# CSCI 4350: Computer Architecture
# Project 2: Implementing a Simple MIPS Function II
# Due: Nov/5 (Fri) 11:59pm (10 points)

## 1. Overview

In this project, you will implement a simple MIPS assembly function. 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: http://courses.missouristate.edu/KenVollmar/mars/. In this lab, you will learn about implementing and calling functions with the MIPS assembly language.

## 2. Function Details

You will implement a simple function, `generate_random (int arr[], int cnt)`. The function generates non-duplicated random numbers and stores them to the array address (`arr`). The second argument (`cnt`) indicates how many numbers need to be generated. You can assume that the generated random numbers are between 1 ~ 9 and `cnt` is the size of array (less than 10).

For function call, `generate_random (arr, 6)`

`arr[]` = {5, 2, 1, 4, 6, 3} → valid

`arr[]` = {1, 3, 4, 2, 5, 7} → valid

`arr[]` = {9, 5, 2, 3, 1, 5} → invalid ('5' is duplicated)

`arr[]` = {9, 5, 2, 3, 1} → invalid (generated numbers is 7)

`arr[]` = {9, 5, 2, 3, 1, 4, 6} → invalid (generated numbers is 5)

The `main` function will call `generate_random()` to generate non-duplicated numbers.

## 3. Implementation Details

You will need to use Mars 4.5 system calls (**41 or 42**) to generate random numbers. 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 `generate_random` function. 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 must show (display) the numbers generated after the function call. **To avoid duplicated numbers in the array, you must use the `lookup` function you implemented in the first project.**

For function calls**, you must use 1) a stack to store and load values of registers and 2) a pair of JAL and JR instructions. In addition, the $v0 register must be used for a return value.** You will lose points if you did not implement your program as requested above, even if your program works well.

You may want to implement `generate_random` 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.**

# 3. Testcase

A skeleton code (**prj2.asm**) will be provided. The code includes a `main` function and some other code for initialization. The array is hard-coded in .data section in the code,

```
arr: .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 (up to 9 numbers)
```

If you call `generate_random(arr, 9)`, the array (`arr` ) will include all numbers from 1 to 9. You will also need to change the second parameter (`cnt`) to test your code.

# 4. Deliverables

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