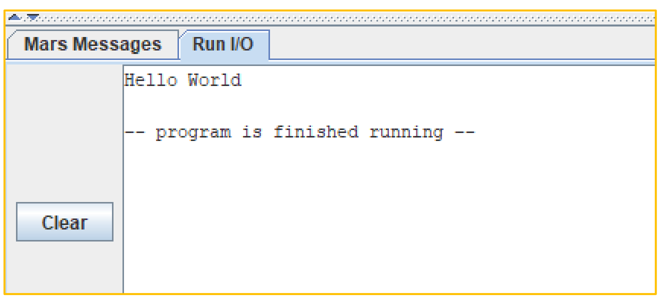
2. Copy the code below then save (File-> Save), assemble and run.

```
.data
        msg: .asciiz "Hello World \n"    # Initialize string in data segment

.text
        li $v0, 4       # System call to print a string
        la $a0, msg     # Load (msg) address to the argument register
        syscall

        # End program
        li $v0, 10      # System call to end the program
        syscall
```

>>Output of the program



>>Data Segment Output

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)
268500992	1819043144	1867980911	543452274	10	0
268501024	0	0	0	0	0
268501056	0	0	0	0	0
268501088	0	0	0	0	0
268501120	0	0	0	0	0
268501152	0	0	0	0	0
268501184	0	0	0	0	0
268501216	0	0	0	0	0

- “Hello World” string occupies three data words in the data segment.

B. Simple Addition

This programs adds two integers sequentially in the same register. The register $t0 should update in two time steps (Use Run One Step at a Time). Then, system call is used to print the output on the output box.

1. Create a new .asm file and name it “simple_add.asm” (File->New, then File->Save as-><filename>)

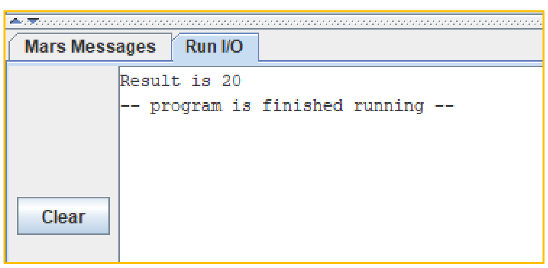
2. Copy the code below then save (File-> Save), assemble and run.

```
.data
        msg: .asciiz "Result is "    # Initialize string in data segment
.text
        # Add integer
        li $t0, 16              # Load immediate value to t0
        add $t0, $t0, 4         # Add 4 to the value of t0

        # Display result
        li $v0, 4               # System call to print a string
        la $a0, msg             # Load (msg) address to the argument register
        syscall
        li $v0, 1               # System call to print an integer
        move $a0, $t0           # Move value from t0 to a0
        syscall

        # End program
        li $v0, 10              # System call to end the program
        syscall
```

>>Output of the program



>>Register Output

Step 1

$a3	7	0
$t0	8	16
$t1	9	0

Step 2

$a3	7	0
$t0	8	20
$t1	9	0

C. Debugging Error

1. Modify the “simple_add.asm” to intentionally put an error line

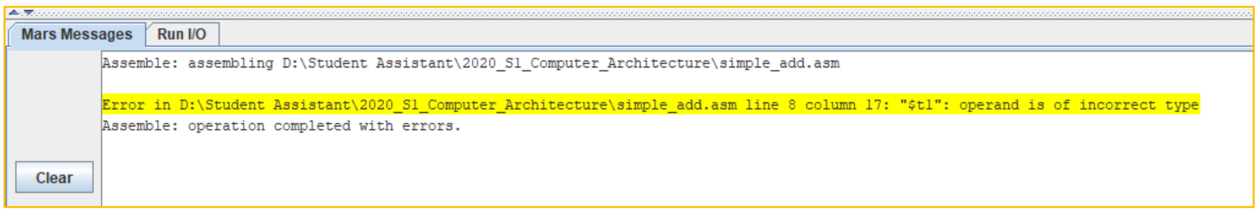
2. Put < addi $t0, $t0, $t1 > after the < add $t0, $t0, 4 > as indicated below. It should prompt an error because < addi > instruction should be used to add immediate value not a value from a register.

3. Then save and assemble it. It should display an error message upon assembling.

```
.text
        # Add integer
        li $t0, 16              # Load immediate value to t0
        add $t0, $t0, 4         # Add 4 to the value of t0
        addi $t0, $t0, $t1      ######## ERROR LINE ########

        # Display result
```

>>Error Message



- Double the error message to highlight the line that caused the error.

```
1
2  .data
3          msg: .asciiz "Result is "    # Initialize string in data segment
4  .text
5          # Add integer
6          li $t0, 16              # Load immediate value to t0
7          add $t0, $t0, 4         # Add 4 to the value of t0
8          addi $t0, $t0, $t1      ######## ERROR LINE ########
9
10         # Display result
11         li $v0, 4               # System call to print a string
12         la $a0, msg             # Load (msg) address to the argument register
13         syscall
```

D. Bubble Sort Ascending Order

So far the first three exercises are just for warm up. Now, you will code an n-element sorting algorithm using bubble sort with the elements sorted in ascending order. This program takes a user input for the length of the input array (*how many elements in the array*) up to a maximum of 64 elements. Then, the user must enter each element one by one.

