Hacettepe Computer Engineering - Spring 2021

BBM234 - MIPS Project

HABABAM SINIFI - The Chaos Class Tries to Cheat

Hababam Sınıfı - the Chaos Class Tries to Cheat

Külyutmaz Necmi enters the Hababam Sınıfı and announces that he's going to test the students. He dictates a couple of ridiculously hard questions and starts shouting:

"No talking, no looking to the right, no looking to the left, no leaning under the desks. I'll take your exam papers. Get ready to start. Five, four, three, two, one, zero, go!"

And as he walks over the student desks, he continues shouting frantically:

"Don't you dare try to cheat! In all of my thirty years of teaching, not a single student managed to cheat EVER. Bana kül yuttura-mazsınız!"

The situation in the classroom was pretty tense and at the very least morbid (to get a better idea see this footage).

As the students have been coming up with more and more sophisticated cheating techniques, while the poor Külyutmaz has been getting older with declining fitness and agility, not to mention the eyesight, he finally understood that it was high time he found some alternative ways of catching the cheaters.

For this exam he came up with a brilliant idea of utilizing a computer for those purposes, and little did the cunning Hababam Sınıfı students know what was about to hit them.

The school had one SGI Indigo workstation computer which runs MIPS R3000, and all he needed were some MIPS programmers to help him realize his clever plan.

This time, you are going to be "the good guys" and help Külyut-maz Necmi catch the cheaters.

Hacettepe Computer Engineering - Spring 2021

BBM234 Computer Organization - MIPS Project

Deadline: Saturday, 10/04/2021 at 23:59:59

MIPS Simulators

In this project, you will learn how to write and simulate **MIPS** code.
You can either use **QtSpim** or **MARS** MIPS simulator for this project.

Download **QtSpim** from http://spimsimulator.sourceforge.net/ and install it to your computer.
There are several tutorials for QtSpim, some of which are listed below. If you can find better ones, please let us know for future references.

- https://www.youtube.com/watch?v=r8WcV7AiLXs
- https://www.lri.fr/~de/QtSpim-Tutorial.pdf
- https://ecs-network.serv.pacific.edu/ecpe-170/tutorials/qtspim-tutorial

The second simulator you can use is **MARS**, which can be downloaded from the following link:
http://courses.missouristate.edu/kenvollmar/mars/

How to Write and Simulate Your Code

- You should use a text editor such as notepad or wordpad and write your MIPS program first. If you are using MARS simulator, you can edit your MIPS programs directly inside the simulator.
- Save your program with an extension .asm
- Open the simulator of your choice and load your program.
- Assemble and run your program.
- Your results will be either in registers or in memory (data segment) based on your program output.

Terminating MIPS Programs on Simulators

Keep in mind that your main function will be called by the simulator as you call any other function in your code (think of it as THE ACTUAL MAIN function is provided by the simulator, while your code is just some other functions that is called). Therefore, there are some rules you need to follow, namely, you need to consider what you need to do to make sure the return address is retrieved properly when your code terminates.

You have two options for termination. The first one: before you type any code, you should first save the return address **$ra** on stack, and as the last step restore it by executing **jr $ra**. An example is shown in Figure 1 below.

The second option: you can execute the **'exit' syscall** as shown in the Figure 2.

```
    .text
    .globl main
main:
    addi $sp, $sp, -4    # make room on stack to save $ra
    sw $ra, 0($sp)       # Save $ra on the stack
    # Your code here....
done:
    lw $ra, 0($sp)       # restore $ra
    addi $sp, $sp, 4     # deallocate stack space
    jr $ra               # return
```

Figure 1: Sample termination of a MIPS program in simulators.

```
    .text
    .globl main
main:
    # Your code here....
done:
    li $v0, 10
    syscall              # Execute the 'exit' syscall
```

Figure 2: Another way to terminate a MIPS program in simulators.

