CS 351
COMPUTER ARCHITECTURE
Fall 2019

ASSIGNMENT 2

Due Date: *Sunday, December 1st, 2019, 23:59*

Assignment Submission: *Turn in your assignment by the due date through LMS.* ***No late submissions will be accepted.*** *For each question, you should create a file and name the file as answer<queston number>.asm. Put all files under a folder (name the folder as <your first name>_<your last name>_ assignment1). Zip this folder and upload the zip file to LMS. Name the zip file as <your first name>_<your last name>_ assignment2.*

All work in the questions must be your own; you must neither copy from nor provide assistance to anybody else. If you need guidance for any question, talk to the instructor or teaching assistants.

In this assignment, you will write and simulate MIPS assembly code to solve various problems. You are going to use MARS MIPS simulator. Please download and install MARS version 4.5 from the following link:

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

In order to run your code, you should first assemble it via clicking Run→Assemble (or press F3) then you can run by clicking this symbol  in the toolbar.

Note: In this assignment, you CANNOT use any MIPS instructions that we didn't cover in the lectures. Also, DO NOT use pseudo-instructions such as blt

QUESTION 1

Assume that a list ***A*** (array ***A***) of integer numbers (each integer in 4-bytes) have been stored in memory. Please write a MIPS assembly code that finds the median of the list.

To be able to find the median of the list, first you will sort the list. Sort the list with SelectionSort algorithm. The median of the list A is A[len(A)/2] after sorting the list.

You can find the description and the Python implementation of the Selection Sort algorithm in the following link:

https://www.geeksforgeeks.org/selection-sort/

As you will notice, there are variables ***i, j*** and ***min_idx*** in Python implementation. In your MIPS code, hold the value of ***i*** in register $t0, the value of ***j*** in register $t1 and the value of ***min_idx*** in register $t2. The value of the *median* should be kept in register $s2.

Note: For this question, start with the code (***answer1.asm***) that has been provided to you with the assignment. This code defines a list, stores the base address of the list to the register ***$s0*** and the size of the list to the register ***$s1***. The values of list A in this code are given as examples. Your code should work for any list of integers. If the size of the list is changed, the value of ***$s1*** needs to be updated manually.

QUESTION 2

The following iterative sequence is defined for positive integers:

n → n/2 (n is even)

n → 3n + 1 (n is odd)

You start with a positive integer and iteratively apply these rules until reaching to 1. For example, if you start with 13, you generate the following sequence:

13 → 40 → 20 → 10 → 5 → 16 → 8 → 4 → 2 → 1

It can be seen that this sequence takes 9 steps to go from 13 to 1. Although it has not been proved yet (Collatz Problem), it is thought that all numbers finish at 1.

You are given the following Python code that finds the number of steps needed to reach to 1 for each element of the list A and finds the maximum of these steps. Please write the corresponding MIPS assembly code. Make sure that your MISP implementation also has a *find_steps* function.

```
def find_steps(x):
        step = 0 # number of steps. Hold this value in $t0
        while x != 1:
                if x % 2 == 0:
                        x = x/2
                else:
                        x = (3*x) + 1
                step += 1
        return step

A= [2,5,7,8,12,17,21]
max = 0   # maximum of steps required. Hold this value in $s2
for i in A:
        temp = find_steps(i)
        if temp > max:
                max = temp
```

Note: For this question, start with the code (***answer2.asm***) that has been provided to you with the assignment. This code defines a list, stores the base address of the list to the register ***$s0*** and the size of the list to the register ***$s1***. The values of list A in this code are given as examples. Your code should work for any list of integers. If the size of the list is changed, the value of ***$s1*** needs to be updated manually.

You should follow proper function development rules: Use $a0-3 registers to pass parameters to the function and $v0-1 registers to return values from the function. If you use $s0-7 registers within the function, make sure to restore them to the original values at the end of the function.