Example:

```
.....Bubble Sort.....
Enter Input Length: 4

Enter Input Values:
-67
45
603
-4
```

1. Create a new .asm file and name it “bubble_sort.asm”. (File->New, then File->Save as-><filename>)

2. Copy the code below then save (File-> Save), assemble and run.

```
.data
        array: .space 256          # Reserve in data segment up to a maximum of 64 integers
        msg1: .asciiz ".....Bubble Sort....."
        msg2: .asciiz "\nEnter Input Length: "
        msg3: .asciiz "\nEnter Input Values: "
        msg4: .asciiz "\nSorted Output Values:"
        newl: .asciiz "\n"
.text
     main:
        # Display program title
        li $v0, 4                  # System call to print a string
        la $a0, msg1               # Load (msg1) address to the argument register
        syscall

        # prompt user to enter input length
        la $a0, msg2               # Load (msg2) address to the argument register
        syscall

        # Get the user's input
        li $v0, 5                  # System call to get integer from the keyboard
        syscall
        move $t0, $v0              # Move the user input to $t0

        li $v0, 4                  # System call to print a string
        la $a0, msg3               # Load (msg3) address to the argument register
        syscall
        jal newline                # Call newline

        addi $t1, $zero, 0         # Initialize scanloop counter
        addi $t2, $zero, 0         # Initialize data segment address counter

    scanloop:
        beq $t0, $t1, initsort     # If counter is equal to input length in $v0 then branch initsort
        li $v0, 5                  # System call to get integer from the keyboard
        syscall
```

>>Continuation

```
        sw $v0, array($t2)

        addi $t2, $t2, 4            # Update data segment address counter
        addi $t1, $t1, 1            # Update loop counter

        j scanloop                  # Goto scanloop

    initsort:
        subi $t0, $t0, 1            # Initialize bubblesort max count
        addi $t3, $zero, 0          # Initialize i counter

    outerloop:
        beq $t0, $t3, initdisp

        addi $t2, $zero, 0          # Initialize data segment address counter
        addi $t4, $zero, 0          # Initialize j counter

        j innerloop

    nexti:
        addi $t3, $t3, 1            # Update i counter by 1
        j outerloop                 # Goto outerloop

    innerloop:
        beq $t0, $t4, nexti

        lw $t5, array($t2)          # Load from data segment address plus $t2 offset to $t5
        addi $t2, $t2, 4            # Add offset by 4 bytes
        lw $t6, array($t2)          # Load from data segment address plus $t2 offset to $t6

        bgt $t5, $t6, swap          # If $t5 is greater than $t6 then swap

    nextj:
        addi $t4, $t4, 1            # Update j counter by 1
        j innerloop                 # Goto innerloop
```

```
    swap:
        subi $t2, $t2, 4            # Subtract offset by 4 bytes
        sw $t6, array($t2)          # Store from $t6 to data segment address plus $t2 offset
        addi $t2, $t2, 4            # Add offset by 4 bytes
        sw $t5, array($t2)          # Store from $t5 to data segment address plus $t2 offset
        j nextj                     # return to nextj

    initdisp:
        addi $t0, $t0, 1            # Initialize display max count
        addi $t1, $zero, 0          # Initialize display counter
        addi $t2, $zero, 0          # Initialize data segment address counter

        li $v0, 4                   # System call to print a string
        la $a0, msg4                # Load (msg4) address to the argument register
        syscall
        jal newline                 # Call newline

    display:
        beq $t0, $t1, end           # If counter is equal to max count in $t0 then branch end

        li $v0, 1                   # System call to print an integer
        lw $a0, array($t2)          # Load from data segment address plus $t2 offset to $a0
        syscall
        jal newline                 # Call newline
```

>>Continuation

```
        addi $t2, $t2, 4            # Update data segment address counter
        addi $t1, $t1, 1            # Update display counter

        j display                   # Goto display

    newline:
        li $v0, 4                   # System call to print a string
        la $a0, newl                # Load (newl) address to the argument register
        syscall
        jr $ra                      # Jump to return address

    end:
        li $v0, 10                  # System call to end the program
        syscall
```

>>Sample Output of the program

Mars Messages | Run I/O

Clear

```
Result is 20
-- program is finished running --

.....Bubble Sort.....
Enter Input Length: 4

Enter Input Values:
-67
45
603
-4

Sorted Output Values:
-67
-4
45
603
```

>>Data Segment Output

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)
268500992	-67	-4	45	603	0	0
268501024	0	0	0	0	0	0
268501056	0	0	0	0	0	0
268501088	0	0	0	0	0	0
268501120	0	0	0	0	0	0
268501152	0	0	0	0	0	0
268501184	0	0	0	0	0	0

- The values stored in the data segment is also sorted because the data segment represents the stored array while the registers are only involved in the process of input and output operations.

HOMEWORK: BUBBLE SORT ODD-EVEN

Modify the n-element bubble sort to sort the input elements in odd and even sets then sort each set in ascending order.

Example:

Input: 15, 28, 9, 45, 4, 16, 33, 44, 89, 2

Output:

>>Sorted Odd: 9, 15, 33, 45, 89

>>Sorted Even: 2, 4, 16, 28, 44

Then, present the following:

1. What changes did you make to do this?
2. Show the screen capture of the modified part of the code.
3. Test the modified code with more than 30 elements then show the results. Also show the output of the data segment.