CSE 220: Systems Fundamentals I

Stony Brook University
Homework Assignment #7
FALL 2023

Due: Friday, Dec 8, 2023 by 11:59 pm Eastern Time (Hard deadline No extension in any case)

Learning Outcomes

After completion of this programming project you should be able to:

- Have a good understanding of MIPS assembly programming.
- Implement non-trivial algorithms that require conditional execution and iteration.
- Read and write one/two-dimensional character arrays of arbitrary length.
- Design and implement functions that implement the MIPS assembly function calling conventions.

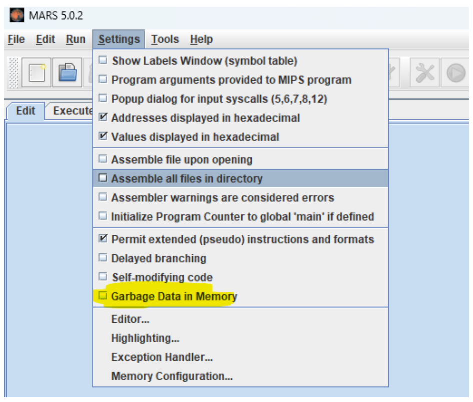
Download the Template Repository

Accept this invitation to start the assignment and clone the repository. Inside `hw7a.asm`, `hw7b.asm`, and `hw7c.asm` you will find some basic code to start with.

Overview

Your task in this assignment is to implement various functions as specified below. If you are having difficulty implementing these functions, write out pseudocode or implement the algorithms in a higher-level language first. Once you understand the algorithm and what steps to perform, then translate the logic to MIPS assembly code.

Be sure to initialize all of your values (e.g., registers) within your functions. Never assume registers or memory will hold any particular values (e.g., zero). MARS initializes all of the registers and bytes of main memory to zeroes by default (Check "Garbage Data in Memory" to execute with garbage data). **The TAs will fill the registers and/or main memory with random values before calling your functions.**



☐ Show Labels Window (symbol table)
☐ Program arguments provided to MIPS program
☐ Popup dialog for input syscalls (5,6,7,8,12)
☑ Addresses displayed in hexadecimal
☑ Values displayed in hexadecimal
☐ Assemble file upon opening
☐ Assemble all files in directory
☐ Assembler warnings are considered errors
☐ Initialize Program Counter to global 'main' if defined
☑ Permit extended (pseudo) instructions and formats
☐ Delayed branching
☐ Self-modifying code
☐ Garbage Data in Memory
Editor...
Highlighting...
Exception Handler...
Memory Configuration...

Preliminaries

- When writing assembly code, try to stay consistent with your formatting and to comment as much as possible. It is much easier for your TAs and the professor to help you if we can quickly figure out what your code does.
- You personally must implement the programming assignments in MIPS Assembly language by yourself. You may not write or use a code generator or other tools that write any MIPS code for you. You must manually write all MIPS assembly code you submit as part of the assignments.
- Do not copy or share code.
- Submit your final .asm file to GitHub, as with the prior assignments. Code that crashes and cannot be graded will earn no credit. *No changes to your submission will be permitted once the deadline has passed .*

How Your MIPS Code Will Be Graded

As with the C assignments, grading scripts will execute your code with input values (e.g., command-line arguments, function arguments) and will check for expected results (e.g., print-outs, return values, etc.). For this assignment, TAs will manually run your program. Therefore it is important that you generate input and output exactly what is asked. It is imperative that you implement the “print statements" for input and output exactly as specified and asked in the assignment.

Some other items you should be aware of:

- Each test case must execute in 10000 instructions or fewer (you should be able to complete each part with 300 instructions).
- Efficiency is an important aspect of assembly programming. This maximum instruction count will be increased in cases where a complicated algorithm might be necessary, or a large data structure must be traversed. To find the instruction count of your code in MARS, go to the **Tools** menu and select **Instruction Statistics**. Press the button marked **Connect to MIPS**. Then assemble and run your code as normal.
- Any excess output from your program (debugging notes, etc.) will impact grading. **Do not leave erroneous print-outs in your code.**
- We will provide you with a small set of test cases for each assignment to give you a sense of how your work will be graded. *It is your responsibility to test your code thoroughly by creating your own test cases. Provide these tests in the Readme file as well.*
- The testing framework we use for grading your work will not be released, but the test cases and expected results used for testing will be released.
- You need to give your ID and name in the Readme file as well.

