CS/ECE 250: Homework 2

Due: 9/28/2015 at 5 pm through Sakai. This homework is to be done individually

Problem 1 [30 points]: Instruction Sets

a. **[9 points] Translate the following binary representation into MIPS instructions. See Appendix A.10 in "Computer Organization and Design" 5th edition for encodings.**

   i. [3 points] 000000 00010 00011 00100 00000 10 0000

   ii. [3 points] 100011 00110 00101 0000 0000 0000 0111

   iii. [3 points] 000100 00010 00111 0000 0000 0000 0110

b. **[9 points] What are the binary representations of the following instructions?**

   Note: $t0, $t1, $t2 correspond to physical register $8, $9, $10, respectively. The address of the label "loop" is 16.

   i. [3 points] and $t0, $t1, $t2

   ii. [4 points] sw $t0, 15($t1)

   iii. [4 points] bne $t0, $t1, loop

c. **[12 points] Write the following program using three different ISAs: Accumulator, Stack, and General Purpose Register. The deliverable for this problem is written code that will not be executed.**

((1*a*b)+2)*3

The source operands and destination operands for these ISAs are provided below:

1. Accumulator:
   Operations: load, store, clear, add, sub, mult, div
   (e.g., add A // acc = acc + mem[A])

2. Stack
   Operations: push, pop, add, sub, mult, div
   (e.g., push A // stack[TOS++] = mem[B])
   (e.g., add // stack[ToS++] = stack[--TOS] + stack[--TOS])

3. General Purpose Registers
   Operations with 3 registers: add, sub, mult, div…
   (e.g., add A, B, C // A = B + C)

Problem 2 [30 Points]: MIPS and Recursion

The goal of this program is to get experience with MIPS calling conventions (passing arguments, manipulating the stack, etc.). Write a MIPS program called *netid_recurse.s* that implements a recursive version of `strlen`, where `strlen` is a standard C library function for calculating the length of a null terminated string.

Your code should start by asking for input from the user. The code will read the string input using syscall. After your code runs and finds the string length, it will print the length to the console using syscall. An example of what the console will display after running your program.

```
Enter a string:
racecar
7
```

Your code must be recursive, and it must follow proper MIPS calling conventions. **Code that is not recursive or that does not follow MIPS calling conventions will be severely penalized!**

Hint: An implementation of non-recursive `strlen` is provided.

```
strlen:
  add $v0, $zero, $zero          # initialize length to 0
loop:
  lbu $t0, 0($a0)                # load one character
  addi $a0, $a0, 1               # point to next char
  addi $v0, $v0, 1               # increment length
  bne $t0, $zero, loop           # repeat if not null yet
s_end:
  addi $v0, $v0, -1              # do not count the null
  j $ra


#implement your recursive strlen in MIPS Assembly
```

Problem 3 [40 Points]: MIPS and Most Valuable Player (MVP)

The goal of this program is get experience reading various input types, structuring a larger program, and using MIPS floating point instructions. You will need to do a little research to understand MIPS floating point instructions. Read the SPIM document, http://spimsimulator.sourceforge.net/HP_AppA.pdf. Page A-73 lists floating-point instructions but you should read more.

Write a MIPS program called *netid_mvp.s* , which reads input typed by the user. The program will read in lines of input as strings.

That is, each line will be read in as its own input using spim's syscall support for reading in inputs of different types (formats). The input is a series of player stats. Each player reports 4 stats distributed across 4 input lines. The first input line is the player's last name (string), the second input line is his points per game (float), the third input is his rebounds per game (float) and the fourth input line is his turnovers per game (float). The very last line of input should be "DONE".

Example input is:

James

27.8

8.1

3.5

Curry

29.5

4.6

6.7

Durant

30.1

8.0

4.5

DONE

Important: There is no constraint on the number of players. Your program cannot simply allocate memory space for a hard-coded number of records (e.g., 10 players). Instead, your program must accommodate an arbitrary number of players by allocating space for each player's data on the heap. Code that does not accommodate an arbitrary number of players will be penalized.

Furthermore, the program may not read the entire input to determine the number of players and then perform a single dynamic allocation of heap space. Instead, your program must dynamically allocate memory as it reads the file. To perform dynamic memory allocation in MIPS assembly, please see this resource:

http://chortle.ccsu.edu/assemblytutorial/Chapter-33/ass33_4.html

Outputs: First, your program should output a number of lines equal to the number of players, and each line is the player's name and his value. The value of a player is define as:

(points-per-game + rebounds-per-game) / turnovers-per-game

The lines should be **sorted in descending order of player values**.

Finally, your program should output **the name of MVP (Most Valuable Player) as the last line**. MVP is the player whose value is greater than all other players.

For the input example, the output is:

```
James 10.26
Durant  8.47
Curry   5.09
James
```

Note 0: The input will specify DONE in caps as illustrated. You may assume an upper bound on a player's name length (e.g., 60 characters). The output statistics (e.g., player value) need not be rounded.

Note 1: You need not free memory in MIPS.

Deliverables:

The following are expected to be submitted using Sakai:

1) A single PDF file with your answers for the first question. The solution for Problem 1(c) is written code but should be included in the PDF.

2) Source code files for all MIPS Assembly programming questions (netid_recurse.s and netid_mvp.s)

3) A README plain text or PDF file that contains valuable information like how to run the code, additional comments, etc.

Note 0: Work submitted must be your own. Plagiarism (including finding solutions on the internet) will not be tolerated and we expect you to comply with the Duke Community Standard.

Note 1: If any part of the assignment is submitted after the due date the entire assignment will be considered late.