COMP 273 Assignment 3

Prepared by Prof. M. Langer

Posted: Tues. Feb. 23, 2016
Due: Wed. March 9, 2016 at 23:59

General Instructions

- If you do not yet know how to use breakpoints to read from Memory and registers and how to step through your code, then you need to learn this before you start this assignment. Otherwise, you are programming blind.

- The TAs handling this assignment are Noor (nurmemet.abudukelimu@mail.mcgill.ca) and Hasan (sh.mozafari@mail.mcgill.ca). Their office hours will be posted on the public web page.

- **Late assignments** will be accepted up to 3 days late, and will be penalized by 20 percent per day. If you submit one minute late, we reserve the right to treat this as equivalent to submitting 23 hours and 59 minutes late, etc. Verify that your submission was submitted correctly. If it was not, then your assignment will be considered late.

- **Bonus points:** If you discover ambiguities or minor errors with the assignment, please notify us immediately. If you are correct in identifying problems, you will be eligible for bonus points.

- **Plagiarism:** We encourage you to discuss the assignment with each other, but the discussion should be public in the sense that anyone should be able to listen in. The postings on facebook as in Assignments 1 and 2 are fine.

  You are not allowed to discuss the assignment with each other e.g. on facebook and mycourses, but under no circumstances should you show solution code to each other. Any strongly suspected cases of plagiarism will be reported to the Faculty of Science disciplinary officer.

- You must properly comment your code. Points will be taken off for code that is difficult to follow or not commented. Commenting your code not only helps the grader, it helps you to keep track of what your code is doing! You should also use debugging tools such as setting breakpoints, and checking the contents of registers and Memory. If you do not know how to do so, then get help.

- The programs will be tested only with valid inputs.

- **Submission Instructions:** Submit your code as a single file in a zip directory **A3.zip**. The directory must contain a file **findRankK.asm that** has only has the function findRankK that you write. Your code will be tested using a different main program that will call your function. Points will be taken off if you do not follow this specification.

Introduction:

This assignment is about function calls and recursion in MIPS. It will also teach you a nice algorithm which should complement some of what you learned in COMP 250. It will also get you wondering about the relationship between MIPS memory and how memory might be used in Java. I will say more about the latter at the end of the course.

Suppose you are given a list of N integers and an integer k from 0 to N-1, and you want to return which of the integers is the "kth smallest", that is, which of the integers would be in position k in an ordered version in this list. For example, if the list were (20, 42, 0, 31, 85, 50, 12, 61, 90, 70) and if k=2, then the returned value would be 20 since the sorted version of the list is (0, 12, 20, 31, 42, 50, 61, 70, 85, 90). You could solve this problem by sorting the list, and returning the element in position k but sorting the list involves unnecessary work.

A faster solution to this common problem uses the following algorithm, written in pseudocode. This given algorithm is recursive. There exist non-recursive versions of this algorithm, but we will implement the recursive version because I want you to understand how recursion works in MIPS.

INPUT:
L is a list of integers, with |L| elements. The elements are not necessarily distinct, that is, there may be repeats.
k is an index in 0,..,|L|-1

OUTPUT:
The kth smallest element in L, namely element that would be at index k if the list were sorted.

```
findRankK(k,L){    //  assume k in 0,…, |L|-1

if |L| == 1 then return the unique integer in L   (base case)
  else {
    pivot = L(0)
    create lists L1, L2, L3 which are the elements in L that are less
         than, equal to, and greater than the pivot, respectively
    if ( k < | L1 | ) then
         return findRankK( k, L1 )
    else if ( k < |L1|+|L2| ) then
          return pivot
    else
         return findRankK( k-|L1|-|L2|, L3)
  }
}
```

You are given Java code that implements this algorithm. You are also given MIPS starter code. Your taks is to complete the MIPS code, namely to implement `findRankK`.

Your solution must obey the usual MIPS conventions which were given in lecture 11. As the author of `findRankK`, you do not know which parent function calls it, and so you don't know which registers are being used by the parent at the time it is called. *In grading your assignment, we will test that you are using the conventions correctly. We will do so by calling your function from a main program that is different from the one which is provide*

Your MIPS function `findRankK` will have three arguments:

- $a0 - the value k
- $a1 - the starting address of the array L
- $a2 - the size of the list, i.e. | L |

The function must return the kth smallest integer in register $v0.

Your MIPS function should be similar to the given `findRankK` Java method, but there will be differences too. In MIPS, you have control over how Memory is used, but in Java you don't.

We require that your MIPS function uses the 2 * | L | consecutive words of Memory that have been allocated in the starter code. For the first call to `findRankK`, the first | L | words will hold the original list L and the second | L | words will serve as a buffer for computing the partition. For the recursive calls, you may use this region of Memory however you wish.

You may assume that the main program which calls `findRankK` does not need the list L to be preserved after the call. You can assume that this Memory block of size 2*|L| will be discarded.

Get started early and have fun !