Register Conventions

You must follow the register conventions taught in lecture and reviewed in recitation. Failure to follow them will result in loss of credit when we grade your work. Below is a brief summary of the register conventions and how your use of them will impact grading. See the lecture notes for more details.

- It is the callee's responsibility to save any $s registers it overwrites by saving copies of those registers on the stack and restoring them before returning.
- If a function calls a secondary function, the caller must save $ra before calling the callee. In addition, if the caller wants a particular $a, $t or $v register's value to be preserved across the secondary function call, the best practice would be to place a copy of that register in an $s register before making the function call.
- A function which allocates stack space by adjusting $sp must restore $sp to its original value before returning.
- Registers $fp and $gp are treated as preserved registers for the purposes of this course. If a function modifies one or both, the function must restore them before returning to the caller. There really is no reason for your code to touch the $gp register, so leave it alone.
- Please provide extensive comments. Atleast one comment for each instructions. Poor commenting will result in lose i

The following practices will result in loss of credit:

- "Brute-force" saving of all $s registers in a function or otherwise saving $s registers that are not overwritten by a function.
- Callee-saving of $a, $t or $v registers as a means of "helping" the caller.
- "Hiding" values in the $k, $f and $at registers or storing values in main memory by way of offsets to $gp. *This is basically cheating or at best, a form of laziness, so don't do it. We will comment out any such code we find.*

Part A (hw7a.asm) (Drawing with MIPS):

Write a program that asks if the user wants a triangle, a square or a pyramid. It then asks the user for the size of the object (the number of lines it takes to draw the object). The program then writes/draws a triangle (right angled), a square or a pyramid of stars "*" to the console.

<u>Some Examples/Test Cases:</u>

(Square of height 6)
```
******
******
******
******
******
******
```

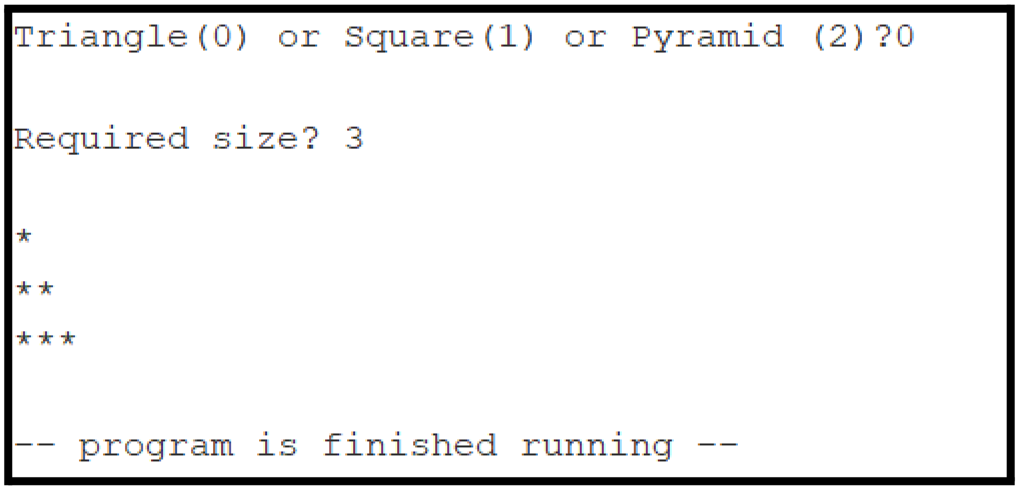
(Triangle of height 6)
```
 *
 **
 ***
 ****
 *****
 ******
```

(Pyramid of height 4)

```
    *
   * *
  * * *
 * * * *
```

Write a subroutine for each figure. In them, use a subroutine print_star_line that writes a line of a given number of stars. (that number is passed as an argument to print_star_line function).

Sample output for the problem using MARS 4.5:



```
Triangle(0) or Square(1) or Pyramid (2)?0

Required size? 3

*
**
***

-- program is finished running --
```

Part B (hw7b.asm): Swapping data between arrays

In this program, the content of two one dimensional **integer** arrays will be exchanged(swapped). The swapping operation will be performed for each array location. The swapping operation for two arrays A and B will be performed as follows:

