

LAB ASSIGNMENT-04

# 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.

## Task 1: Nth Catalan Number

In this task, we have been given a number  $n$  by the user, and we have to find the  $n$ th catalan number. Catalan numbers are a sequence of natural numbers that appear in various combinatorial problems.

### BreakDown of the code:

#### 1. Prompt and Input

- a) It starts by printing a prompt asking for the value of  $n$  to calculate the  $n$ th Catalan number using a syscall (`li $v0, 4` to print, `li $v0, 5` to read).
- b) The input number  $n$  is stored in register `$a0`.

#### 2. Catalan Functions

- a) The program calls the `catalan` function to compute the  $n$ th Catalan number.
- b) If  $n$  is less than 2, the function directly returns 1, which corresponds to the base case (Catalan numbers for `C(0)` and `C(1)` are 1).

#### 3. Recursive Case

- a) For  $n \geq 2$ , the recursion is implemented:
  - 1) The function recursively computes `C(n-1)`.
  - 2) Then, it performs some arithmetic operations to compute the  $n$ th Catalan number using the recursive formula:  
$$C(n) = (2(2n-1) * C(n-1)) / n+1$$
  - 3) Multiplication (`mult`), division (`div`), and bitwise shifts (`sll`) are used to calculate this formula.

#### 4. Output

- a) After computing the Catalan number, it prints the result and exits.

## Task1 Output

```
Enter n to calculate the nth Catalan number: 1
The nth Catalan number is: 1

-- program is finished running --

Enter n to calculate the nth Catalan number: 2
The nth Catalan number is: 2

-- program is finished running --

Enter n to calculate the nth Catalan number: 0
The nth Catalan number is: 1

-- program is finished running --

Enter n to calculate the nth Catalan number: 5
The nth Catalan number is: 42

-- program is finished running --
```

**Subtask 2:** Given  $x$ , find  $e^x$ . Here is the breakdown of the code:

### 1) Prompt and Input

- a) The program begins by prompting the user to enter the value of  $x$ .
  - 1) It loads the address of the string `prompt_x` into register `$a0`.
  - 2) It uses `li $v0, 4` to indicate a print string syscall, followed by `syscall` to execute it.
- b) After displaying the prompt, it reads a floating-point number from the user:
  - 1) It sets `li $v0, 6` to indicate a read float syscall and executes it with `syscall`.
  - 2) The entered value is stored in the floating-point register `$f0`.
- c) Next, the program prompts for the number of terms  $n$  to be used in the approximation:

- 1) Similar to the first prompt, it loads the address of `prompt_n` into `$a0`, prints it, and then reads an integer input using `li $v0, 5`.
- 2) The integer value is stored in register `$t0`.

## 2) Exponential Function

- a) The program calls the `exp` function to compute the approximation of `pow(e,x)`.
  - 1) It uses `jal exp` to jump to the function.
- b) Inside the `exp` function:
  - 1) It initializes two floating-point registers: `$f2` (to hold each term of the series) and `$f4` (to accumulate the sum), both set to 1.0 using `l.s.`

## 3) Loops for series expansion

- a) The loop begins with initializing a counter in register `$t1` set to 1:
  - 1) This counter keeps track of how many terms have been processed.
- b) The loop continues until `$t1` exceeds `n`.
  - 1) Within each iteration:

It calculates the current term of the series using  $x^k/k!$
- c) After updating for each term, it increments `$t1` and jumps back to continue looping until all terms are processed.

## 4) Output

- a) Once all terms have been processed:
  - 1) The final result (approximation of  $e^x$ ) is moved from `$f4` back to `$f0`.
- b) The program then prints out the result:
  - 1) It loads `result_msg` into `$a0` and prints it using `syscall`.
  - 2) Finally, it moves the computed approximation from `$f0` into `$f12`, and uses `li $v0, 2` to print this floating-point number.
- c) The program concludes by exiting cleanly with `li $v0, 10`, which indicates termination.

## SubTask 2 Output

```
Enter the value of x: 1
Enter the number of terms n: 10
The approximation of e^x is: 2.718282
-- program is finished running --

Enter the value of x: 2
Enter the number of terms n: 10
The approximation of e^x is: 7.388995
-- program is finished running --


Enter the value of x: 3
Enter the number of terms n: 10
The approximation of e^x is: 20.079666
-- program is finished running --

Enter the value of x: 4
Enter the number of terms n: 10
The approximation of e^x is: 54.443104
-- 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/>