Microprocessor Programming and Interface

Lab Assignment – 03

Name: Jangle Part

Div. 1 Grp.2 Roll Number: 22BCP083

Problem Statement:

**Q1. Consider the MIPS assembly program you have written using MARS. Explain how you would use the .data directive to declare and initialize static data variables, and how you would utilize the .text directive to define executable instructions. Provide examples of both data and instruction declarations in your explanation?**

Certainly! In MIPS assembly language, the `.data` directive is used to declare and initialize static data variables, and the `.text` directive is used to define executable instructions. Let's go through each of them with examples.

1. data Directive (Declaring Data Variables):

The `.data` section is used to declare and initialize static data variables, such as integers, strings, arrays, etc. You can use labels to name these variables and assign values to them. Here's an example:

assembly

.datamy\_integer: .word 42 # Declare a 32-bit word (4 bytes) initialized with 42

my\_string: .asciiz "Hello, MIPS!" # Declare a null-terminated string

my\_array: .word 1, 2, 3, 4, 5 # Declare an array of integers

In this example, three data variables are declared:

- `my\_integer` is initialized with the value `42`.

- `my\_string` is initialized with the string `"Hello, MIPS!"`.

- `my\_array` is initialized with an array of integers `{1, 2, 3, 4, 5}`.

2. .text Directive (Defining Executable Instructions):

The `.text` section is used to define the actual executable instructions of the program. Here, you write your MIPS assembly code to perform various tasks. Instructions are written sequentially, and you can use labels to mark locations in the code for branching or jumping.

Here's an example:

assembly

.text

main:

# Load data from memory

lw $t0, my\_integer # Load my\_integer into register $t0

# Add a constant value

li $t1, 10 # Load immediate value 10 into register $t1

add $t0, $t0, $t1 # Add $t1 to $t0 and store in $t0

# Store result back to memory

sw $t0, my\_integer # Store the updated value back to my\_intege

# Exit the program

li $v0, 10 # Load immediate value 10 (exit syscall code)

syscall

In this example, the `.text` section contains executable instructions:

- The `lw` instruction loads the value of `my\_integer` from memory into register `$t0`.

- The `li` instruction loads the immediate value `10` into register `$t1`.

- The `add` instruction adds the values of `$t0` and `$t1` and stores the result in `$t0`.

- The `sw` instruction stores the updated value back to the memory location of `my\_integer`.

- Finally, the `li` and `syscall` instructions are used to exit the program.

Remember that MIPS assembly instructions have their opcode, source and destination registers, and sometimes immediate values, all of which are essential for proper execution.

In summary, the `.data` directive is used for declaring and initializing static data variables, while the `.text` directive is used for defining executable instructions in MIPS assembly language.

**2: Imagine you are tasked with writing a MIPS assembly program in MARS to perform the following tasks: read an integer input from the user, compute the factorial of the input using a loop, and then display the factorial result. Describe step-by-step how you would accomplish this task using directives such as .data****, .****text,. word, .asciiz, and others, as necessary. Provide the complete assembly code along with explanations for each section?**

.data

input\_prompt: .asciiz "Enter an integer: "

result\_prompt: .asciiz "Factorial: "

.text

.globl main

main:

# Display input prompt

li $v0, 4  **# Load print string syscall code**

la $a0, input\_prompt **# Load address of the input\_prompt string**

syscall

**# Read integer input from user**

li $v0, 5 **# Load integer read syscall code**

syscall

move $t0, $v0  **# Store the input in $t0**

**# Initialize factorial result to 1**

li $t1, 1

**# Loop to calculate factorial**

loop:

beqz $t0, done **# If input is zero, exit loop**

mul $t1, $t1, $t0  **# Multiply factorial result by input**

sub $t0, $t0, 1 **# Decrement input by 1**

j loop

done:

# Display the factorial result

li $v0, 4  **# Load print string syscall code**

la $a0, result\_prompt **# Load address of the result\_prompt string**

syscall

li $v0, 1 **# Load integer print syscall code**

move $a0, $t1  **# Load the factorial result to be printed**

syscall

# Exit the program

li $v0, 10  **# Load exit syscall code**

Syscall

**Explanation:**

**1.data** section: This section contains the string prompts for input and result.

**2.text** section: This is where the main code resides.

**main:**: The entry point of the program.

**Display input prompt:**

•Load the syscall code **4** into **$v0** to perform a print string operation.

•Load the address of the **input\_prompt** string into **$a0**.

•Call the syscall with **syscall**.

**Read integer input from user:**

•Load the syscall code **5** into **$v0** to perform an integer read operation.

•Call the syscall with **syscall**.

•Move the input value from **$v0** to **$t0** for processing.

**Initialize factorial result:**

Load the constant **1** into **$t1**.

**Loop to calculate factorial:**

•Branch to the **done** label if the input in **$t0** is zero.

•Multiply the current factorial result in **$t1** with the input in **$t0**.

•Decrement the input in **$t0** by **1**.

•Jump back to the **loop** label.

**Display the factorial result:**

•Load the syscall code **4** into **$v0** to perform a print string operation.

•Load the address of the **result\_prompt** string into **$a0**

•Call the syscall with **syscall**.

•Load the syscall code **1** into **$v0** to perform an integer print operation.

•Move the factorial result in **$t1** to **$a0**.

•Call the syscall with **syscall**.

**Exit the program:**

•Load the syscall code **10** into **$v0** to perform an exit operation.

•Call the syscall with **syscall**.