CS224 - Spring 2023 - Lab #1 (Version 1: February 19, 21:34)

Creating and Running Simple MIPS Assembly Language Programs

Dates (TAs) - (Tutor(s))

Section 1: Mon, 27 Feb, 8:30-12:20 in EA-Z04 (Deniz, Ece) - (Berkan)
Section 2: Wed, 1 Mar, 13:30-17:20 in EA-Z04 (Sepehr, Pouya) - (Berkan, Ege)
Section 3: Tue, 28 Feb, , 13:30-17:20 in EA-Z04 (Onur, Utku) - (Talay)
Section 4: Fri, 3 Mar, Fri 8:30-12:20 in EA-Z04 (Sanaz, Soheil) - (Eren)
Section 5: Wed, 1 Mar Feb, 8:30-12:20 in EA-Z04 (Navid, Onur) - (Yarkın)
Section 6: Fri, 3 Mar, 13:30-17:20 in EA-Z04 (Sanaz, Soheil) - (Yarkın)

TA Name (email address: @bilkent.edu.tr): Section (hours)

Ece Kunduracıoğlu (e.kunduracioglu@): Section 1 (Mon 08:30-10:30)
Deniz Uzel (deniz.uzel@): : Section 1 (Mon 08:30-10:30)
Navid Ghamari (navid.ghamari@): Section 2 (Wed 13:30-15:30)
Onur Yıldırım (o.yildirim@): Section 3 (Tue 13:30-17:30)
Pouya Ghahramanian (ghahramanian@): Section 2 (Wed 13:30-17:30)
Sanaz Gheibuni (sanaz.gheibuni@): Section 4 (Fri 08:30-12:30), Section 6 (Fri 13:30-15:30)
Sepehr Bakhshi (sepehr.bakhshi@): Section 2 (Wed 13:30-17:30)
Soheil Abadifard (soheil.abadifard@): Section 4 (Fri 08:30-12:30), Section 6 (Fri 13:30-15:30)
Utku Gülgeç (utku.gulgec@): Section 3 (Tue 13:30-17:30)

Tutor Name (email address: @ug.bilkent.edu.tr): Section (hours)

Berkan Şahin (berkan.sahin@): Section 1 (Mon 08:30-10:20) & Section 2 (Wed 13:30-15:20)
Hasan Ege Tunç (hasan.tunc@): Section 2 (Wed 13:30-17:20)
Atak Talay Yücel (talay.yucel@): Section 3 (Tue 13:30-17:20)
Mehmet Eren Balasar (eren.balasar@): Section 4 (Fri 08:30-12:20)
Hasan Yarkın Kurt (yarkin.kurt@): Section 5 (Wed 08:30-10:20) & Section 6 (Fri 13:30-15:20)

Purpose: This lab tries to achieve the following purposes. **1**. Learning CS224 lab rules and procedures. **2.** Introduction to the MIPS assembly language programming and MARS environment. **3.** Learning simple array processing (accessing array elements using load address etc.). **4.** Using multiplication and division and HI and LO registers. **5**. Program debugging in MIPS and principles of sub program writing (jal: -jump and link- and jr -jump register instructions-, use of $v0 and argument registers like $a0).

Preliminary Work: 40 points

1. Array operations (40 points)

Lab Work: 60 points

1. Using MARS with "hello world" (15 points)
2. Fibonacci program debugging (jal: jump and link, jr: jump register instructions) (25 points)
3. Implementing an arithmetic expression given in lab (20 points)

4. Implementing a simple menu with loops (20 points)

Important Notes for All Labs About Attendance, Performing and Presenting the Work

1. You are obliged to read this document word by word and are responsible for the mistakes you make by not following the rules.
2. Not attending to the lab means 0 out of 100 for that lab. If you attend the lab but do not submit the preliminary part you will lose only the points for the preliminary part.
3. Try to complete the lab part at home before coming to the lab. Make sure that you show your work to your TAs and answer their questions to show that you know what you are doing before uploading your lab work and follow the instructions of your TAs.
4. In all labs if you are not told you may assume that inputs are correct.
5. In all labs when needed you have to provide a simple user interface for inputs and outputs.
6. Presentation of your work
   You have to provide a neat presentation prepared in txt form. Your programs must be easy to understand and well structured.
   Provide following six lines at the top of your submission for preliminary and lab work (make sure that you include the course no. CS224, important for ABET documentation).

