CS224 Lab #2

MIPS Floating Point Instructions and Programs

Dates: Section 1 Monday 22 February 08:40-12:30
Section 2 Friday 26 February 13:40-17:30
Section 3 Tuesday 23 February 08:40-12:30

Purpose: In this lab you will study floating point number instructions in MIPS "coprocessor 1"and will write a program to do matrix operations on floating point numbers.

Preparation: Study matrix operations from math books, or the internet. Know how to perform all common matrix operations before coming to the lab. Write all the MIPS code you can in advance; test it (even using "dummy" routines) to be sure that those parts of the Lab 2 assignment are working well.

Part 1. Exercising MIPS floating point instructions (30 points)

1. Write a simple program (both .text and .data segments) which will prompt user to enter a set of floating point numbers. These numbers will be stored in an array in the memory (***The TA will specify on the board where in memory to locate this value***.) Your program should display the array content on the console window. Additionally, your program should include the following MIPS floating point operations. You may put them in any order, and may use any registers and memory and small constants (immediate values) as needed.

Arithmetic Instructions

```
add.s $f0, $f1, $f2     #$f0 := $f1 + $f2
sub.s $f0, $f1, $f2     #$f0 := $f1 - $f2
mul.s $f0, $f1, $f2     #$f0 := $f1 * $f2
div.s $f0, $f1, $f2     #$f0 := $f1 / $f2
abs.s $f0, $f1          #$f0 := |$f1|
neg.s $f0, $f1          #$f0 := -$f1
```

Memory Transfer Instructions

```
l.s $f0, 100($t2)       # load word into $f0 from address $t2+100
s.s $f0, 100($t2)       #store word from $f0 into address $t2+100
```

Data Movement between registers

```
mov.s $f0, $f2          #move between FP registers
mfc1 $t1, $f2           #move from FP registers (no conversion)
mtc1 $t1, $f2           #move to FP registers (no conversion)
```

Data conversion

```
cvt.w.s $f2, $f4        #convert from single precision FP to integer
cvt.s.w $f2, $f4        #convert from integer to single precision FP
```

2. Use MARS (editor, assembler, and simulator for execution) to make your program run. Run your program in single-step mode, advancing one instruction at a time under your control. As your program is executing 1 step at a time, look at the values in the registers at each step, including the PC register, and also at the relevant memory locations, to follow and verify its execution. Does it behave as you expected? Why or why not? When it is working correctly, call the TA or Tutor, show your program execution in single-step mode, and explain what the values are as it runs. Now save this MIPS source code, you will need it later. Save it with a filename that includes your name: name_surname_MIPS.txt.

Part 2. MIPS floating point routines (30 points)

3. Similar to first part, write a simple program (both .text and .data segments) which will prompt user to enter a set of floating point numbers. These numbers will be stored in an array in the memory (***The TA will specify on the board where in memory to locate this value***.) Your program should display the array content on the console window. Then your program will ask the user for several options which will perform a set of tasks (***The TA will specify the set of tasks to perform.).*** For example, you may find the minimum among the set of numbers.

After performing the specified task your program should print out the result array content again. Consult the MIPS Green Card for assembly instructions for floating point numbers. Here is one instruction pair that you might use:

```
c.eq.s   $f2, $f4
bc1t   Label1
```

Here if the value in the register $f2 is equal to the value in $f4, it jumps to the Label1. If it should jump when the value in the register $f2 is NOT equal to the value in $f4, then it should be:

```
c.eq.s   $f2, $f4
bc1f   Label1
```

To load a single precision floating point number (instead of lw for an integer), you might use:

```
l.s   $f12, 0($t0)
```

To store a single precision floating point number (instead of sw for an integer), you might use:

```
s.s   $f12, 0($t0)
```

To assign a constant floating point number (instead of li for an integer), you might use:

```
li.s    $f12, 123.45
```

To copy a floating point number from one register to another (instead of move for an integer), you might use:

```
mov.s   $f10, $f12
```

The following shows the syscall numbers needed for this assignment.

