CSCI 4350: Computer Architecture
Project 3: Implementing a Simple MIPS Program
Due: 12/14 (Mon) 11:59pm (10 points)

1. Overview

In this project, you will make a simple program for a user to play with using functions you have implemented in previous projects. You will need to use MIPS simulator, Mars 4.5, for this project. You can find detail information about Mars 4.5 from the link below:

http://courses.missouristate.edu/KenVollmar/mars/.

2. Program Details

You will implement a program in which the goal of a user is to guess and find the numbers (and their positions) in array generated by the program. When the program begins, it generates three non-duplicated random numbers in memory (***array***), by calling `generate_random (arr[], 3)`. After generating three numbers, the program asks users to input three numbers by printing a message to the screen like below until a user finds the exact numbers and their positions

```
Ex) Input numbers:
```

A user inputs three numbers (***input***) between 1 and 9 to find the numbers in ***array*** generated by the program. Using ***input***, the program returns a hint showing if the numbers in ***input*** are included in the ***array*** and positions are the same. The program will return **Ball** and **Strike** as a hint.

Ball: indicating how many numbers in ***input*** are in ***array*** at different positions.
Strike: indicating how many numbers in ***input*** are in ***array*** at the same positions.

Let's assume that the array **[8, 1, 5]** are generated by the program at the beginning. Note, initial values in array need to be generated only once for each single play.

```
1. Input numbers: 9 2 4
```

➔ The program returns (prints to screen) "**Out**" as a result because none of numbers in input are in the array.

```
2. Input numbers: 1 2 3
```

➔ "**1 ball**" as a result because **1** is in the array but its position is different.

3. `Input numbers:` **`2 1 8`**

➔ "**1 strike 1 ball**" as a result because **1** and **8** are in the array and only **1**'s position is the same as the array.

4. `Input numbers:` **`5 8 1`**

➔ "**3 balls**" as a result because all numbers of input are in the array and all of their positions are different.

5. `Input numbers:` **`8 1 3`**

➔ "**2 strikes**" as a result because **8** and **1** are in the array and the positions are the same.

6. `Input numbers:` **`8 1 5`**

➔ "**Good Job!**" as a result.

In short, the program compares values and positions of numbers randomly generated to user's input. The program repeatedly asks a user to input numbers until it gets numbers that are exactly same with the array.

3. Implementation Details

You will need to merge all the MIPS functions you have implemented into a single MIPS program file. Please take a look at the page below to see how you can use the system calls.

https://courses.missouristate.edu/KenVollmar/MARS/Help/SyscallHelp.html

You will need to use MIPS instructions below (but not limited to) to implement the program. You can use any other MIPS instructions to complete the function.

Instruction	Example	Description
`li`	`li $t0, 1`	An integer value is loaded into a register ($t0) with 1
`la`	`la $t0, sym`	An address is loaded into $t0 with the address 'sym'
`lw`	`lw $t0, offset($t1)`	A word is loaded into $t0 from the specified address (offset + $t1)
`sw`	`sw $t0, offset($t1)`	The contents of $t0 is stored at the specified address (offset + $t1)
`add`	`add $t0, $t1, $t2`	Adds $t1 and $t2 and stores the result in $t0
`addi`	`addi $t0, $t1, 1`	Adds $t1 and a sign-extended immediate value (1) and stores the result in $t0
`sll`	`sll $t0, $t1, 4`	Shifts $t1 value left by the shift amount (4) and places the result in $t0. Zeroes are shifted in
`mul`	`mul $t0, $t1, $t2`	Multiply $t1 and $t2 and stores the result in $t0
`jal`	`jal target`	Jumps to the calculated address and stores the return address in $ra ($31)

`j`	`j    target`	Jumps to the calculated address
`beq`	`beq $t0, $t1, target`	Branches to target if $t0 and $t1 are equal
`blt`	`blt $t0, $t1, target`	Branches to target if $t0 is less than $t1
`slt`	`slt $t0, $t1, $t2`	If $t1 is less than $t2, $t0 is set to 1, 0 otherwise

You will need to show the numbers generated at first for debugging (testing) purpose. you must use the functions you implemented for previous projects. **You must use JAL and JR instructions for making function calls**.

You may want to implement the program in C first to understand how the function works clearly. You can take a look at any other MIPS code or references from Internet or book. You can discuss with your classmates for this project to get some hints. **But it is not allowed to share the solution with your classmates. I will use a tool to detect cheating.**

4. Deliverables

a. Document describing how your assembly code works. Not to exceed 1 page.
b. MIPS source code