   CS224
   Lab No.
   Section No.
   Your Full Name
   Bilkent ID
   Date

   Please also make sure that your work is identifiable: In terms of which program corresponds to which part of the lab.
7. **If we suspect that there is cheating we will send the work with the names of the students to the university disciplinary committee. You cannot use ChatGPT code, it is classified as plagiarism. Note that MOSS is capable of detecting ChatGPT code.**
8. **The lab is prepared with the assumption that it will be face to face. It will be updated according to the latest requirements of YÖK.**

DUE DATE PRELIMINARY WORK: SAME FOR ALL SECTIONS

No late submission will be accepted. Please do not try to break this rule and any other rule we set.

a. Please upload your programs of preliminary work to Moodle by 8:30 on Monday Feb 27, 2023.

b. Please note that the submission closes sharp at 8:30 and no late submissions will be accepted. You can make resubmissions so do not wait for the last moment. Submit your work earlier and change your submitted work if necessary. Note that only the last submission will be graded.

c. Please familiarize yourself with the Moodle course interface, find the submission entry early, and avoid sending an email like "I cannot see the submission ınterface." (As of now it is not yet opened.)

d. Do not send your work by email attachment they will not be processed. They have to be in the Moodle system to be processed.

e. Use filename **StudentID_FirstName_LastName_SecNo_PRELIM_LabNo.txt** Only a NOTEPAD FILE (txt file) is accepted. Any other form of submission receives 0 (zero).

DUE DATE PART LAB WORK: (different for each section) YOUR LAB DAY

a. You have to demonstrate your lab work to your TA for grading. Do this by **12:00** in the morning lab and by **17:00** in the afternoon lab. Your TAs may give further instructions on this. If you wait idly and show your work last minute, your work may not be graded.

b. At the conclusion of the demo for getting your grade, you will **upload your Lab Work** to the Moodle Assignment, for similarity testing by MOSS. See below for the details of lab work submission.

c. Try to finish all of your lab work before coming to the lab, but make sure that you upload your work after making sure that it is analyzed by your TA and/or you are given the permission by your TA to upload.

Part 1. Preliminary Work (40 points)

1. Simple Array Operations (40 points)

Create an array of maximum size of 20 elements. Ask the user to enter the number of elements and then enter the elements one by one. Perform the following operations by providing a menu (no GUI just comment based menu) to the user.

For array storage allocation use

```
        .data
array:  .space   80  # Allocate 80 bytes = space enough to hold 20 words
```

After initializing the array ask the user input a integer number (N). Following that

a. Find the number of array members equal to N.

b. Find the number of array members less than N.
   Find the number of array members greater than N.

c. Find the number of elements evenly divisible by N. (See the lab part below for div: divide, mflo: move from lo, mfhi: move from hi instructions)

   Display the above results with proper messages.

Part 2. Lab Work (60 points)

1. Using MARS, a MIPS simulator (15 points, A, B, C each part: 5 points)

A. Examine the controls and options in MARS

1. Open the MARS simulator. Take a few minutes to explore each of the windows (Edit & Execute; MARS Messages & Run I/O; Registers, etc.). Look at each of the controls on the pull-down menus from the task bar at the top (File, Edit, Run, Settings, Tools, Help) and discuss what you think it does. Change the run speed to 1 instuction/sec using the slide bar at the top.

2. Now, starting from the left, slowly put the mouse over each of the icons in the row across the top, to see the action that it will cause (but don't click the icon). Determine which actions, represented by the icons, also can also be selected from a pull-down menu. Click the "?" icon (or choose it from the Help pull-down menu) and in the Help window that open, click each of the tabs and briefly look at the Help contents for that topic. Note that the MIPS tab offers a box with scroll bar, and 6 tabs of its own. Similarly, the MARS tab opens a window with 8 tabs. Be sure to look at each of these.

3. On the top menu, Tools offers a list of tools in the pull-down menu. Open each of these and look at it briefly to begin to understand the range of additional capabilities of MARS

B. Using a simple program in MARS

4. Load in Program1 (Generates "hello world", see the program in the Moodle folder), using File > Open, or the appropriate icon button. In the Edit window, examine this program, and learn what it does. Try to understand how it does it.

5. Assemble the program, using Run > Assemble , or the appropriate icon button. In the Execute window, examine the code portion of memory (called Text Segment) and the data portion of memory (called the Data Segment). Note the beginning address of each, and the contents of each. Knowing that 2 hex characters represent one byte, and remembering that ASCII characters are each coded as one byte, determine which characters are stored in which locations in the Data Segment. Examine the initial values of the registers—which ones are non-zero? Make a note of their values. Read the messages in the MARS Messages window. Check the Run I/O window.

6. Set the Run Speed bar to 1 instruction/second. Now run the assembled program, using Run > Go, or the appropriate icon button. What happens to the yellow highlight bar in the Text Segment during execution? What is written in the MARS Messages window? In the Run I/O window? Compare the final values of the non-zero registers—did you expect to find $1, $4, and $2 changed by the program? What is the final value of the $pc register?

7. Now edit the Program1 so that it produces a different output: Hello <name of your TA> Choose one of the TAs names or the Tutor's name, and make it print out that name. Then call that TA or Tutor over to show your output.

C. Finding and fixing errors in MARS

8. Load in Program2 (Conversion program), assemble it, and run it, providing the necessary input from the keyboard. What does this program do? Examine the MIPS assembly program to understand how it does it.

9. Edit the program by changing li $v0, 5 to li v0, 5 . Then assemble it and read the error message. Then fix the error and re-assemble it.

10. Edit the program by changing li $v0, 5 to $li v0 . Then assemble it and read the error message. Then fix the error and re-assemble it. Now run the program, at 1 instr/sec, and make note of the value of $v0 after the 2nd syscall is completed.

