CSCI 4350: Computer Architecture
Project 1: Implementing a Simple MIPS Function
Due: 10/6 (Tue) 11:59pm (5 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, `lookup (int arr[], int num, int cnt)`.
The function finds the integer value (`num`) from the integer array (`arr`). The last argument (`cnt`) is used to set the last index of array to be checked. The function returns the **index of array** if `num` is found in `arr`, else -1. For example, assume an array

`arr[]` =

5	2	1	4	6	3

Return values for each function call.

1. `lookup (arr, 5, 3)` → 0 (the index of '5' in the array)
2. `lookup (arr, 1, 3)` → 2 (the index of '1' in the array)
3. `lookup (arr, 7, 6)` → -1 ('7' is not found in the array)
4. `lookup (arr, 4, 3)` → -1 (the last parameter indicates to check only first 3 values in the array)

The `main` function will call `lookup` to see if the integer array (`arr`) stores the integer value (`num`).

3. Implementation Details

3.1. Some instruction list (including pseudo-instructions)

You will need to use MIPS instructions below (but not limited to) to implement the `lookup` 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

3.2. Translating C code into MIPS code

You may want to implement `lookup` in C first to understand how the function works clearly. You will need to know how to translate loop (`for` and `while`) statement and `if` statement into MIPS code. You will also need to know how to load data in memory into registers and how to store data in registers into memory. Finishing HW2 will help you to translate C code into MIPS assembly code. Some MIPS program examples are also available on Canvas.

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. Testcase and extra credit

A skeleton code (**prj1.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 5, 3, 1
```

You will need to change the numbers and/or to increase the size of array by adding more numbers like below.

```
arr:    .word 7, 4, 2, 6, 5, 4
```

You can assume that the array stores any integer values between 0 and 9. If there is duplicated value in the array, the function will return the first index.

You will also need to change the second parameter (`num`) and the third parameter (`cnt`) to test your code.

```
li      $a1, 7    # change 7 to different number to test
li      $a2, 3    # change 3 to different number to test
```

Extra credit: The second parameter is hard-coded. If you implement the `main` function to get the second parameter from a user, you will get 1 extra credit. You will need to print a message for user to input a number to be looked up.

5. Deliverables

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