MIPS Extended Math Functions

Submitted By: Joe Fennimore, Christian Gunderman, Phillip Blackburn, Elliot Essmen, Laura Subak
Submitted To: Swarup Bhunia
Submitted On: May 2, 2014

Problem Statement: MIPS has a very limited suite of math functions in its native command library, so our group decided to package additional functions which pertain to common mathematical operations more advanced than add, subtract, multiply and divide. These functions include, in no particular order:

Unit Conversions	Average, Median, and Mode of a set of numbers	Trigonometric Functions
Multiplying by Powers, Factorial, and Logarithms	Volume and Surface Area of Common Shapes	Matrix Multiplication

Major Challenges:

The crux of the problem for developing each of these functions is that it required each complex function to be broken down into individual operations of add, multiply, and divide which could only be carried out between two variables at a time. Another limitation is that in our implementation user input is given by entering only one value at a time. This was done for the convenience of using MIPS's syscall functions, which natively only accept one value at a time anyway. Once each function is broken down correctly however, MIPS can solve all of these math functions with little to no wait time.

Key Components:

For the matrix multiplication section, the main challenges involved determining the best way to get user input for the individual array elements and then indexing them into the matrices correctly. For my first versions of the program, I had issues determining where each element of user input ended up in memory, as the matrices had to be represented by arrays that were split on word boundaries. I had a recurring bug where none of my array allocations would end up on a word boundary even though they were allocated a space in word-multiples (i.e.. multiples of 4). I was able to solve the issue by choosing to format my array (Matrix) allocations after declaring ".align 4" beforehand. However, declaring each array with ".align 4" individually continued to cause the misalignment issue. I was unable to find out why this error would occur, and likely wouldn't have figured it out if not through trial-and-error.

Additionally, the algorithm for matrix multiplication in a higher level language like C involves multiple nested loops and iterations across three counter variables. Replicating this in MIPS was difficult as it involved placing several label statements: one to increment each counter variable, and one to check whether each counter had reached it's maximum value to see if a jump had to be made to increment the counters. This is excluding the actual matrix multiplication itself, which required a lot of debugging as I had to use numerous temporary registers that were hard to keep track of.

For the average, median, and mode sections, while the math for these functions is relatively simple, working with the limited set of MIPS functions made it difficult. These functions required implementing sorts and searches of arrays. The mode calculator was probably the most difficult to implement, because it required the use and tracking of two arrays.

For the trigonometry, all six trigonometric functions could be derived from the sine function, and so once an accurate algebraic approximation could be made for sine, the other five functions called sine in various ways.

Integration of Components: (e.g. user interface/menus tying everything together)

Since this project was distributed across several team members, we needed a way to keep our code organized and cohesive. Our eventual solution was to follow a file nomenclature similar to: username-library.asm. This code was then managed in Git to allow us to track our changes. Each team member was assigned the task of making his or her functions, as well as the UI asking for inputs and printing outputs for each function. All labels in each library are of the form: library_label, to prevent conflicts. In this way, the functions can operate fully independently of each other, but each one still has their own entry point, allowing them to be added into a unified application. When each component was pushed to Git, they were then integrated into the menus by adding an additional *beq* instruction for the desired menu option to an existing menu template. This branch jumps to the *libraryname_main* label corresponding to the library or function when it is selected from the menu.

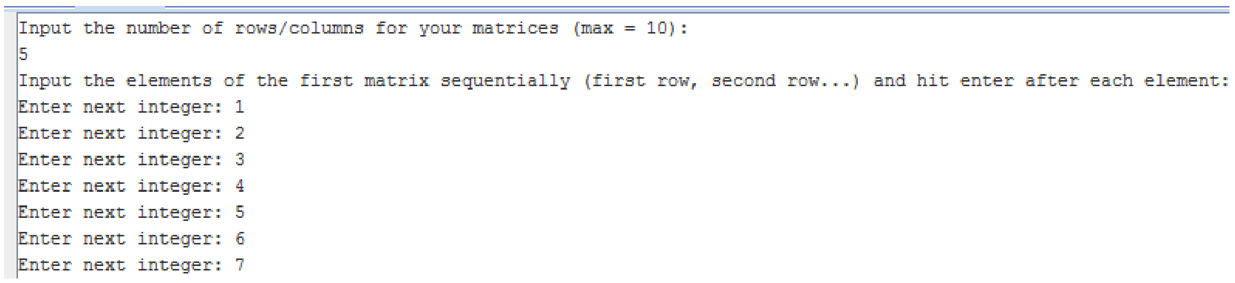
To further facilitate development, we use a shared set of macros that implement commonly used bits of code that are not complex enough to mandate using a full routine. gundermanc-macros.asm contains 13 macros for common tasks, as well as system service calls. These macros have the benefit of making much of the code more readable, and also simplify the implementation of routines and pushing return values to the stack.