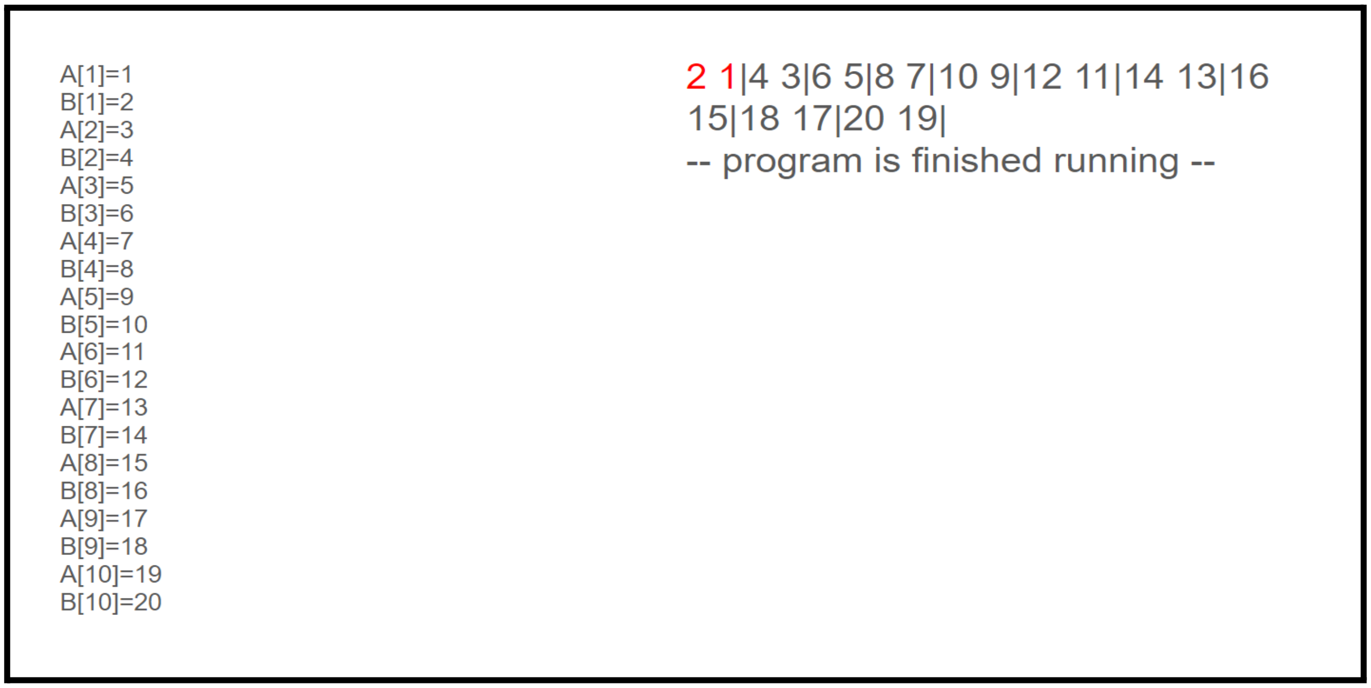
Temp = A[i]
A[i] = B[i]
B[i] = temp

You will implement the swapping operation in MIPS. Both arrays will be static and defined in the data stack (.data) as:

A: .space 40 # memory space for Array A
B: .space 40 # memory space for Array B
m: .word 10 # number of elements (integer)
# TAs can change these values . However, The size of arrays (.space) will be >= m x 4

This example is for 10 integer elements. You need to takecare of the elements size and clearly defined it. The TAs might change it. You need to ask input values from user for each element of arrays, A and B, and then print the swapped values at the end.

Please see the following snapshot of the MARS terminal. The left hand side is the input values from the user and the right hand side are the swapped values.



```
A[1]=1                                    2 1|4 3|6 5|8 7|10 9|12 11|14 13|16
B[1]=2                                    15|18 17|20 19|
A[2]=3                                    -- program is finished running --
B[2]=4
A[3]=5
B[3]=6
A[4]=7
B[4]=8
A[5]=9
B[5]=10
A[6]=11
B[6]=12
A[7]=13
B[7]=14
A[8]=15
B[8]=16
A[9]=17
B[9]=18
A[10]=19
B[10]=20
```

Output with 10 elements for Array A and B using MARS 4.5

Here is what is expected from your code (see the above Figure). You will ask the user for each element of array in the way it is printed in the figure; input for A[1], B[1], … printed on the screen and values are being asked from the user. Here, we are using 10 elements. Therefore, we have total 20 inputs; 10 for array A and 10 for array B. In the end you will print the swapped result in the format shown in the figure. **2 1** are the first elements of A and B respectively that are swapped (See the left hand side as well. A[1]=1 and B[1]=2. After swapping, A[1]=2 and B[1]=1). We have "|" between the swapped elements. The second elements for the arrays after swapping are **4 3** and so on. Any other output format will not be accepted and will result in mark deduction.

