# ICS 233 Project1 – Spring 2007
# Computer Architecture and Assembly Language

Floating-Point Addition, Matrix Multiplication, & Instruction Frequencies

Due Saturday, May 5, 2007 by Midnight

**Objectives:**

- Writing and Testing MIPS assembly language code
- Doing floating-point arithmetic in software to understand it thoroughly
- Counting instruction frequencies
- Teamwork

**Problem 1: Single-Precision Floating-Point Addition in Software**

Write and test a MIPS assembly language program to do single-precision floating-point addition in software rather than in hardware. The procedure *floatadd* should receive its input parameters in **`$a0`** and **`$a1`** (as single-precision floating-point numbers) and produce its result in **`$v0`** (as single-precision float). **You cannot use the floating-point addition instruction `add.s` to do the addition**. Only integer instructions are allowed. Write additional procedures, if needed, to extract the fields, normalize, and round the result significand.

You should also make sure to handle special cases:

■ Zero, infinity, and NaN

■ Overflow and underflow

■ Denormalized numbers

Round the result to the nearest even, which is the default rounding mode in IEEE 754 standard. This is the only rounding mode that should be supported.

Write a *check* procedure to do the floating-point addition using the **add.s** instruction. Compare the result of the *check* procedure against the result produced by the *floatadd* procedure to ensure correctness.

Write a *main* procedure to call and test the *float_multiply* procedure. Specifically, you should ask the user to input two floating-point numbers and to print the result of addition.

A sample run should look as follows:

```
Enter 1st float: 1.297e-15
Enter 2nd float: 7.014e-13
Result of floatadd: 7.026969e-13
Result of add.s:    7.026969e-13
```

**Problem 2: Matrix Multiplication and Counting Instruction Frequencies**

Write and test a MIPS assembly language program to do matrix multiplication of *n* by *n* matrices of double-precision floating-point numbers. The matrix data should be defined in the program itself under the data segment. Define six matrices (two 4×4, two 7×7, and two 10×10 matrices). Initialize them with arbitrary floating-point values. Write a procedure to do matrix multiplication of *n* × *n* matrices, where *n* is passed to the procedure as a parameter.

Call the procedure three times to do the multiplication of the 4×4, 7×7, and 10×10 matrices. Define arrays to store the result matrices, and display the three output matrices. Test and verify your results.

After succeeding in matrix multiplication and producing the correct results, you will analyze the MIPS code of the matrix multiply procedure, to have a better understanding of instruction frequencies. *You will count the dynamic number of instructions that are executed at runtime to determine their frequencies in the matrix multiply procedure*. You need a total of five counters to count instructions for the following classes of instructions:

- Class 1: ALU instructions: **add, addu, addiu, or, ori, slt, sltu, sll, srl, mflo**, etc.
- Class 2 is for integer multiply instructions: **mult, multu**.
- Class 3 is for floating-point instructions: **add.d, mul.d**, etc.
- Class 4 is for load and store instructions: **lw, sw, ldc1**, **sdc1**, etc.
- Class 5 is for branch and jump instructions: **beq, bne, blez, bgtz, bltz, bgez, j, jr**, etc.

You will *augment the code of the matrix multiply procedure with additional instructions* to count the original number of instructions. At the beginning of the procedure, initialize all counters to zeros. Before each instruction, insert additional instructions to count that instruction. For example, if the original instruction is **addiu** then increment the counter of Class1 by inserting additional instructions to do the increment before the instruction itself. If the same instruction is executed 100 times (in different loop iterations), it will be counted as 100. Count only the real instructions. For pseudo-instructions, count the equivalent real instructions. Make sure that your additional code does not interfere with the original program code. Count only the original instructions of matrix multiply, not the new ones that you have added.

At the end of the matrix multiply procedure, display the statistics that you have produced for each call of the procedure. A sample run is show below:

```
Multiplying 4x4 Matrices:
Total               instructions = ???
ALU                 instructions = ??, Percentage = ?%
Integer Multiply    instructions = ??, Percentage = ?%
Floating-Point      instructions = ??, Percentage = ?%
Load & Store        instructions = ??, Percentage = ?%
Branch & Jump       instructions = ??, Percentage = ?%

Output Matrix:
. . .

Multiplying 7x7 Matrices:
Total               instructions = ???
ALU                 instructions = ??, Percentage = ?%
Integer Multiply    instructions = ??, Percentage = ?%
Floating-Point      instructions = ??, Percentage = ?%
Load & Store        instructions = ??, Percentage = ?%
Branch & Jump       instructions = ??, Percentage = ?%

Output Matrix:
. . .
```

```
Multiplying 10x10 Matrices:
Total                instructions = ???
ALU                  instructions = ??, Percentage = ?%
Integer Multiply     instructions = ??, Percentage = ?%
Floating-Point       instructions = ??, Percentage = ?%
Load & Store         instructions = ??, Percentage = ?%
Branch & Jump        instructions = ??, Percentage = ?%

Output Matrix:
. . .
```

**Tools:**

Use MARS simulator to write and test your code.

**Groups:**

Two or at most three students can form a group. Make sure to write the names of all the students involved in your group on the project report.

**Submission Guidelines:**

All submissions will be by email sent to:

mudawar@kfupm.edu.sa ; ghalioun@cccse.kfupm.edu.sa

Subject: Project 1 from *your Name, your ID number*

Attach one zip file containing the source code of programs 1 and 2, as well as a report. Make sure that both programs are well documented.

**Report:**

The report should contain the names of all the group members and the division of work among the group members (how the work was divided and who has done what). It should also contain sample runs of programs 1 and 2. For program 1, discuss all the cases that were handled by the *floatadd* procedure, and provide many sample inputs and outputs making sure to test all cases, such as normalized numbers, denormalized numbers, zero, overflow, and underflow. Also test and demonstrate rounding. For program 2, show the statistics and the output of the matrix multiply procedure for different matrix sizes. Comment on these statistics, how they change with the matrix size, and the additional code that you inserted.

**Grading policy:**

The grade will be divided according to the following components:

- Correctness of code: program produces correct results
- Completeness of code: all cases were handled properly
- Documentation of code: program is well documented
- Participation and contribution to the project
- Report document

**Late policy:**

The project should be submitted on the due date by midnight. Late projects are accepted, but will be penalized 5% for each late day and for a maximum of 5 late days (or 25%). Projects submitted after 5 late days will not be accepted.