**MIPS Assembly Calculator Project Report**

**Course:** COMPUTER ORGANIZATION & ASSEMBLY LANGUAGE

**Student Name:** Avinash Talreja

**Institution:** Iqra University, Karachi

**Date:** June 4, 2025

**Contents**

[1 Introduction 6](#_Toc2551)

[1.1 Opening Statement/Overview 6](#_Toc2552)

[1.2 Rationale for Topic Selection 6](#_Toc2553)

[1.3 Aim of the Assignment 6](#_Toc2554)

[1.4 Topic Scope and Definitions 6](#_Toc2555)

[1.5 Outline of Structure 6](#_Toc2556)

[2 Main Body 6](#_Toc2557)

[2.1 Problem Breakdown 6](#_Toc2558)

[2.2 Methodology 6](#_Toc2559)

[2.3 Implementation and Design 6](#_Toc2560)

[2.4 Evaluation 7](#_Toc2561)

[2.5 Discussion 7](#_Toc2562)

[3 Conclusion 7](#_Toc2563)

**1. Introduction**

**1.1 Opening Statement/Overview**

This project presents a console-based calculator implemented in MIPS assembly language. The calculator supports both integer and floating-point arithmetic operations, providing users with a menu-driven interface to perform a variety of mathematical and logical computations. The program demonstrates low-level programming skills and efficient use of MIPS instructions for arithmetic, control flow, and input/output handling.

**1.2 Rationale for Topic Selection**

The choice to develop a calculator in assembly language stems from the educational value of understanding how high-level arithmetic operations are translated into machine-level instructions. It highlights fundamental computer architecture concepts such as register management, branching, syscall interface, and floating-point operations. Moreover, calculators are a classic example of interactive programs that combine user input, computation, and output, making it an ideal project for demonstrating practical assembly programming skills.

**1.3 Aim of the Assignment**

The primary goal of this project is to design and implement a fully functional calculator in MIPS assembly that supports a comprehensive set of integer and floating-point operations. The project aims to reinforce understanding of assembly language programming, system calls for I/O, arithmetic and logical operations, error handling, and modular code design in a resource-constrained environment.

**1.4 Topic Scope and Definitions**

Scope: The calculator supports integer operations such as addition, subtraction, multiplication, division, modulus, power, square, square root, factorial, minimum, maximum, and bitwise logical operations. Floating-point operations include addition, subtraction, multiplication, division, power (with non-negative integer exponents), square root, minimum, and maximum.

**Definitions:**

**Syscall**: A system call interface used for input/output operations.

**Floating Point**: Representation and operations on real numbers using IEEE 754 standard.

**Bitwise Operations:** Logical operations applied to individual bits of integer operands.

**Boundary Conditions:** The calculator handles invalid inputs gracefully, including division by zero, negative inputs for factorial and square root, and unsupported negative exponents for floating-point power.

**1.5 Outline of Structure**

This report is organized into three main sections:

**Main Body:** Detailed analysis of the problem, methodology, implementation, and evaluation.

**Conclusion**: Summary of findings, reflections on the project outcomes, and recommendations for future improvements.

**2. Main Body**

**2.1 Problem Breakdown:**

The problem involves creating a user-interactive calculator that can perform a variety of arithmetic and logical operations on both integers and floating-point numbers. The challenge includes:

* Designing a menu-driven interface for mode and operation selection.
* Handling input and output via MIPS syscalls.
* Implementing arithmetic operations using MIPS instructions.
* Managing single and dual operand operations.
* Providing error detection and handling for invalid operations.
* Displaying clear and informative output including operation names and operands.

**2.2 Methodology:**

The project employs the following technologies and approaches:

**MIPS Assembly Language:** For low-level programming and direct hardware control.

**Syscalls:** Used for console input/output operations.

**Control Flow Instructions**: Branching (beq, bnez, etc.) and loops to manage menus and computations.

**Registers**: Utilized to store operands, results, and temporary values.

**Floating Point Unit (FPU):** Used for floating-point arithmetic operations.

**String Handling:** Null-terminated strings for menus and operation names, with traversal logic to print operation labels.

