

EE 443**Lab #1: Assembly Language Programming I;
MIPS Instruction Set.****Assigned** 1/31/2023**Due** 2/07/2023 (2:00 pm)**Introduction.**

The goal of this laboratory is to introduce you to assembly language programming of the MIPS microprocessor using one of the MIPS assembly language simulators. The simulator we will be using for this laboratory exercise is *MARS* (available for download online).

ASSIGNMENT**Objective**

The main goal of this assignment is to become familiar with the MARS simulator. Simulators should be already installed on lab computers. You can download your copy of the MARS simulator from the following address:

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

Students should be able to step through the program and observe changes in the registers and in the memory of a MIPS microprocessor.

1. Go through the tutorial and perform all the tasks listed there. The tutorial for this Lab is a shortened and revised version of the full tutorial posted on Blackboard last week. It was written by the authors of the MARS simulator and slightly changed by the instructor.
2. Download the programs “Fibonacci.asm” and “row-major.asm” from Blackboard, you will need them for this lab assignment.
3. Change the program to prompt the user for the length of the Fibonacci sequence to generate.
4. *Follow the template for laboratory reports; the template is available for download on Blackboard*
5. Make sure that the assembly language code you wrote is well commented
6. **SEND THE PROGRAM** to your TA by email, before the due date and time.
7. Please demonstrate the functionality of the program to your TA in the lab, and when turning in your report, be prepared to answer questions about the programming environment and your programs
8. **Answer all the questions** (and fill in all the blanks) from the tutorial and include those pages in your report.

Grading (for TA):

_____	Printout of an assembly language program
_____	Student displays understanding of assembly environment
_____	Student demonstrates a working program
_____	Student answered questions

An Assembly Language I.D.E. To Engage Students of All Levels

* A Tutorial *

Pete Sanderson, Otterbein College, PSanderson@otterbein.edu
 Ken Vollmar, Missouri State University, KenVollmar@missouristate.edu

MARS is a software simulator for the MIPS assembly language intended for educational use. We will explore the capabilities of MARS release 3.2.1 in this three part tutorial (*note: some of the command might look slightly different in the current 4.5 release*).

MARS may be downloaded from www.cs.missouristate.edu/MARS.

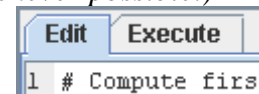
Part 1: Basic MARS Use



The example program is `Fibonacci.asm` to compute everyone's favorite number sequence.

1. Start MARS from the Start menu or desktop icon.

2. Use the menubar File...Open or the Open icon  to open `Fibonacci.asm`. (*All icons have menubar equivalents; the remainder of these steps will use the icon whenever possible.*)

- Select the Edit tab in the upper right to return to the program editor.
- The MIPS comment symbol is `#`. All characters on the line after the character `#` are ignored.



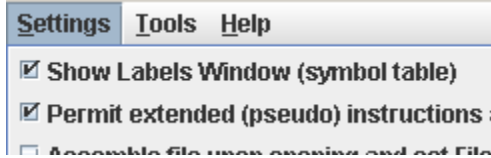
3. The provided assembly program is complete. Assemble the program using the icon . **Identify all the assembly instructions that were compiled into different instructions from those written by the programmer, including those that compiled into more than one instruction. List those instructions (both original and compiled versions) in your report and explain.**
4. Identify the location and values of the program's initialized data. Use the checkbox to toggle the display format between decimal and hexadecimal ☐ Hexadecimal Values.
 - The twelve-element array `fib` is initialized to zero, at addresses  `0x10010000 ... _____?`
 - The data location `size` has value `12ten` at `_____?`
 - The addresses `_____ ... _____` contain null-terminated ASCII strings.

Use the checkbox to toggle the display format between decimal and hexadecimal,

☐ Hexadecimal Values.

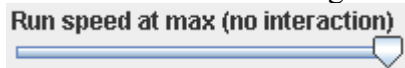
- Use the Settings menu to configure the MARS displays. The settings will be retained for the next MARS session.

- 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.







- Select your preference for allowing pseudo-instructions (programmer-friendly instruction substitutions and shorthand).
 - Select your preference for assembling *only one* file, or *many* files together (all the files in the current folder). This feature is useful for subroutines contained in separate files, etc.
 - Select the startup display format of addresses and values (decimal or hexadecimal).
- Locate the Registers display, which shows the 32 common MIPS registers. Other tabs in the Registers display show the floating-point registers (Coproc 1) and status codes (Coproc 0).

- Use the slider bar to change the run speed to about 10 instructions per second.