User Interface:

For this project, we decided to keep our UI limited to console inputs and outputs. The user interface is designed around a menu hierarchy that splits from general categories, to specific categories, to individual functions. When the program is first launched, the user is presented with a list of the authors and a menu of math libraries. A particular library can be selected using the number keys and [Enter]. After selecting a library, the user is presented with a submenu of functions for that particular library where he or she can pick which function to calculate. The function then prompts the user for inputs and prints the results. Finally, it jumps back to the *libraryname_main* label corresponding to the library that the function belongs too.

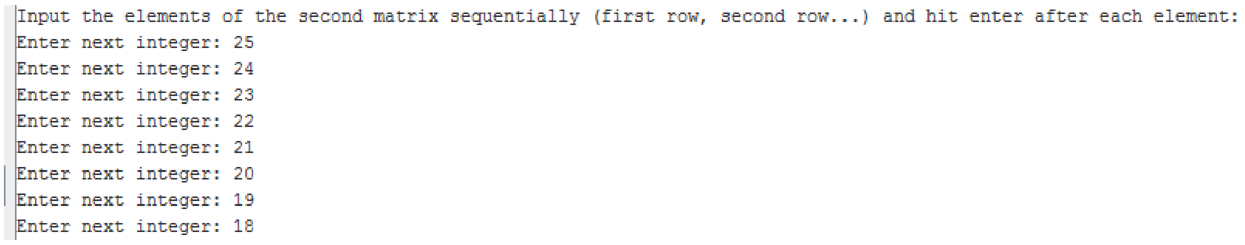
Snapshots:

Matrix Multiplication



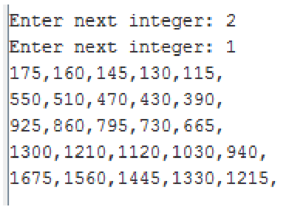
```
Input the number of rows/columns for your matrices (max = 10):
5
Input the elements of the first matrix sequentially (first row, second row...) and hit enter after each element:
Enter next integer: 1
Enter next integer: 2
Enter next integer: 3
Enter next integer: 4
Enter next integer: 5
Enter next integer: 6
Enter next integer: 7
```

(break)



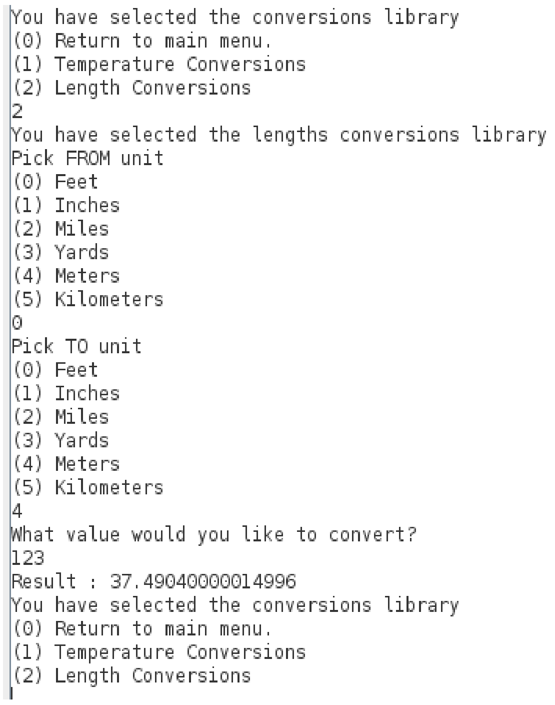
```
Input the elements of the second matrix sequentially (first row, second row...) and hit enter after each element:
Enter next integer: 25
Enter next integer: 24
Enter next integer: 23
Enter next integer: 22
Enter next integer: 21
Enter next integer: 20
Enter next integer: 19
Enter next integer: 18
```

(break)



```
Enter next integer: 2
Enter next integer: 1
175,160,145,130,115,
550,510,470,430,390,
925,860,795,730,665,
1300,1210,1120,1030,940,
1675,1560,1445,1330,1215,
```

Unit Conversions:



```
You have selected the conversions library
(0) Return to main menu.
(1) Temperature Conversions
(2) Length Conversions
2
You have selected the lengths conversions library
Pick FROM unit
(0) Feet
(1) Inches
(2) Miles
(3) Yards
(4) Meters
(5) Kilometers
0
Pick TO unit
(0) Feet
(1) Inches
(2) Miles
(3) Yards
(4) Meters
(5) Kilometers
4
What value would you like to convert?
123
Result : 37.49040000014996
You have selected the conversions library
(0) Return to main menu.
(1) Temperature Conversions
(2) Length Conversions
```

Specific Contributions:

- Christian Gunderman - Unit Conversions and Menu Navigation Designer and Coordinator.
- Joe Fennimore - Trigonometry and Geometry
- Phillip Blackburn - Linear Algebra
- Elliot Essman - Powers, factorial, and logarithms
- Laura Subak - Average, Median, and Mode Calculator