

LAB ASSIGNMENT-03

Computer Architecture

Ritik Tiwari

B21CS098



MARS SOFTWARE

MARS stands for "MIPS Assembly and Runtime Simulator." It is a piece of software made for writing in MIPS assembly language. One of its best features is that it has an integrated development environment (IDE) that lets you write, assemble, and run MIPS assembly code. This makes it a useful tool for workers and students who want to learn or work with MIPS architecture.

Key features of MARS

- 1) Assembler: Users can write MIPS assembly code in MARS and then have it put together into machine code that can be run. It checks for errors in real time and highlights incorrect grammar to help you find problems in the code.
- 2) Runtime Simulator: There is a runtime simulator in the program that can run MIPS machine code. This model lets users see the MIPS CPU in great detail, including the program counter, registers, and memory. They can then step through their code, set breakpoints, and watch the program run.
- 3) Interactive Interface: The MARS software incorporates a user-friendly graphical interface with a code editor, an output terminal, and windows for monitoring CPU and memory operations during simulation.
- 4) Debugging Tools: The MARS suite provides debugging features including single-step execution, breakpoints, and variable watches. These utilities facilitate the process of detecting and resolving defects in MIPS assembly code.
- 5) Support for Syscalls: MARS provides a variety of system calls (syscalls) that emulate input/output (I/O) tasks, such as keyboard reading or console writing, frequently needed in MIPS assembly applications.

MIPS PROGRAMMING

MIPS programming, short for Microprocessor without Interlocked Pipeline Stages, is the act of developing software using the MIPS assembly language tailored for programming processors that are based on the MIPS architecture. MIPS is a prevalent RISC (Reduced Instruction Set Computer) architecture.

Overview of the MIPS Architecture:

- 1) RISC Architecture: MIPS is a RISC architecture, meaning it uses a small, highly optimized set of instructions. This simplicity allows for faster instruction execution and easier instruction pipelining.
- 2) 32 Registers: MIPS processors typically have 32 general-purpose registers (\$0 to \$31), where most computations take place. Register \$0 is hardwired to the value 0.
- 3) Load/Store Architecture: In MIPS, arithmetic operations can only be performed on data stored in registers. To work with memory, data must first be loaded into a register, manipulated, and then stored back to memory if needed.
- 4) Fixed Instruction Length: Each MIPS instruction is 32 bits long, making instruction decoding straightforward and uniform.

To install the MARS software on your PC, you must have Java installed, and it should be the latest version. Initially, I had Java version 18 installed, but I was unable to run the MARS software until I upgraded Java to the latest version. After upgrading Java, the software ran without issues. To download the MARS JAR file, visit the official MARS website. The link is mentioned in the "Resources Used" section.

SubTask 1: Concatenate

This subtask is to concatenate two strings. Here my approach is to use the another empty string declaration and first iterate on the first string, and for each character in the first string, just take it on the resultant string, and then after the end of the first string we iterate through the second string to concatenate the string, and in return we just print the resultant string.

SubTask 1 Output

```
RitikTiwari
-- program is finished running --
```

SubTask 2: Input/Output: Pallindrome

This subtask is to check the given string by the user is pallindrome or not, so let's first define what a pallindrome string is. A string is said to be a pallindrome if I read the string from right to left, and when read from left to right, it results in the same string with the characters present at the indexes equal in both the strings. Like, take an example of "MOM." This string is a pallindrome. So to approach this task, I have many ideas, but the simplest one is to take two registers in which the first points at index 0 and the second one at index $n-1$, where n is the length of the string, and compare characters at those indexes, and if matched, move the first index to the next character while the second index moves to the lesser index, and this will go till we can't find any index where the characters are the same or the first index crosses the second index, and I have to return 'P' if the given string is a pallindrome; otherwise, return 'NP'. So this is the basic approach, also called the two-pointer approach. Hence, I have used this approach to solve this subtask.

SubTask 2 Output

```
Enter a string: radar
P
-- program is finished running --

Enter a string: mom
P
-- program is finished running --

Enter a string: ritik
NP
-- program is finished running --
```

SubTask 3: Reverse

This task involves reversing the string given by the user. The approach to solving this problem is to take an empty string and move the given string in reverse order using the change of the index, like when I have a register that, let's say, is used to iterate through the string and its initial value is 0, i.e., at the 0th index, then our goal is to access the last index of the given string, and to do so, we just write like this: `s[size-index-1]` where `s` denotes the given string, `size` is the size of the given string, `index` is the current index, and this value of the string we store at `index` of the resultant string, and with this, when we reach the loop end, we have an answer in the final resultant string. So this is the index that I have used to reverse the given string.

SubTask 3 Output

```
Messages | Run I/O
Enter a string: ritikB21CS098

Reversed string:
890SC12Bkitir
-- program is finished running --
```


SubTask 4: Size of the String

This task is to return the size of the given string. To accomplish this task, we need to create a variable with an initial value of 0. As we iterate through the given string, we increment the value of that register by 1. The loop ends when we reach the last element of the string. At the null terminator, we simply break the loop. So, this is the basic approach to determining the size of the given string.

SubTask 4 Output

```
Messages | Run I/O
Enter a string: ritiktiwari

The size of the string is: 11
-- program is finished running --
```



Challenges Faced: I have not faced such a specific challenge but yes, this is the new language, which I have learned so familiar with the syntax it takes a little bit of time otherwise the MIPS programming is completely logical.

Learning from the assignment and use of MIPS in real-world scenarios:

So this assignment focuses on the basics of the introduction of the MIPS programming, where I have learned the different registers, where some of them are for special purposes while others are for general purposes, and also learn the syntax involved in MIPS, like how to access the registers value stored in them, and also what are the different instructions the MIPS have. So this assignment has some basic learning, like checking the simple operations on the strings, which might have an impact on larger codes where I have to carry out more computations regarding strings.

Resources Used:

- 1) <https://courses.missouristate.edu/kenvollmar/mars/download.htm>
- 2) <https://student.cs.uwaterloo.ca/~cs241/mips/mipsasm.html>
- 3) <https://learnxinyminutes.com/docs/mips/>