**2.3 Implementation and Design:**

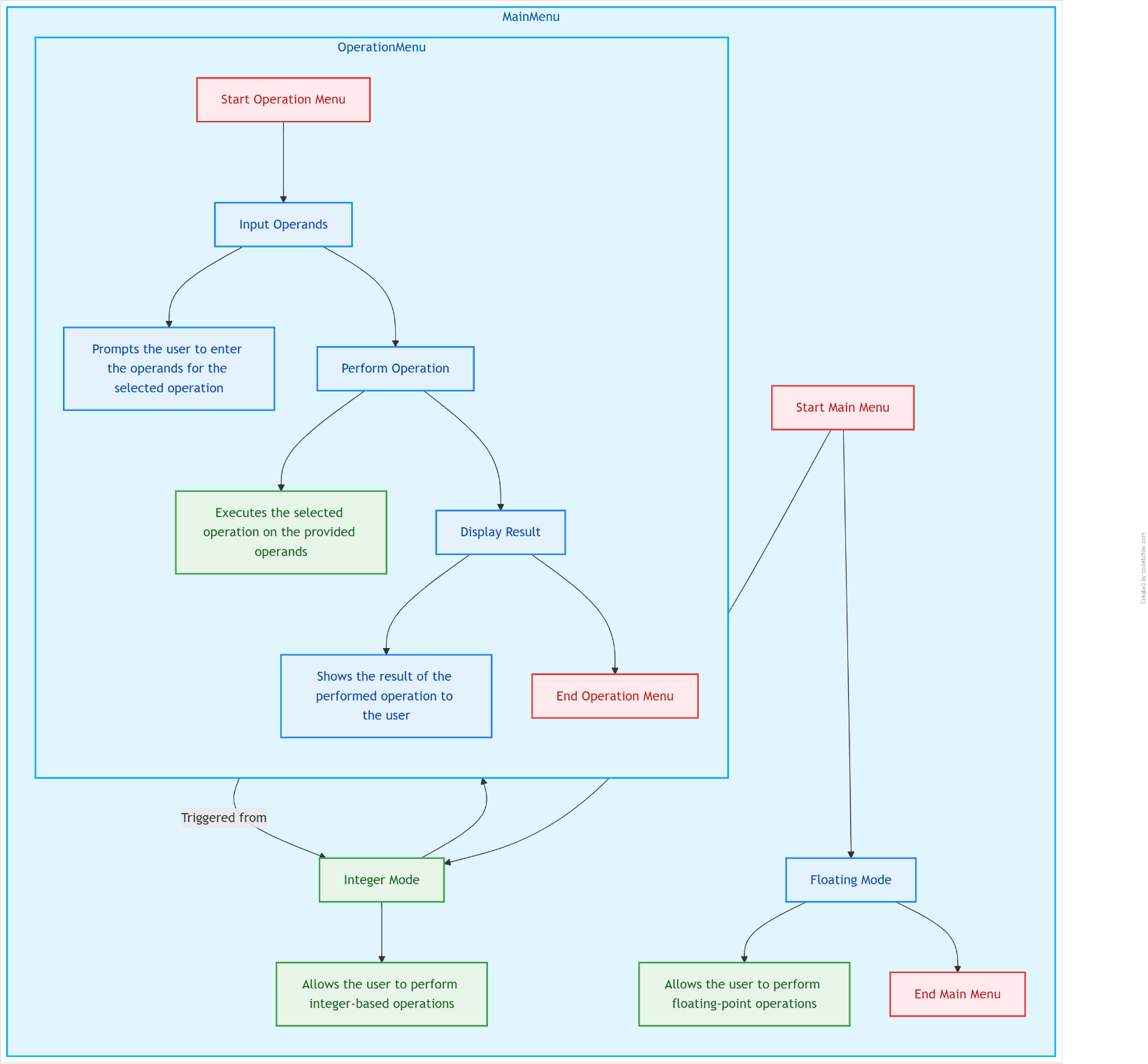
The program is divided into two main modes: integer and floating-point. Each mode presents a menu to the user, takes input for operation choice and operands, performs the computation, and displays results.