This allows us to “watch the action” instead of the assembly program finishing directly.



- Choose how you will execute the program:


- The  icon runs the program to completion. Using this icon, you should observe the yellow highlight showing the program’s progress and the values of the Fibonacci sequence appearing in the Data Segment display.
- 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).

- Observe the output of the program in the Run I/O display window:

The Fibonacci numbers are:
1 1 2 3 5 8 13 21 34 55 89 144 ...
-- program is finished running --

- Modify the contents of memory. (Modifying a register value is exactly the same.)






- Set a breakpoint at the first instruction of the subroutine which prints results. Use the checkbox at the left of the instruction in the Execute window.
- Reset  and re-run  the program, which stops at the breakpoint.
- You can double-click in one of the memory locations containing the computed Fibonacci numbers. The cell will be highlighted and will accept keyboard entry, similar to a spreadsheet. Enter “1111” as the value and use the Enter key or click outside the cell to indicate that the change is complete.

- Click  to continue from the breakpoint. The program output includes your entered value instead of the computed Fibonacci number. What is the “wrong” value displayed in the Fibonacci sequence?



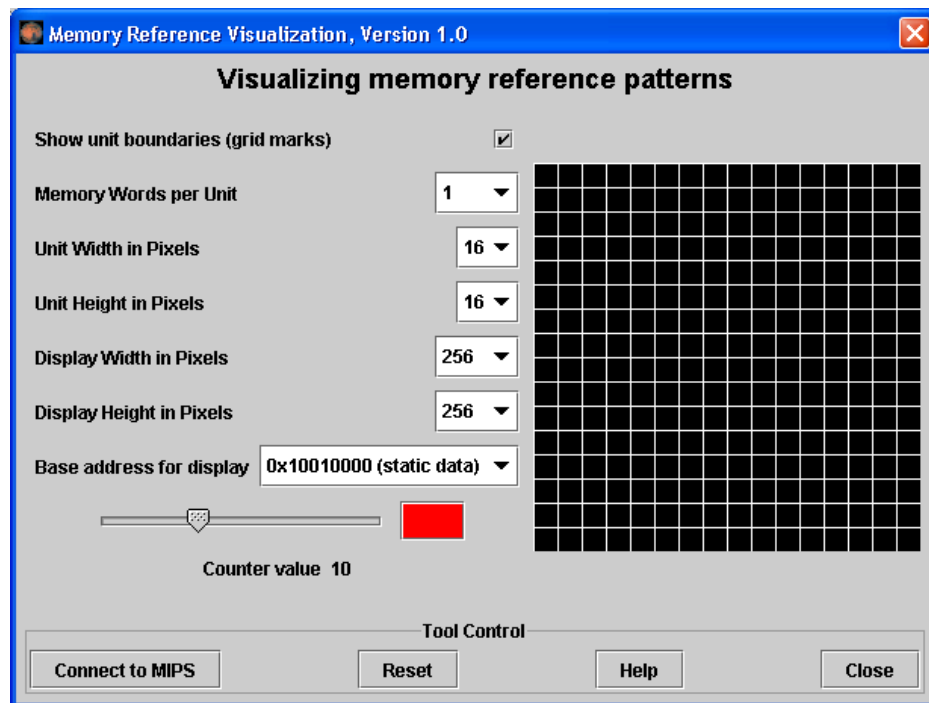
- Open the Help  for information on MIPS instructions, pseudoinstructions, directives, and syscalls.

- Modify the program so that it prompts the user for the Fibonacci sequence length:

-  **Write a new program fragment that will prompt the user for the length of the Fibonacci sequence to generate, in the range $2 \leq x \leq 19$. (The length of the sequence must be limited to the size of the declared space for result storage.)**
- Determine the correct `syscall` parameter to perform “read integer” from the user (see the table in Figure B.9.1, attached to this Lab assignment. The list of `syscall` parameter may also be found at Help  ... Syscall tab...read integer service. The completed line will have the form `li $v0, 42` (where in this case 42 is not the right answer).
- Reset  and re-run  the program. The program will stop at the breakpoint you inserted previously. Continue and finish with .

MARS Tools Activity: The Memory Reference Visualization tool


1. Open the program `row-major.asm` if it is not already open.
2. Assemble the program.
3. From the **Tools** menu, select **Memory Reference Visualization**. A new frame will appear in the middle of the screen.



This tool will paint a grid unit each time the corresponding MIPS memory word is referenced. The base address, the first static data segment (`.data` directive) word, corresponds to the upper-left grid unit. Address correspondence continues in row-major order (left to right, then next row down).

