COE 308 – Computer Architecture
Term 072 – Spring 2008

Project 1: Writing, Simulating, and Testing MIPS Assembly Code

Due Monday, April 7, 2008 by Midnight

Objectives:

- Using the MARS MIPS simulator tool
- Writing, simulating, and testing MIPS assembly language code
- Doing floating-point arithmetic in software to understand it thoroughly
- Teamwork

Problem 1: Robot Kinematics

Write and test a MIPS assembly language program to compute the robot kinematics of a six degree-of-freedom robot arm. In this project we consider 3-dimensional vectors and matrices. Each vector V is 3×1 to say that there are three vector elements V[*i*]. Each element V[*i*] is 32-bit integer, where *i* is in [0,1,2]. V is an array whose three elements are stored in successive memory locations where each element occupies 4 bytes.

Similarly, we use matrices where each matrix M is 3×3 to say that there are three rows and each row has three elements. Matrix element M[*i*][*j*] is also 32-bit integer, where *i* and *j* are in [0,1,2]. M is a 2-dimentional array that is stored according to Row-Major scheme, each row of three elements are stored in successive memory locations starting from first row, second row, and at last the third row.

You should have the following three "Linear Algebra" procedures:

- ■ A procedure to compute the sum of two vectors such as Vector_Add(Vd, Vs, Vt). This procedure passes two pointers Vs and Vt (base addresses) of source vectors which allows accessing the vector data within the procedure. Next it computes the sum of the two source vectors and store in memory the resulting vector as destination vector pointed by Vd.
- ■ A procedure to compute the matrix-vector product: Matrix_Vector_Product(Vd, Ms, Vt). This procedure passes two pointers Ms and Vt (base addresses) of source matrix and source vector which allows accessing the matrix and vector elements within the procedure. Next it computes the matrix-vector product and store the resulting vector in memory as destination vector pointed by Vd.
- ■ A procedure to compute the product of two matrices such as Matrix_Product(Md, Ms, Mt). This procedure passes two pointers Ms and Mt (base addresses) which allow accessing the matrix elements within the procedure. Next it computes the product of the two source matrices and stores in memory the product as destination matrix pointed by Md.

There are six input vectors LV[*i*] and six input matrices R[*i*]. Read the input vectors and matrices from a text file. The input text file should list the six input vectors on the first six lines, followed by the six input matrices on the next six lines. Comments that begin with a hash (#) symbol can appear at the end of each line and should be ignored. A sample input text file is given below. The vector and matrix element values are separated by space or tab characters.

```
0    0    300  # LV[0]
5    10   300  # LV[1]
5    10   250  # LV[2]
0    10   250  # LV[3]
10   50   100  # LV[4]
50   50   100  # LV[5]
1    0    0    0    1    0    0    0    1    # R[0]
1    0    0    0    0    -1   0    1    0    # R[1]
1    0    0    0    0    1    0    -1   0    # R[2]
1    0    0    0    -1   0    0    0    1    # R[3]
1    0    0    0    1    0    0    0    -1   # R[4]
0    0    1    1    0    0    0    1    0    # R[5]
```

Ask the user to enter the name of the text file. Open the text file and read its content. Use **`syscall 13`** (**`$v0 = 13`**) to open a text file and **`syscall 14`** to read from a file. Check the MARS Help to learn more about syscalls. Read all the characters of the input file into an array in memory. Then traverse the array character by character to convert digit character strings into integer values. Skip the comments at the end of each line. Initialize the six input vectors LV[i] and the six input matrices R[i] in your program with their corresponding integer values.

The aim of this project is to iteratively perform the following computation:

__**Step 1:**__ Initialization. We need a working vector V and a working matrix M initialized to:

`V=(0,0,0) and M = R[0]`.

__**Step 2:**__ Compute:

```
for k = 0 to 5 {
        M = M × R[k]
        V = V + M × LV[k]
}
```

At the end of the above loop we need to save the final values of M and V which are the result of the above algorithm. The program outputs are the final values of V and M. You need to submit your source assembly code written in MIPS for MARS, the running output of the program, and your comments on the above implementation.

Problem 2: 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 *floatadd* procedure. Specifically, you should ask the user to input two floating-point numbers and to print the result.

A sample run should look as follows:

```
Enter 1st float: 1.25e-4
Enter 2nd float: 0.75e-3
Result of floatadd: 8.75e-4
Result of add.s:    8.75e-4
```

Tool

Use the MARS tool to write, execute, and test your code. To get started, familiarize yourself with the MARS MIPS simulator. You should familiarize yourself with the assembly language syntax and system calls. The MARS Help provides a description of all the syscalls that are needed to complete this project. It also provides a list of all the basic and pseudo instructions.

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.

Coding and Documentation

Develop the code for the given problems with the following aspects in mind:

- Correctness: the code works properly
- Completeness: all cases have been covered
- Efficiency: the use of relevant instructions and algorithms
- Documentation: the code is well documented through the appropriate use of comments.

Report Document

The project report must contain sections highlighting the following:

- **Program Design**

  Specify clearly the design of each procedure giving detailed description of the algorithm used/developed and the implementation details.

- **Program Simulation**

  Describe all the simulator features that you have used for simulating your code with a clear emphasis on its advantages and limitations (if any), debugging for errors, the use of system calls and displaying the results of the program.

- **Program Output and Discussion**

  Provide snapshots of the Simulator window and show all the results.

  Discuss all the cases that were handled. For program 1, provide more than one input text file and show the final values of M and V for each run.

  For program 2, provide sample inputs and outputs and discuss all the cases that were handled by the *floatadd* procedure, such as normalized and denormalized numbers, zero, overflow, and underflow. Also test and demonstrate rounding.

- **Teamwork**

  Group members are required to divide the work equally among themselves, so that everyone is involved in algorithm design, program development, and debugging.

  Show clearly the division of work among the group members using a Chart and also prepare a Project execution plan showing the time frame for completing the subtasks of the project.

  Students who **helped** other team members should mention that to earn credit for that.

Submission Guidelines

All submissions will be done through WebCT. Submit one zip file containing the source code of programs 1, 2, and the report document. Also, submit a **hard copy** of the report in class.

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
- Team Work : 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.