COMP 273 Assignment 4

Prepared by Prof. M. Langer

Posted: Thurs. March 24, 2016
Due: Fri. April 8, 2016 at 23:59

General Instructions

- **TA's handling this assignment** are Ben (tzu-yang.yu@mail.mcgill.ca) and Josh (joshua.romoff@mail.mcgill.ca). Office hours and locations will be posted on the course web page.

- **Late assignments** will be accepted up to only 3 days late, and will be penalized by 20 points per day.

- **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. You are not allowed to tell each other how to do the assignment, although some help is permitted e.g. if you are stuck on a small detail. Under no circumstances should you show solutions to each other. Any strongly suspected cases of plagiarism will be reported to the Faculty of Science disciplinary officer.

- Properly comment your code. Points will be taken off for code that is poorly commented.

- The programs will be tested only with valid inputs.

- **TIP:** Be very careful using MARS pseudoinstructions. Many students had difficulties in previous assignments because they misunderstood what certain pseudoinstructions are doing. Verify how the pseudoinstruction is defined before you use it.

- Your solutions will be tested individually. This is only possible if they obey MIPS conventions.

You need to submit a zip directory containing the following files:

- **findRootRecursive.asm** and **findRootIterative.asm**
  Each file must contain your name and student number as a comment. Note that the function label is **findRoot** in both cases. We will call this function from tester code.

- **Q3.txt** or **Q3.pdf** which has your solution for question 3

- **README.txt** file that comments on any issues/problems/features of your code.

Introduction

This assignment will give you more practice with functions in MIPS. It will also give you a chance to use MIPS instructions for (single precision) floating point variables. Finally, it will give you some experience with making your code efficient, both in terms of the number of instructions as well as with the use of the cache.

The main problem is to write a MIPS functions that computes the root of a polynomial, using the *bisection method*. A recursive version of the algorithm is shown below. Note this is a *tail recursive* algorithm, which means the recursive call occurs at the end. Tail recursive algorithms can be written alternatively as an iterative algorithm using a loop. You will need to write both recursive and iterative versions.

The idea underlying the algorithm is that if f(x) is any continuous function over an interval [a,c] and f(a) and f(c) are of opposite sign, then there must exist at least one value b in the interval [a,c] such that f(b)=0. In the case that f(x) is a polynomial p(x), we say that the value b is a *root* of the polynomial. The algorithm finds one such root. (There may be many roots in a given interval. The algorithm finds one.)

```
findRoot(a,c) {                          //  assume p(a) and p(c) are of opposite sign
        b = (a + c) / 2
        if ( p(b) == 0 )                 //  b is a root of p(x)
            return b
        else if ( abs(a - c) < epsilon)  //  epsilon is a given constant
            return b
        else if ( p(a)*p(b) > 0 )        //   a and b have the same sign
             a = b
        else
             c = b

        return   findRoot(a,c)
}
```

Function calls must obey MIPS register conventions. For floats, these conventions are:

- $f0, … $f3 are used to return float values from functions
- $f4, ..., $f10 are temporary registers
- $f12,…,$f15 are used for passing float arguments to functions
- $f20,…,$f30 are save registers

Notice that these conventions are different than for integer valued arguments and returned values. However, the same principles apply: return value registers and temporary registers are not preserved on function calls so parent much save them if parent needs them, and save registers and argument registers are preserved on call so child must save and restore old values if child uses these registers.

In this assignment, we work with single precision only.

You are provided with two helper functions. The first of these evaluates a polynomial. The second is the power function which evaluates x^n.

[**ASIDE (for your interest only):** There exist more efficient methods for evaluating polynomials, e.g. Horner's method. There also exist faster methods for computing the power x^n. Here we use a O(n) method to compute power(x,n) but there is an asymptotically faster method, namely O(log n), which you may have learned in COMP 250. http://www.cim.mcgill.ca/~langer/250/11-binarysearch.pdf The algorithm described there is recursive, however, and so there are extra instructions associated with using stack which could make it more expensive than the O(n) in practice unless n is huge. [ADDED April 6: But you can write that algorithm without recursion (see Paul's April 5 posting on the discussion board).]

Data directives are used to define the following constants. The constants are given in the starter code. POLYORDER is the order of the polynomial. The COEFFICIENTS of the polynomial are integers. The INTERVAL [a,c] is given by two single precision floats (a and c). EPSILON is the error tolerance for the solution according to the 'epsilon' in the above algorithm.

```
.data
#  define the polynomial
POLYORDER:          .word    3          # Order of polynomial.
COEFFICIENTS:       .word    1, -2, -1,  2   # Integer coefficient of highest order comes first in list

#  define the interval in which we search for the root,  and the tolerance for the solution
INTERVAL:           .float   -1.53, 0.51          # Find a root between these values.
EPSILON:            .float   0.000001    # Value e in algorithm above.
```

For example, the polynomial represented here is p(x) = x^3 -2x^2 - x + 2. It has roots x = -1, 1, 2.

When we test the correctness of your code, we will change the values of these constants. Only valid constants will be tested.

The main program (starter code) calls a findRoot function that you will write which returns a root of a given polynomial p(x) in a given interval [a,c] with precision EPSILON, namely the absolute difference between of the returned value from the true root must be at most EPSILON.

The polynomial p(x) is evaluated using a helper function evaluate(x) which in turn calls the power function mentioned earlier.

Question 1 (50 points):

Write a recursive MIPS function findRoot that implements the above algorithm.

Your function may call the helper function evaluate. Note that evaluate has access to the names POLYORDER, COEFFICIENTS but the function that you write does not. Rather we are insist that you pass only the arguments a, c, and epsilon. The two arguments a, c, epsilon of findRoot are single precision floats. They are passed in registers $f12, $f13, $f14 respectively. The root is returned in register $f0.

Submit your recursive solution as a file **findRootRecursive.asm** (no starter code).

Question 2 (30 points):

Write a non-recursive function function which replacing the tail recursion with a loop. The other requirements from Q1 are the same.

Note that your usage of the stack [MODIFED Apr 5] may ~~should~~ be quite different in this solution.

Submit your iterative solution as a file **findRootIterative.asm** (no starter code).

Question 3 (20 points)

a) Use the Tools -> Data Cache Simulator in MARS to compare the cache hit rates for your solutions in Question 1 and 2. For both solutions, fix the size of the cache at 1024 bytes and examine performance as you vary the block size and number of blocks (for fixed cache size). Use the Direct Mapping method only.

   Briefly discuss your findings. Are they consistent with your expectations? Why?

b) Use Tool -> Instruction Statistics -> Total to compare the number of instructions for the recursive versions iterative versions used *at runtime*. Briefly summarize your findings, and whether they are consistent with your expectations.

Submit your discussion as a text file **Q3.txt**.

Note that for this question, you need to give numerical values. These values must be obtained using the polynomial and interval given in this document and in the starter code. However you should also test you code using other values to be sure it is correct.