Hacettepe Computer Engineering - Spring 2021

BBM234 Computer Organization - MIPS Project

Deadline: Saturday, 10/04/2021 at 23:59:59

Mission: Detecting Cheaters

Considering how Hababam Sınıfı students have proven time and time again how cunning, deceitful, and resourceful they can be, special precautions had to be taken to hide the intention of this cheating detection program, in case they gain unauthorized access to the school's computer and snoop around the files. If they were to understand what the data meant and what its purpose is, they could easily change it to their benefit to avoid detection again. Therefore, Külyutmaz Necmi decided to keep all the data about the students and their scores using covert methods as described in the following paragraphs.

The students wouldn't be stored by their names, but a special IDs were to be given to each student, which would only be known to teacher Külyutmaz. He decided to assign the following IDs to his students:

- İnek Şaban ID: 1
- Güdük Necmi ID: 2
- Damat Ferit ID: 3
- Domdom Ali ID: 4
- Tulum Hayri ID: 5
- Hayta İsmail ID: 6, etc.

Furthermore, the data about the similarity of students' exam papers also requires clandestine storage methods. If students found a file containing only numbers between 0 and 100, they would certainly get suspicious and know it had something to do with the exam scores. Therefore, similarity scores will be encrypted in such a way that the even numbers will be multiplied by 8, while the odd numbers will be divided by 5.

Finally, the encrypted similarity scores will be stored as a row-major 1-D implementation of an upper triangular matrix which holds the similarity scores between each pair of students as shown in Table 1 given below.

Table 1: Similarity report given as an encrypted upper triangular matrix.

Student IDs	**1**	**2**	**3**	**4**
1	#	720	480	80
2	#	#	3	1
3	#	#	#	0
4	#	#	#	#

Your mission is to tackle these problems one at a time, and detect cheaters. You will be given some C code to help you in the process.

Part 1: Arrays Using For Loops - Memory Access

The similarity scores should be defined as an array which will be stored in the memory. You can imagine it as illustrated below:

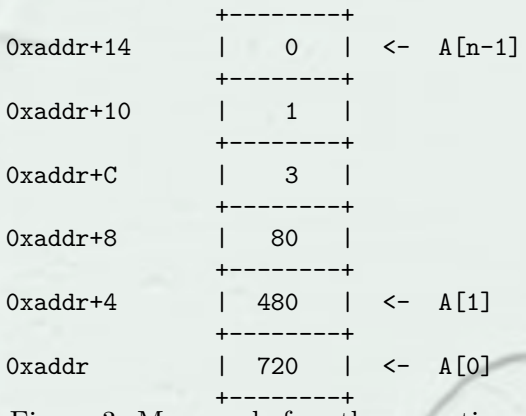

Figure 3: Memory before the execution.

Your program will first have to decrypt the similarity scores saved in the memory. The resulting actual similarity scores **must be saved in the same memory locations as the encrypted ones**.

Consider the C code given below and write its MIPS equivalent.

```c
int datasize; // May vary with each different input
int data[datasize]; // Will be given for each run. Must be stored in memory
for(int i = 0; i < datasize; i++){
    if(data[i]%2 == 0)
        data[i] = data[i]/8; // It is forbidden to use div instructions
    else
        data[i] = data[i]*5; // It is forbidden to use mul/mult instructions
}
```

Figure 4: C code for Part 1.

Important constraints for this part:

- You are not allowed to use multiplication and division instructions (mult, mul, div) for this problem.
- Usage of those instructions will result in penalty of 10 points.

In MIPS an array A can be defined as illustrated in Fig. 5. Here, an array of six elements $\{720, 480, 80, 3, 1, 0\}$ is declared in the memory, and the address of the first element is loaded to register $t0$.

```
    .data
A:  .word 720, 480, 80, 3, 1, 0  #Do not forget a space after a comma!
    .text
    .globl main
main:
    la $t0, A # Load the address of A[0] to register t0
```

Figure 5: Declaring an array in MIPS and loading the address of its first element.

After **la** instruction, the address of the first element of array A will be stored in register $t0$. You can write your remaining code afterwards. Note that the addresses of the next elements are increments by 4 as MIPS is byte-addressable. You can define the array size in the data segment as well.

