**San Jose State University**

**Department of Computer Engineering**

**CMPE 200 Report**

**Assignment 2 Report**

**Title** MIPS Instruction Set Architecture & Programming (2)

**Semester:** Fall 2023 **Date: 09/16/2023**

**by**

**Name:** Sai Kashyap Kurella **SID: 016018925**

1. **SOURCE CODE**

# mipstest.smd

# Test the following MIPS instructions.

# add, sub, and, or, slt, addi, lw, sw, beq, j

# Assembly Description Address Machine

main: addi $2, $0, 5 # initialize $2 = 5 0 20020005

addi $3, $0, 12 # initialize $3 = 12 4 2003000c

addi $7, $3, -9 # initialize $7 = 3 8 2067fff7

or $4, $7, $2 # $4 <= 3 or 5 = 7 c 00e22025

and $5, $3, $4 # $5 <= 12 and 7 = 4 10 00642824

add $5, $5, $4 # $5 = 4 + 7 = 11 14 00a42820

beq $5, $7, end # shouldn't be taken 18 10a7000a

slt $4, $3, $4 # $4 = 12 < 7 = 0 1c 0064202a

beq $4, $0, around # should be taken 20 10800001

addi $5, $0, 0 # shouldn't execute 24 20050000

around: slt $4, $7, $2 # $4 = 3 < 5 = 1 28 00e2202a

add $7, $4, $5 # $7 = 1 + 11 = 12 2c 00853820

sub $7, $7, $2 # $7 = 12 - 5 = 7 30 00e23822

sw $7, 68($3) # [80] = 7 34 ac670044

lw $2, 80($0) # $2 = [80] = 7 38 8c020050

j end # should be taken 3c 08000011

addi $2, $0, 1 # shouldn't execute 40 20020001

end: sw $2, 84($0) # write adr 84 = 7 44 ac020054

j main # go back to beginning 48 08000c00

1. **TEST LOG**

**A table of numbers and digits

Description automatically generated with medium confidence**

1. **MARS & MIPSASM SNAPSHOTS**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

1. **DISCUSSIONS**
2. ***Initialization****:*

At the beginning of the **main** section, there is variable initialization using **addi** instructions. Registers **$2**, **$3**, and **$7** are initialized with values 5, 12, and 3, respectively.

1. ***Logical and Arithmetic Operations****:*

**or**: The **or** instruction performs a bitwise OR operation between registers **$7** and **$2**, storing the result in **$4**. In this case, it calculates 3 OR 5, resulting in 7.

**and**: The **and** instruction performs a bitwise AND operation between registers **$3** and **$4**, storing the result in **$5**. It calculates 12 AND 7, resulting in 4.

**add**: The **add** instruction adds registers **$5** and **$4**, storing the result back in **$5**. It calculates 4 + 7, resulting in 11.

**slt**: The **slt** instruction sets **$4** to 1 if **$3** is less than **$4** (12 < 7). In this case, it sets **$4** to 0.

1. ***Branching****:*

**beq**: There are two **beq** (branch on equal) instructions. The first one checks if **$5** is equal to **$7** and branches to the **end** label if true. The second one checks if **$4** is equal to 0 and branches to the **around** label if true.

1. ***Label Usage****:*

Labels like **around**, **end**, and **main** are used as targets for branching instructions, providing control flow within the program.

1. ***Memory Operations****:*

**sw**: The **sw** (store word) instruction stores the value of register **$7** into memory at address offset 68 from the address in register **$3**.

**lw**: The **lw** (load word) instruction loads a word from memory at address offset 80 from the address in register **$0** (which is typically used as a null register) into register **$2**.

1. ***Jump Instructions****:*

**j**: The **j** (jump) instruction is used to unconditionally jump to the specified label. It is used to control the program flow, both at the **end** and **main** labels.

1. ***Commented Instruction Addresses****:*

The comments indicate the memory addresses for each instruction in hexadecimal format.

1. ***Execution Flow****:*

The code has conditional branches (**beq**) and unconditional jumps (**j**), which determine the execution flow of the program. The program starts at the **main** label and loops between the **around** and **end** labels.

1. **CONCLUSION**

To sum up, this MIPS assembly code performs basic arithmetic and logic operations, utilizes branching for conditional execution, and interacts with memory through load and store operations. The specific behaviour and results of the code execution would depend on the initial values of registers and memory locations, as well as the branching conditions. The machine codes of the corresponding assembly codes and register values were observed on the MARS & MIPSASM assemblers which helped me in realizing how an assembly code is interpreted by the underlying CPU.