The color depends on the number of times the word has been referenced. Black is 0, blue is 1, green is 2, yellow is 3 and 4, orange is 5 through 9, red is 10 or higher. View the scale using the tool's slider control. You can change the color (but not the reference count) by clicking on the color patch.

4. Click the tool's **Connect to MIPS** button. This causes the tool to register as an observer of MIPS memory and thus respond during program execution.
5. Back in MARS, adjust the **Run Speed slider** to 30 instructions per second.
6. Run the program. Watch the tool animate as it is updated with every access to MIPS memory. *Feel free to stop the program at any time.*

7. Hopefully you observed that the animation sequence corresponded to the expected memory access sequence of the row-major.asm program. *If you have trouble seeing the blue*, reset the tool, move the slider to position 1, change the color to something brighter, and re-run.
8. Repeat again for **fibonacci.asm** to observe the animated pattern of memory references. Adjust the run speed and re-run if necessary.
 -  a. **Which memory location is used most frequently? Which variable this memory location corresponds to? How many times was this location accessed?** If it was accessed more than 20 times, you can give a range (e.g., 40-50 times) as an answer.

Service	System call code	Arguments	Result
print_int	1	\$a0 = integer	
print_float	2	\$f12 = float	
print_double	3	\$f12 = double	
print_string	4	\$a0 = string	
read_int	5		integer (in \$v0)
read_float	6		float (in \$f0)
read_double	7		double (in \$f0)
read_string	8	\$a0 = buffer, \$a1 = length	
sbrk	9	\$a0 = amount	address (in \$v0)
exit	10		
print_char	11	\$a0 = char	
read_char	12		char (in \$v0)
open	13	\$a0 = filename (string), \$a1 = flags, \$a2 = mode	file descriptor (in \$a0)
read	14	\$a0 = file descriptor, \$a1 = buffer, \$a2 = length	num chars read (in \$a0)
write	15	\$a0 = file descriptor, \$a1 = buffer, \$a2 = length	num chars written (in \$a0)
close	16	\$a0 = file descriptor	
exit2	17	\$a0 = result	

FIGURE B.9.1 System services.

```

li      $v0, 4      # system call code for print_str
la      $a0, str    # address of string to print
syscall                      # print the string

li      $v0, 1      # system call code for print_int
li      $a0, 5      # integer to print
syscall                      # print it

```

The `print_int` system call is passed an integer and prints it on the console. `print_float` prints a single floating-point number; `print_double` prints a double precision number; and `print_string` is passed a pointer to a null-terminated string, which it writes to the console.

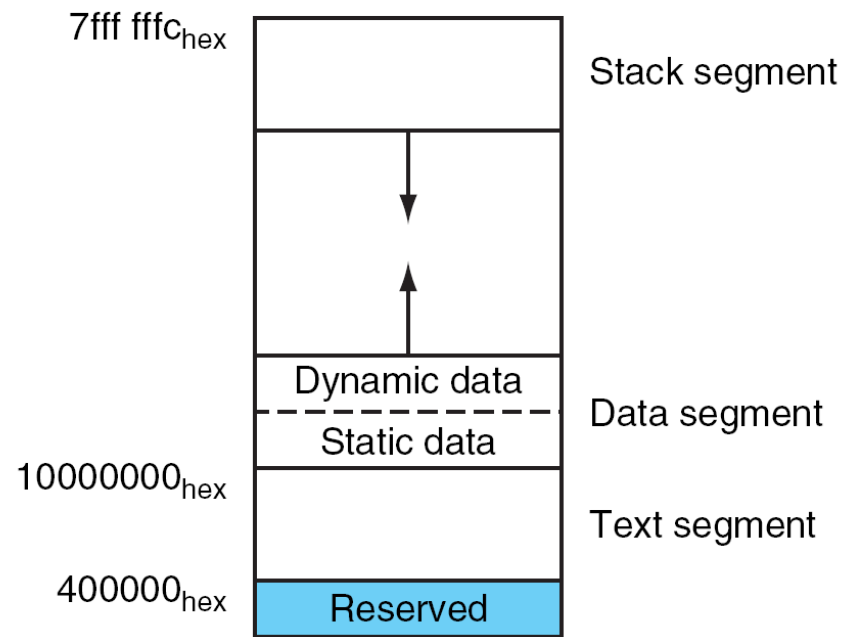


FIGURE B.5.1 Layout of memory.

Note that the Static data segment in MARS simulator starts at the address 10010000_{hex}