After your code executes, the change in the memory should reflect as shown below.

```
                +--------+
0xaddr+14       |    0   |  <-  A[n-1]
                +--------+
0xaddr+10       |    5   |
                +--------+
0xaddr+C        |   15   |
                +--------+
0xaddr+8        |   10   |
                +--------+
0xaddr+4        |   60   |  <-  A[1]
                +--------+
0xaddr          |   90   |  <-  A[0]
                +--------+
```

Figure 6: Memory after the execution.

In your report, you should include the code with comments, as well as the before and after screenshots of the Data Segment as shown in your simulator. See Figures 7 and 8 as an example.

Figure 7: Data segment before the execution.

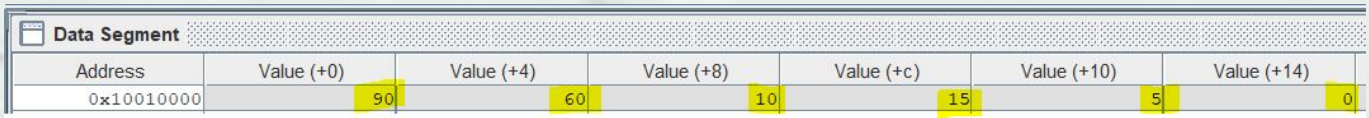

Figure 8: Data segment after the execution.

Part 2: Function Calls

The vice-principal and history teacher Kel Mahmut suggested they also calculate the average similarity score as it will be a good indicator of the overall deceitfulness of the class. However, coding the usual average formula would be too conspicuous, and would definitely catch the attention of possible intruders. So, Külyutmaz decided that it would be better to have a separate function which will calculate the average recursively. This approach would definitely throw any prying student off.

The recursive formula to calculate the average of a given set of n numbers $AVG(n)$ is given in Equation 1 below.

$$AVG(n) = \frac{(AVG(n-1) \times (n-1)) + n^{th}element}{n} \tag{1}$$

Having this formula in mind, you should expand your MIPS program to implement this recursive function that will operate on the decrypted array. The C code to which you may refer to is given in Fig. 9 below.

```c
int average_recursive(int *data, int n){
    int sum, avg;
    if(n == 1)
        sum = data[0];
    else
        sum = data[n-1] + ((n-1) * average_recursive(data, n-1));
    avg = sum / n;
    return avg;
}
```

Figure 9: A recursive function for calculating the average.

The final result of the recursive function should be stored in register $v1$, and output to the console as shown in Figures 10 and 11 respectively. Note that we are dealing with the integers here only, and that the result will not be precise, but only an approximation of the real average. You should disregard the precision issue for this part.

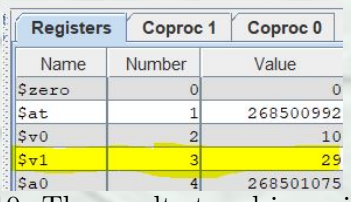

Figure 10: The result stored in register $v1$.

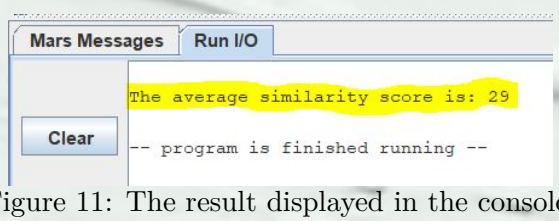

Figure 11: The result displayed in the console.

Important notes and constraints for this part:

- You must use **jal** instruction for function calls. You should also use registers $a0-a3$ for passing arguments, and $v0-v1$ for returning results, as is the convention in MIPS assembly.
- For full credit you MUST implement the recursive function.
- You are allowed to use multiplication and division instructions (mult, mul, div) in this part.
- Implementing a function which is not recursive (one single call to a function that calculates average using a loop) will only get you 50% of credit for this part, provided you use the function call and stack properly. Otherwise 0 credit will be given.
- Calculating the average using only loop without any function calls will not get any credit either.