System Call Number	System Call Operation	System Call Description
2	print_float	$v0 = 2, $f12 = float number to be printed
4	print_string	$v0 = 4, $a0 = address of beginning of ASCIIZ string
6	read_float	$v0 = 6; user types a float number at keyboard; value is store in $f0
8	read_string	$v0 = 8; user types string at keybd; addr of beginning of string is stored in $a0; len in $a1

4. Show this program, properly working, to the TA or Tutor. Now save the program, adding it to the name_surname_MIPS.txt file that you created earlier. The file now contains 2 MIPS assembly programs you wrote. Save this MIPS.txt file, you will need it later.

Part 3. Using MIPS for matrix operations on floating point values (40 points)

5. Write a MIPS assembly program that takes in two matrices A and B in row-major order and performs several tasks over these matrices. The dimension of these matrices is N x N. Results generated by these operations should create outputs also in row-major form. The first part of the program will require you to write interface functions:

Create Matrix: user may define and enter matrix entries.
Create Value: user may define and enter value.
Display Matrix: user may display contents of matrix.
Display Value: user may display contents of value.

--the number of inputs N is a variable, so the user should be prompted and respond with a positive integer for N. Then the user is prompted for the values of each of the N x N inputs. All

must be floating point numbers for the matrix operations to work, so your program must not accept non-floats. When the inputting is done, give the user a message. [Note: since the number of inputs is known only at runtime, these inputs should be stored in dynamic data memory. A syscall to obtain heap memory can be used for this.]

6. You will be asked to write a set of Matrix Functions ***(given by the TA)*** on a matrix or two matrices.

-- If for some reason, the required operation can not be executed an error message should be displayed to the user.

7. When your program is working correctly and robustly (no matter what the inputs), call the TA or Tutor to show your program running, displaying correct results. Explain how it works (and calculates the Matrix Functions).

8. Now save the program by adding it to the name_surname_MIPS.txt file that you created earlier. The file now contains 3 MIPS assembly programs you wrote. Save this MIPS.txt file, you will submit it in the next part, and you may need its contents later.

Part 4. Submit your code for MOSS similarity testing

Submit your MIPS codes for similarity testing to the Unilica > Assignment specific for your section. You will upload one file: name_surname_MIPS.txt created in Parts 1, 2 and 3. Be sure that the file contains exactly and only the codes which are specifically detailed above. Check the specifications! *Even if you didn't finish, or didn't get the MIPS codes working, you must submit your code to the Unilica Assignment for similarity checking.* Your codes will be compared against all the other codes in the class, by the MOSS program, to determine how similar it is (as an indication of plaigarism). So be sure that the code you submit is code that you actually wrote yourself ! All students must upload their code to Unilica > Assignment while the TA watches. Submissions made without the TA observing will be deleted, resulting in a lab score of 0.

Part 5. Cleanup

1) Erase all the files that you created or used, including your MIPS files and this Lab2.doc file In other words, leave the lab computer exactly as you found it.
2) Put back all the hardware, boards, wires, tools, etc where they came from.
3) Clean up your lab desk, to leave it completely clean and ready for the next group who will come.

CONGRATULATIONS, you are done with Lab2, and one step closer to becoming a Computer Engineer!

LAB POLICIES

1. Each student will earn their own individual lab grade, by how much they do, how much they learn, how much they know. The questions asked by the TA will have an effect on your individual lab score, so be prepared to answer questions concerning the lab topics.
2. Lab score will be reduced to 0 if the code is not submitted for similarity testing, or if it is plaigarized. MOSS-testing will be done, to determine similarity rates. Trivial changes to code will not hide plaigarism from MOSS—the algorithm is quite sophisticated and powerful. Please also note that obviously you should not use any program available on the web, or in a book, etc since MOSS will find it. The use of the ideas we discussed in the classroom is not a problem.
3. You must be in lab, working on the lab, from the time lab starts until your work is finished and you leave. (Bathroom and snack breaks are the exceptions to this rule).
4. No cell phone usage during lab. Tell friends not to call during the lab hours--you are busy learning how computers work !
5. Internet usage is permitted only to lab-related technical sites. No Facebook, Twitter, email, news, video games, etc--you are busy learning how computers work !