**Menu System:** Implemented with loops and conditional branching to allow repeated operations until the user exits or switches modes.

**Operation Dispatch:** Uses conditional branches to jump to specific subroutines based on user choice.

**Error Handling**: Checks for invalid inputs such as division by zero and negative inputs for factorial and square root.

**Result Display:** Prints operation names by traversing concatenated null-terminated strings, then prints operands and results.



**2.4 Evaluation**

Testing: The calculator was tested with various valid and invalid inputs to ensure correctness and robustness.

**Performance**: Operations are performed efficiently using native MIPS instructions and loops for iterative calculations.

**Accuracy**: Integer operations provide exact results; floating-point operations rely on IEEE 754 standard precision.

**Error Handling**: Proper messages are displayed for division by zero, negative square roots, negative factorials, and unsupported negative exponents.

**User Experience:** The menu system is intuitive, allowing users to switch modes or exit gracefully.

**2.5 Discussion**

**Results Interpretation:** The calculator correctly performs all specified operations and handles errors gracefully.

**Limitations**:

* Floating-point power operation supports only non-negative integer exponents.
* No support for negative shifts or floating-point factorial.
* Overflow detection for power and factorial operations is not implemented.

**Challenges:**

* Managing string traversal for operation names in assembly.
* Handling floating-point operations and conversions between integer and float.
* Designing a user-friendly interface within the constraints of syscall I/O.

**3. Conclusion**

The MIPS assembly calculator project successfully demonstrates the implementation of a versatile calculator supporting both integer and floating-point operations. It highlights key assembly programming concepts such as syscall usage, arithmetic/logical operations, control flow, and error handling. While the program meets its goals, future improvements could include support for more complex operations, enhanced error detection (e.g., overflow), and a more sophisticated user interface. This project provides a solid foundation for understanding low-level programming and system interaction.

**References**

* COMPUTER ORGANIZATION & ASSEMBLY LANGUAGE MANUAL
* CHATGPT, BLACKBOX

**Appendix A – Sample Code**

.data

# Menus

mainMenu: .asciiz "\n--- Calculator ---\n1. Integer Mode\n2. Floating Point Mode\n0. Exit\nChoose mode: "

intMenu: .asciiz "\nInteger Operations:\n 1.Add 2.Sub 3.Mul 4.Div 5.Mod 6.Pow 7.Square 8.Sqrt 9.Fact\n10.Min 11.Max 12.AND 13.OR 14.XOR 15.NOT 16.Shift Left 17\nChoose operation: "

floatMenu: .asciiz "\nFloating Operations:\n 1.Add 2.Sub 3.Mul 4.Div 5.Pow 6.Sqrt 7.Min 8.Max\n0.Back\nChoose operation: "

# Inputs prompts

inputInt1: .asciiz "Enter first integer: "

inputInt2: .asciiz "Enter second integer: "

inputFloat1: .asciiz "Enter first float: "

inputFloat2: .asciiz "Enter second float: "

# Messages

resultMsg: .asciiz "Result: "

invalidMsg: .asciiz "Invalid choice. Try again.\n"

zeroDivMsg: .asciiz "Error: Division by zero.\n"

negSqrtMsg: .asciiz "Error: Square root of negative number.\n"

negFactMsg: .asciiz "Error: Factorial of negative number.\n"

negPowerMsg: .asciiz "Error: Negative exponent not supported for power.\n"

newline: .asciiz "\n"

comma: .asciiz ", "

opLabel: .asciiz "[Operation: "

closeBracket: .asciiz "]\n"

operandLabel: .asciiz "Operands: "

# Operation names for printing (null terminated strings)

intOpNames: .asciiz "Add\0Sub\0Mul\0Div\0Mod\0Pow\0Square\0Sqrt\0Fact\0Min\0Max\0AND\0OR\0XOR\0NOT\0Shift Left\0Shift Right"

floatOpNames: .asciiz "Add\0Sub\0Mul\0Div\0Pow\0Sqrt\0Min\0Max"