In your report, you should include the code with comments, as well as the before and after execution screenshots of the registers in your simulator.

BONUS Part 3: Naming the Cheaters [EXTRA CREDIT]

If you wish to get extra credit, implement an additional functionality of detecting who the cheaters are. Assume that there will always be only two cheaters in each test case. **If the similarity score between two students is 80 or higher, then those two students are the cheaters.** Your program should store the cheater's IDs in registers $s6$ and $s7$ and output their names to the console. You may refer to the list of student IDs given at the beginning of the section (ID 1 is İnek Şaban, etc.).

For the sample input given in Table 1, the output should be as shown if Figures 12 and 13 respectively.

$s6	22	1
$s7	23	2

Figure 12: Cheaters' IDs stored in registers.

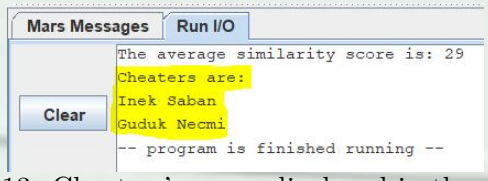

Figure 13: Cheaters' names displayed in the console.

As this is an extra credit part, you will not be given C code to refer to but you may keep the following in mind:

- The similarity scores are stored as a row-major 1-D implementation of an upper triangular matrix.

Grading

- **MIPS code: 90%**
  - Part 1 - Memory (array) access using loops: 50 points
  - Part 2 - Function calls: 50 points
  - Bonus part 3 - Find the cheaters: 10 bonus points
- **Report: 10%**

Testcases & What to Include in the Report

Test for the following two testcases which you will include in your report, but your code will be tested with different testcases as well:

```
Testcase 1:                          Expected output for testcase 1:
-----------                          -------------------------------
num_students = 4                     data[] = {90,60,10,15,5,0}
datasize = 6                         average = 29
data[] = {720, 480, 80, 3, 1, 0}     Bonus: Cheaters IDs: 1 and 2
                                     The cheaters are İnek Şaban and Güdük Necmi
-------------------------------------------------------------------------------

Testcase 2:
-----------
num_students = 6
datasize = 15
data[] = {0, 1, 5, 400, 112, 17, 7, 0, 560, 13, 0, 11, 3, 5, 0}

Expected output for testcase 2:
-------------------------------
data[] = {0, 5, 25, 50, 14, 85, 35, 0, 70, 65, 0, 55, 15, 25, 0}
average = 27
Bonus: Cheaters IDs: 2 and 3, the cheaters are Güdük Necmi and Damat Ferit
-------------------------------------------------------------------------------
```

Figure 14: Testcases to include in the report.

Your report should include the following:

- A cover page with course details, and project title, your name and surname, and student ID.
- A brief problem definition.
- Your solution MIPS **code with comments**. You **must** write comments to explain the purpose of the instructions in your program. Failing to comment each instruction will result in penalty.
- The explanation of some specific parts of your solution should also be included:
  - How you used function calls (**jal** instructions),

  - **How you used stack during function calls and why**.

- The screenshots of the given test cases before and after execution (**you should show both registers and data segment**). The results should be clearly explained. You may explain portions of your results on the screenshots if you wish.
- Don't forget to include ALL NECESSARY TEST CASES in your results.
- References if any.

We encourage you to use the LaTeX report template.

What to Turn In & Deadline

In this assignment, you will write and simulate a MIPS program as described in the instructions. Please turn in the following items in the specified format through the submission system (only .zip files are supported):

- **b<studentID>.zip**
  - hababam.asm
  - report.pdf

Note that submissions via email or any other medium other than the submission system **will NOT be accepted**. The submission deadline is **Saturday, 10/04/2021 at 23:59:59**. Extensions or late submissions **will NOT be allowed**.

Plagiarism Control Notice

Students must implement their solutions individually. All submissions will be submitted to a plagiarism check. Any submissions that show a high level of similarity will be reported as plagiarism attempts to the ethics committee.