CS161 Lab:
Hands-on with MIPS ISA using MARS

Instructor: Elaheh Sadredini

University of California – Riverside

Introduction

In this set of assignments, you will learn how to interact with the MIPS ISA using the MARS simulator. These assignments are designed to provide hands-on experience with core computer architecture concepts including register manipulation, memory addressing, loops, and function calls. Students will analyze and modify MIPS assembly programs to deepen their understanding of how instructions operate within a CPU.

Download MARS Simulator from:

MARS Official Website: https://dpetersanderson.github.io/

Running MARS on macOS

1. Ensure you have Java installed. In Terminal, run:

```
java -version
```

2. If not installed, download and install Java JDK from AdoptOpenJDK or Oracle.

3. Open Terminal and run:

```
cd ~/Downloads
java -jar Mars4_5.jar
```

4. If blocked by Gatekeeper, go to System Preferences → Security & Privacy → General → Allow Anyway.

Running MARS on Windows

1. Ensure you have Java installed. Open Command Prompt and run:

```
java -version
```

2. If not installed, download and install Java JDK from AdoptOpenJDK or Oracle.

3. Double-click the Mars.jar file to launch. If it does not open:

  a. Open Command Prompt and navigate to the folder containing Mars.jar.

```
cd C:\Users\YourName\Downloads
```

b. Run:

```
java -jar Mars4_5.jar
```

Part 1: Registers and Arithmetic (3 points)

Goal: Understand registers and basic ALU instructions.

Instructions: In this assigment, you will modify a partially completed MIPS program in the MARS simulator to practice register operations. Begin by swapping the contents of registers `$t0` and `$t1` without using any temporary registers. Then perform the specified arithmetic operations on the swapped values. Finally, run the program in MARS, observe the register values, and verify that your results are correct.

Deliverables:

1. Final code with modifications (name the file to: “part1_Solution.asm”).
2. Screenshot of register values after execution.
3. Provide a brief explanation of the register swapping strategy you used to avoid an extra register.

Part 2: Debugging a Loop and Working with Arrays (3 points)

Goal: Understand memory addressing and loop control.

Instructions: In this assigment, you will work with a partially completed MIPS program that attempts to sum the elements of an array. Your first task is to debug and fix the loop so that it correctly traverses the array and computes the total sum of its elements. Once the loop is functioning properly, extend it to modify each array element by doubling its value and storing it back into memory. Finally, enhance the program to determine the maximum value among the updated array elements and store it in a designated memory location. After making these modifications, run the program in the MARS simulator, verify the results, and check that the sum, doubled array values, and maximum value are correctly computed and stored.

Deliverables:

1. Explain what was wrong in the original loop implementation.
2. Final code with modifications (name the file to: “part2_Solution.asm”).
3. Screenshot showing sum, max, and doubled values.

Part 3: Stack and Function Call Simulation (4 points)

Goal: Simulate function call behavior with stack and return values.

Instructions: In this assigment, you will complete a MIPS program that simulates a stack-based function call. The starter code provides a partially implemented `sum` function, which should take two input parameters, add them, and return the result. Your first task is to use the stack to save and restore any necessary registers, such as the return address (`$ra`) and any saved registers you use. After computing the sum of the two input arguments, store the result in `$v0` so it can be accessed by the caller. Then modify the `main` function to call `sum` one more time with different arguments, storing each result in memory. Finally, run the program in the MARS simulator and verify that the function calls return the correct results and the stack is properly managed.

Deliverables:

1. Completed code with at two function calls. (name the file to: “part3_Solution.asm”).
2. Screenshot showing correct return values.
3. Explanation of how stack frame works.