.text

.globl main

main:

# Main mode selection loop

main\_mode\_loop:

li $v0,4

la $a0, mainMenu

syscall

li $v0,5

syscall

move $t0, $v0 # mode choice

beq $t0, 0, exit\_program

beq $t0, 1, int\_mode

beq $t0, 2, float\_mode

# invalid choice retry

li $v0,4

la $a0, invalidMsg

syscall

j main\_mode\_loop

##############################################################

# Integer mode

##############################################################

int\_mode:

int\_menu\_loop:

li $v0,4

la $a0, intMenu

syscall

li $v0,5

syscall

move $t1, $v0 # operation choice

beq $t1, 0, main\_mode\_loop

# Read first integer operand

li $v0,4

la $a0, inputInt1

syscall

li $v0,5

syscall

move $t2, $v0

# For single-operand operations: 7 Square, 8 Sqrt, 9 Fact, 15 NOT

li $t9, 7

li $t8, 8

li $t7, 9

li $t6, 15

beq $t1, $t9, int\_skip\_second\_op

beq $t1, $t8, int\_skip\_second\_op

beq $t1, $t7, int\_skip\_second\_op

beq $t1, $t6, int\_skip\_second\_op

# Read second integer operand

li $v0,4

la $a0, inputInt2

syscall

li $v0,5

syscall

move $t3, $v0

int\_skip\_second\_op:

move $s0, $t1 # Save operation choice

# Dispatch integer operations via jal

li $t4,1

beq $t1, $t4, int\_add

li $t4,2

beq $t1, $t4, int\_sub

li $t4,3

beq $t1, $t4, int\_mul

li $t4,4

beq $t1, $t4, int\_div

li $t4,5

beq $t1, $t4, int\_mod

li $t4,6

beq $t1, $t4, int\_pow

li $t4,7

beq $t1, $t4, int\_square

li $t4,8

beq $t1, $t4, int\_sqrt

li $t4,9

beq $t1, $t4, int\_fact

li $t4,10

beq $t1, $t4, int\_min

li $t4,11

beq $t1, $t4, int\_max

li $t4,12

beq $t1, $t4, int\_and

li $t4,13

beq $t1, $t4, int\_or

li $t4,14

beq $t1, $t4, int\_xor

li $t4,15

beq $t1, $t4, int\_not

li $t4,16

# Invalid choice

li $v0,4

la $a0, invalidMsg

syscall

j int\_menu\_loop

# Integer operations subroutines

int\_add:

addu $t4, $t2, $t3

j int\_print\_result

int\_sub:

subu $t4, $t2, $t3

j int\_print\_result

int\_mul:

mul $t4, $t2, $t3

j int\_print\_result

int\_div:

beq $t3, $zero, int\_div\_zero\_error

div $t2, $t3

mflo $t4

j int\_print\_result

int\_mod:

beq $t3, $zero, int\_div\_zero\_error

div $t2, $t3

mfhi $t4

j int\_print\_result

int\_pow:

# power with overflow check omitted for brevity, assume small powers

li $t4,1

move $t5, $zero

int\_pow\_loop:

beq $t5, $t3, int\_pow\_done

mul $t4, $t4, $t2

addi $t5, $t5, 1

j int\_pow\_loop

int\_pow\_done:

j int\_print\_result

int\_square:

mul $t4, $t2, $t2

j int\_print\_result

int\_sqrt:

bltz $t2, int\_neg\_sqrt\_error

li $t4,0

li $t5,0

int\_sqrt\_loop:

mul $t6, $t5, $t5

bgt $t6, $t2, int\_sqrt\_done

move $t4, $t5

addi $t5, $t5, 1

j int\_sqrt\_loop

int\_sqrt\_done:

j int\_print\_result

int\_fact:

bltz $t2, int\_neg\_fact\_error

li $t4,1

li $t5,1

int\_fact\_loop:

bgt $t5, $t2, int\_fact\_done

mul $t4, $t4, $t5

addi $t5, $t5, 1

j int\_fact\_loop

int\_fact\_done:

j int\_print\_result

int\_min:

slt $t6, $t2, $t3

beq $t6, $zero, int\_min\_else

move $t4, $t2

j int\_print\_result

int\_min\_else:

move $t4, $t3

j int\_print\_result

int\_max:

slt $t6, $t3, $t2

beq $t6, $zero, int\_max\_else

move $t4, $t2

j int\_print\_result

int\_max\_else:

move $t4, $t3

j int\_print\_result

int\_and:

and $t4, $t2, $t3

j int\_print\_result

int\_or:

or $t4, $t2, $t3

j int\_print\_result

int\_xor:

xor $t4, $t2, $t3

j int\_print\_result

int\_not:

nor $t4, $t2, $zero

j int\_print\_result

# Integer error handlers

int\_div\_zero\_error:

li $v0,4

la $a0, zeroDivMsg

syscall

j int\_menu\_loop

int\_neg\_sqrt\_error:

li $v0,4

la $a0, negSqrtMsg

syscall

j int\_menu\_loop

int\_neg\_fact\_error:

li $v0,4

la $a0, negFactMsg

syscall

j int\_menu\_loop

# Integer print result subroutine

int\_print\_result:

# Print operation label

li $v0,4

la $a0, opLabel

syscall

# Print operation name by traversing null-terminated strings

li $t5,1

move $t6, $zero

la $t7, intOpNames

int\_find\_name\_loop:

beq $s0, $t5, int\_print\_name\_found

addi $t5, $t5, 1

int\_skip\_str\_loop:

lb $t8, 0($t7)

addi $t7, $t7, 1

bnez $t8, int\_skip\_str\_loop

j int\_find\_name\_loop

int\_print\_name\_found:

# $t7 points after null, go back to start of name

subi $t7, $t7, 1

# Back up to start of string

int\_back\_loop:

lb $t8, -1($t7)

bnez $t8, int\_back\_continue

addi $t7, $t7, 1

j int\_print\_name\_done

int\_back\_continue:

subi $t7, $t7, 1

j int\_back\_loop

int\_print\_name\_done:

move $a0, $t7

li $v0,4

syscall

li $v0,4

la $a0, closeBracket

syscall

# Print operands

li $v0,4

la $a0, operandLabel

syscall

# Print first operand

li $v0,1

move $a0, $t2

syscall

# For single operand ops skip printing second operand

li $t9,7

li $t8,8

li $t7,9

li $t6,15

beq $s0, $t9, int\_skip\_op2\_print

beq $s0, $t8, int\_skip\_op2\_print

beq $s0, $t7, int\_skip\_op2\_print

beq $s0, $t6, int\_skip\_op2\_print

# Print comma and second operand

li $v0,4

la $a0, comma

syscall

li $v0,1

move $a0, $t3

syscall

int\_skip\_op2\_print:

# Newline

li $v0,4

la $a0, newline

syscall

# Print result

li $v0,4

la $a0, resultMsg

syscall

li $v0,1

move $a0, $t4

syscall

# Newline

li $v0,4

la $a0, newline

syscall

j int\_menu\_loop

##############################################################

# Floating point mode

##############################################################

float\_mode:

float\_menu\_loop:

li $v0,4

la $a0, floatMenu

syscall

li $v0,5

syscall

move $t1, $v0 # operation choice

beq $t1, 0, main\_mode\_loop

# Read first float operand

li $v0,4

la $a0, inputFloat1

syscall

li $v0,6

syscall

mov.s $f12, $f0 # save float input

# For single operand ops: 6 sqrt only

li $t9, 6

beq $t1, $t9, float\_skip\_second\_op

# Read second float operand

li $v0,4

la $a0, inputFloat2

syscall

li $v0,6

syscall

mov.s $f13, $f0 # save float input

float\_skip\_second\_op:

move $s0, $t1 # Save operation choice

# Dispatch float operations

li $t4,1

beq $t1, $t4, float\_add

li $t4,2

beq $t1, $t4, float\_sub

li $t4,3

beq $t1, $t4, float\_mul

li $t4,4

beq $t1, $t4, float\_div

li $t4,5

beq $t1, $t4, float\_pow

li $t4,6

beq $t1, $t4, float\_sqrt

li $t4,7

beq $t1, $t4, float\_min

li $t4,8

beq $t1, $t4, float\_max

# Invalid choice

li $v0,4

la $a0, invalidMsg

syscall

j float\_menu\_loop

# Float operations subroutines

float\_add:

add.s $f0, $f12, $f13

j float\_print\_result

float\_sub:

sub.s $f0, $f12, $f13

j float\_print\_result

float\_mul:

mul.s $f0, $f12, $f13

j float\_print\_result

float\_div:

c.eq.s $f13, $f0 # Compare divisor to 0.0

bc1t float\_div\_zero\_error

div.s $f0, $f12, $f13

j float\_print\_result

float\_pow:

# Simplified: only integer exponents >=0

cvt.w.s $f14, $f13 # convert exponent float to int

mfc1 $t5, $f14 # exponent int

bltz $t5, float\_neg\_power\_error

li $t4,1

mtc1 $t4, $f2 # float 1.0 in $f2

# move base to $f4

mov.s $f4, $f12

li $t6,0

float\_pow\_loop:

beq $t6, $t5, float\_pow\_done

mul.s $f2, $f2, $f4

addi $t6, $t6, 1

j float\_pow\_loop

float\_pow\_done:

mov.s $f0, $f2

j float\_print\_result

float\_sqrt:

c.lt.s $f12, $f0 # compare base < 0.0

bc1t float\_neg\_sqrt\_error

sqrt.s $f0, $f12

j float\_print\_result

float\_min:

c.lt.s $f12, $f13

bc1t float\_min\_set\_f0

mov.s $f0, $f13

j float\_print\_result

float\_min\_set\_f0:

mov.s $f0, $f12

j float\_print\_result

float\_max:

c.lt.s $f13, $f12

bc1t float\_max\_set\_f0

mov.s $f0, $f13

j float\_print\_result

float\_max\_set\_f0:

mov.s $f0, $f12

j float\_print\_result

# Float error handlers

float\_div\_zero\_error:

li $v0,4

la $a0, zeroDivMsg

syscall

j float\_menu\_loop

float\_neg\_sqrt\_error:

li $v0,4

la $a0, negSqrtMsg

syscall

j float\_menu\_loop

float\_neg\_power\_error:

li $v0,4

la $a0, negPowerMsg

syscall

j float\_menu\_loop

# Float print result subroutine

float\_print\_result:

# Print operation label

li $v0,4

la $a0, opLabel

syscall

# Print operation name by traversing null-terminated strings

li $t5,1

la $t7, floatOpNames

float\_find\_name\_loop:

beq $s0, $t5, float\_print\_name\_found

addi $t5, $t5, 1

float\_skip\_str\_loop:

lb $t8, 0($t7)

addi $t7, $t7, 1

bnez $t8, float\_skip\_str\_loop

j float\_find\_name\_loop

float\_print\_name\_found:

subi $t7, $t7, 1

float\_back\_loop:

lb $t8, -1($t7)

bnez $t8, float\_back\_continue

addi $t7, $t7, 1

j float\_print\_name\_done

float\_back\_continue:

subi $t7, $t7, 1

j float\_back\_loop

float\_print\_name\_done:

move $a0, $t7

li $v0,4

syscall

li $v0,4

la $a0, closeBracket

syscall

# Print operands

li $v0,4

la $a0, operandLabel

syscall

# Print first float operand

mov.s $f12, $f12

mov.s $f0, $f12

li $v0,2

syscall

# For single operand ops (sqrt)

li $t9,6

beq $s0, $t9, float\_skip\_op2\_print

# Print comma and second operand

li $v0,4

la $a0, comma

syscall

mov.s $f12, $f13

mov.s $f0, $f12

li $v0,2

syscall

float\_skip\_op2\_print:

# Newline

li $v0,4

la $a0, newline

syscall

# Print result

li $v0,4

la $a0, resultMsg

syscall

mov.s $f0, $f0

li $v0,2

syscall

# Newline

li $v0,4

la $a0, newline

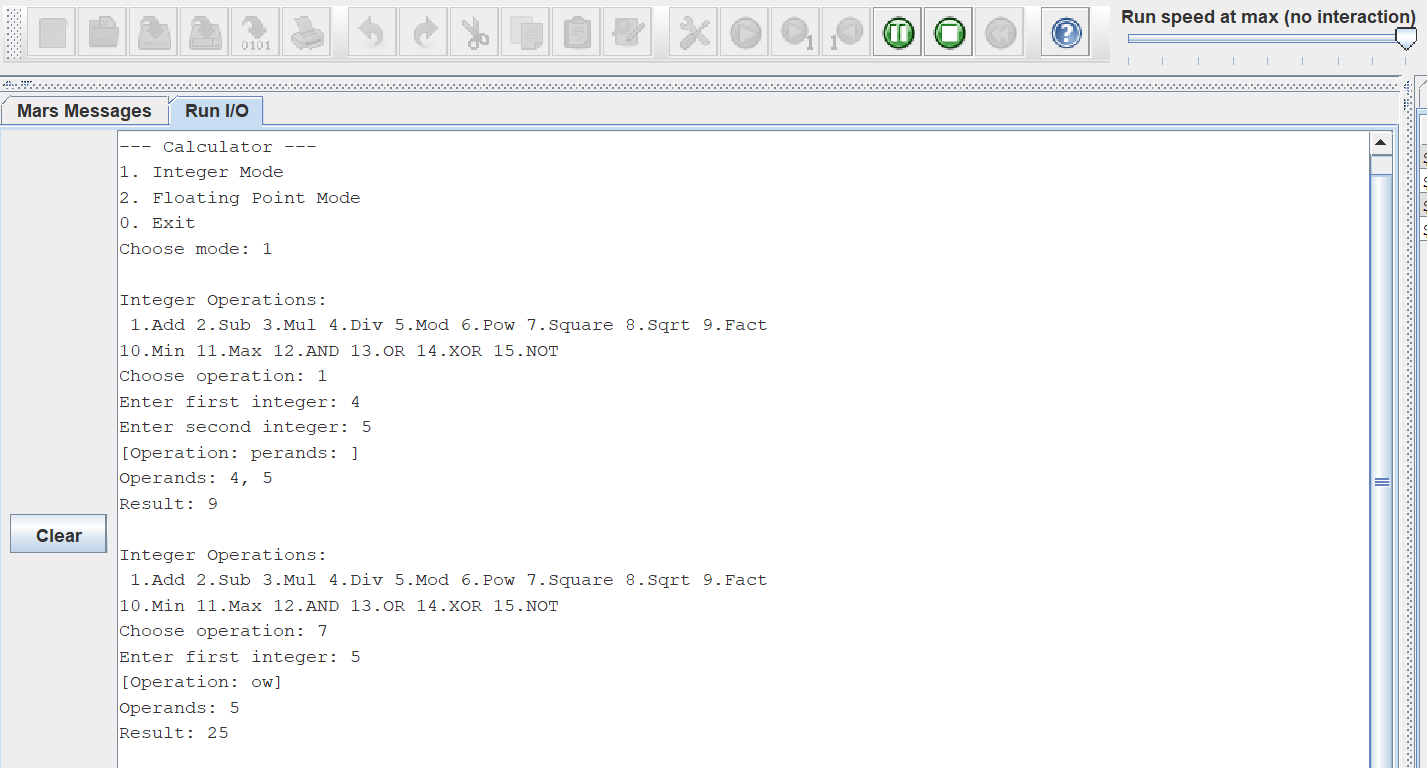
syscall

j float\_menu\_loop

exit\_program:

li $v0,10

syscall

**Appendix B – Raw Output**

A computer screen shot of a computer

AI-generated content may be incorrect.