Part C (hw7c.asm) (Matrix Vector Multiplication)

(Check the example on 2D matrix (row major) we covered in the class. You can use the code as the starting point for this part)

Matrix-vector multiplication can be described by the following function:

```
int* MVM (int n, int A[n][n], int X[n]) {
 int* V = new int[n]; // allocate an array of n ints   int i,
j;
 for (i=0; i<n; i++) {
 int sum = 0;
 for (j=0; j<n; j++) { sum = sum + A[i][j] * X[j]; }
V[i] = sum;
 }
 return V; // return a pointer to vector V  }
```

In addition, you need the following support functions:

```
int* read_vector (int n) {
 // allocate a vector of n ints
 // ask the user to input n ints and read them into allocated
vector   // return address of vector
}
int* read_matrix (int n) {
 // allocate a matrix of n*n ints
 // ask the user to input n*n ints and read them into allocated
matrix   // return address of matrix
}
void print_vector (int n, int V[n]) {
 // Display the n elements of vector V
}
```

Translate the above functions into MIPS code. Write a **main** function that asks the user to input **n** elements for matrix/vector by calling functions **read_matrix** and **read_vector** from **main** to read a matrix and a vector. Call function **MVM** to do matrix-vector multiplication. Then call **print_vector** to print the result vector.

Your MIPS program should be well written and documented. The final outcome should be a correct vector with correct dimension. If matrix is m x n and vector is nx1, then the final vector should be mx1 (print with m rows and one element in each row).

- You will define the rows and columns for matrix and vector (column is always 1) in the program and TAs might change it.
- You need to check the compatibility of Matrix and Vector. If columns of the matrix are not equal to the rows of vector, your program should exit with the message " **NOT WORKABLE BECAUSE OF THE DIMENSIONS**"
- You will allocate the necessary space for arrays in the program (.space directive). For this HW, we set it 400 and will not change it.
- Cols x Rows <= Allocated memory. You need to have this check as well
- If Cols x Rows > Allocated memory, your program should exit with the message **"NOT WORKABLE BECAUSE OF THE MEMORY".**
- You should ask inputs from the user. Since, MIPS follows row major, you can number each element accordingly. A[0][0] is element 1, A[0][1] is element 2 and so on. You will ask the input from the user based on element number.
    - Enter element 1 = (input from user for A[0][0])
    - Enter element 2 = (input from user for A[0][1])
    - Enter element 3 = (input from user for A[0][2])
- The same approach will be used for taking inputs for the vector
- The program should print the result at the end (vector)

Testing & Grading Notes

- The assignment will be marked by running each part manually by the TAs using MARS 4.5.
- Add atleast 10 test cases for each part. The TAs will have their own for evaluation.
- Comments are important. We will read and deduct reasonable marks if they are not clear.
- Write your SBU ID and Name in the readme file
- Remember, if the program works on your test cases, there is a high chances that code works and is correct.

Academic Honesty Policy

Academic honesty is taken very seriously in this course. By submitting your work for grading you indicate your understanding of, and agreement with, the following Academic Honesty Statement:

1. I understand that representing another person's work as my own is academically dishonest.
2. I understand that copying, even with modifications, a solution from another source (such as the web or another person) as a part of my answer constitutes plagiarism.
3. I understand that sharing parts of my homework solutions (text write-up, schematics, code, electronic or hard-copy) is academic dishonesty and helps others plagiarize my work.
4. I understand that protecting my work from possible plagiarism is my responsibility. I understand the importance of saving my work such that it is visible only to me.
5. I understand that passing information that is relevant to a homework/exam to others in the course (either lecture or even in the future!) for their private use constitutes academic dishonesty. I will only discuss material that I am willing to openly post on the discussion board.
6. I understand that academic dishonesty is treated very seriously in this course. I understand that the instructor will report any incident of academic dishonesty to the University's Academic Judiciary.
7. I understand that the penalty for academic dishonesty might not be immediately administered. For instance, cheating on a homework assignment may be discovered and penalized after the grades for that homework have been recorded.
8. I understand that buying or paying another entity for any code, partial or in its entirety, and submitting it as my own work is considered academic dishonesty.
9. I understand that there are no extenuating circumstances for academic dishonesty.