CS224 - Fall 2016 - Lab #4 (Version 1: Nov. 4, 10:47 pm)

Floating Point Numbers/Instructions & Branch/Jump Instructions

Dates: Section 3, Wednesday, 9 November, 08:40-12:30
Section 1, Wednesday, 9 November 13:40-17:30
Section 4, Thursday, 10 November 13:40-17:30
Section 2, Friday, 11 November 13:40-17:30

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. Furthermore the preliminary work provides some exercises for branch, jump instructions, and floating number representation.

Please bring and drop your preliminary work into the box provided in front of the lab by 5:00 pm Tuesday November 8.

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 the relevant parts. Be sure that the file contains exactly and only the codes which are specifically detailed below. 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 plagiarism).

We plan to run your code to make sure that it really works.

You have to show your work to your TA and upload it before by 11:30 in the morning lab and by 4:30 in the afternoon lab. Note that you cannot wait for the last moment to do this. Make sure that you have shown your work to the Lab TA before uploading. If you wait for the last moment and show your work after the deadline time 30 points will be taken off. If you upload before showing your work to the TA you will receive 0 (zero) for the lab work.

Part 1. Preliminary Work Related to branch/jump instructions and number representation

(30 points)

Please prepare word document for this part and provide the required information as specified earlier (Lab no, your name, section no., etc.)

1. Consider the following code

```
continue:
        nop
        nop
        nop
        be      $t0, $t1, continue
        bne     $t0, $t1, next
        nop
        nop
        nop
next:
```

Generate the object code for the be and bne instructions in hexadecimal. Show your work.

2. Consider the following code

```
        .text
        .globl  _Lab4main
_Lab4main:
        nop
        nop
        j       _Lab4main
        j        next
        nop
        nop
```

next:

Generate the object code for the j (jump) instructions in hexadecimal. Show your work.

3. Using 4 bits what is the minimum and maximum integers that we can represent using sign magnitude representation. Explain your answer. Answer the same question this time for 16 bits.

4. Using 4 bits what is the minimum and maximum integers that we can represent using 2's complement notation representation for negative numbers. Explain your answer. Answer the same question this time for 16 bits.

5. For a number with hexadecimal representation is 4AB.5F. Show its IEEE 754 single precision an double precision representations.

6. For the following single precision number represented by using IEEE 754 format give the corresponding decimal number: 0x43824000

Part 2. Exercising MIPS floating point instructions

(30 points)

1. Write a simple program which will prompt user to enter a set of floating point numbers. These numbers will be stored in an array that will be dynamically allocated. First ask the user how many numbers the user wants to enter. Then enter them one by one. Provide a simple menu to find minimum, maximum, summation of numbers, and the difference between the minimum and the maximum number.

Part 3. Using MIPS for matrix operations on floating point values

(40 points)

Write a MIPS assembly program that takes in a matrix in row-major order. Row-major means that if we have 4 by 4 matrix, first you consecutively store the 4 elements of the first row, then store the 4 elements of the second row. In this example the first entry of the second row will be 16 bytes away from the beginning of the matrix. 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:

a. Create Matrix: user may define and enter matrix entries.

b. Display Matrix: user may display contents of matrix. Provide a nice presentation with proper separation of entries etc.

c. Write a method that performs matrix addition like A= A + B To the subprogram you should pass the address of the matrices and the matrix dimension information. Note that we have square matrices and the first matrix will be modified.

d. Write a method that returns the transpose of the input parameter matrix. Note that the input matrix is changed.

e. Write a method to check if a matrix is a symmetric matrix.

--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. 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 need to be used for this.]

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

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 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.
2) When applicable 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.

For your convenience a set of floating point instructions are provided.

Use them according to your needs. You may not need to use all of them.

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
```

Comparison:

```
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 some 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
6	read_float	$v0 = 6; user types a float number at keyboard; value is store in $f0