11. Edit the program by changing mul $t0, $v0, 9 to mul $t0, $t0, 9 . Then assemble it and run it, and explain why the output value is what it is. Then fix the error and re-assemble it and re-run it, to verify that Program2 again works correctly.

12. Change the Run speed back up to the maximum. Assemble the program, and instead of hitting the normal Run button, instead hit the "Run 1 step at a time" icon (looks like >1) repeatedly, to single-step through the program. Notice that you can go slowly, examining the memory and registers after each step, or go quickly. This single-step mode is good for debugging. Explain to the TA or Tutor how you could use it to help find a logical error or typo error that accidently passed the assembler's syntax check.

2. Fibonacci program: Using breakpoints in MARS, fixing logic errors and using jal and jr instructions (25 points)

Load in Program3 (Fibonacci program), which says it is a fibonacci number finder, implemented using a loop (iteratively, not recursively). The program has several errors, syntax and logical, that you must find and fix. After getting it to assemble correctly, note that it runs, but gives the wrong value of fib(7), which should be 13. (**Study and learn jal and jr instructions.**) Fibonacci: https://en.wikipedia.org/wiki/Fibonacci.

To find and fix logical errors, you need to go through the code determining what the values are of the critical registers at the places in the program where they are changed. In long programs, and especially programs with loops, it would take too long to single-step through the whole program. Instead, you must set breakpoints, and run up to those points and stop. Since the value of $v0 is where the fibonacci number is being accumulated in this program, you should set a breakpoint in the fib function wherever $v0 is changed. This way you can look at its value and determine if it is being calculated correctly. To set

a breakpoint at an instruction, check the Bkpt box in the left-hand column next to the instructions you want to break at. Then hit Run.

[Hint: if you are not sure if the state of the machine at the breakpoint will include the effect of that instruction or not, you should experiment and learn by setting breakpoints in pairs, both at an instruction of interest, and after it, to see when the actual change in values takes place].

When you have the Program3 debugged and running correctly, call the TA or Tutor to show how breakpoints work, and show that it calculates any Fibonnacci number.

3. Using MIPS for mathematical calculations (20 points)

Write a program that prompts the user for one or more integer input values, reads these values from the keyboard, and computes the following mathematical formula: (***it will be given on the board by the TA***) When the computation is finished, the program should print the result along with an explanatory comment to the user via the display.

Note that the mathematical formula will possibly involve all four operations and the mod operator.

Try your program with the following expression x = (2 *a - b) / (a + b)

Study mult instruction and understand hi and lo registers

```
mult    $t0, $t1        # The result goes to lo register
mflo    $t2             # $t2= $t0 x $t1
```

Study div instruction and understand hi and lo registers

```
div     $t0, $t1
mflo    $t3             # $t3= $t0 / $t1
mfhi    $t4             # $t4= $t0 mod $t1
```

Part 3. Submit Lab Work for MOSS Similarity Testing

1. Submit your Lab Work MIPS codes for similarity testing to Moodle.
2. You will upload one file. Use filename **StudentID_FirstName_LastName_SecNo_LAB_LabNo.txt**
3. Only a NOTEPAD FILE (txt file) is accepted. No txt file upload means you get 0 from the lab. Please note that we have several students and efficiency is important.
4. *Even if you didn't finish, or didn't get the MIPS codes working, you must submit your code to the Moodle Assignment for similarity checking.*
5. Your codes will be compared against all the other codes in the class, by the MOSS program, to determine how similar it is (as an indication of plagiarism). So be sure that the code you submit is code that you actually wrote yourself ! You are not allowed to use web resources that solves the assigned programs.

6. The effectiveness of MOSS for chatbot code is quite good, with a detection rate of much higher than 50%. (The answer is provided by chatbot.)

Part 4. Cleanup

1. After saving any files that you might want to have in the future to your own storage device, erase all the files you created from the computer in the lab.
2. When applicable put back all the hardware, boards, wires, tools, etc where they came from.
3. Clean up your lab desk, to leave it completely clean and ready for the next group who will come.

LAB POLICIES

1. You can do the lab only in your section. Missing your section time and doing in another day is not allowed.
2. The questions asked by the TA will have an effect on your lab score.
3. Lab score will be reduced to 0 if the code is not submitted for similarity testing, or if it is plagiarized. MOSS-testing will be done, to determine similarity rates. Trivial changes to code will not hide plagiarism from MOSS—the algorithm is quite sophisticated and powerful works for ChatGPT code too. Please also note that obviously you should not use any program available on the web, or in a book, etc. since MOSS will find it. The use of the ideas we discussed in the classroom is not a problem.
4. You must be in lab, working on the lab, from the time lab starts until your work is finished and you leave.
5. No cell phone usage during lab.
6. Internet usage is permitted only to lab-related technical sites.
7. **For the Earthquake related cases we follow the policies set by the university administration. Note that they require official documentation. If applicable to you please follow the rules set by the university.**