CS224 - Fall 2020 - Lab #1 (Version 1: October 1, 16:40)

Creating and Running Simple MIPS Assembly Language Programs

Dates:

Section 1: Wednesday, 7 October, 13:30-16:20 in EA-Z04, Only odd no. students may come to the lab
Section 2: Friday, 9 October, 13:30-16:20 in EA-Z04, Only odd no. students may come to the lab
Section 3: Sunday, 11 October, 8:30-11:20 in EA-Z04, Only odd no. students may come to the lab
Section 4: Thusday, 8 October, 13:30-16:20 in EA-Z04, Only even no. students may come to the lab

TAs:

Section 1 (Wed): Ege Berkay Gülcan, Zülal Bingöl
Section 2 (Fri): Pouya Ghahramanian, Yusuf Dalva
Section 3 (Sun): Yusuf Dalva, Zülal Bingöl
Section 4 (Thu): Berkay Gülcan, Eren Çalık

TA email Addresses:

Berkay Gülcan: berkay.gulcan@bilkent.edu.tr
Eren Çalık: eren.calik@bilkent.edu.tr
Pouya Ghahramanian: ghahramanian@bilkent.edu.tr
Yusuf Dalva: yusuf.dalva@bilkent.edu.tr
Zülal Bingöl: zulal.bingol@bilkent.edu.tr

Lab Attendance Policy:

All labs can be done online. Your id cards can be used to enter to the lab. If you want to come to the physical lab please follow the even/odd day policy: On even numbered days students with an even number student ID can come to the lab. Follow the same for odd numbered days.

Purpose: **1**. Introduction to MIPS assembly language programming and MARS environment (please see the self study part given at the end of this document). **2.** Learning simple array processing, sending getting values from subprograms, and using multiplication and division and HI and LO registers. **3**. Learning CS224 lab rules and procedures.

You are obliged to read this document word by word and are responsible for the mistakes you make by not following the rules. Your programs should be reasonably documented must have a neat syntax in terms of variable names, comments and spacing as you have learned in CS101 and as suggested by the programs in the textbook.

Summary

Preliminary Work:

Part 1 (25 points): Simple array operations, inputting values and checking if it is a palindrome.
Part 2 (25 points): Arithmetic expression calculation using a subprogram.
Due date of this part is the same for all groups.

Lab Work:

Part 3 (25 points): Using breakpoints, fixing errors in a Fibonacci program; using jal (jump and link) and jr (jump register) instructions.

Part 4 (25 points): Implementing an arithmetic expression.

Note try, study, and try to complete lab part at home before coming to the virtual/physical lab.

DUE DATE OF PART 1 & 2 (PRELIMINARY WORK): SAME FOR ALL SECTIONS

No late submission will be accepted.

a. Please upload your programs of preliminary work to Moodle by 17:00 on Tuesday October 6. You may be given further instruction for uploading please check Moodle course interface.
b. Please **upload your programs of Part 1 & 2 (PRELIMINARY WORK)** to Moodle by 17:00 on Tuesday October 6. Use filename **StudentID_FirstName_LastName_SecNo_PRELIM_LabNo.txt** Upload only the programs. Only a NOTEPAD FILE (txt file) is accepted. Any other form of submission receives 0 (zero).

DUE DATE PART 3 & 4 (LAB WORK): (different for each section) YOUR LAB DAY

a. You have to demonstrate (e.g. by using chat on Zoom) your lab work to the TA for grade by **11:00** in the morning lab and by **16:00** in the afternoon lab. Your TAs may give further instructions on this. If you wait idly and show your work last minute, minimum 20 points will be taken off from your grade. We will see it together how it works with Zoom.
b. At the conclusion of the demo for getting your grade, you will **upload your entire program work including Part 1 & 2 (Preliminary Work)** to the Moodle Assignment, for similarity testing by MOSS. See Part 5 below for details.

If we suspect that there is cheating we will send the work with the names of the students to the university disciplinary committee.

Part 1 & 2. Preliminary Work (50 points)

You have to provide a neat presentation prepared in txt form. Provide following five 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

Make sure that you identify what is what at the beginning of each program by using proper comments.

Important Note - Assume that all Inputs are Correct: In all lab works if it is not explicitly stated you may assume that program inputs are correct.

Part 1. (25 points) Writing a MIPS program that

- Initializes an array of maximum size of 20 elements: Do this by asking the user first the number of elements and then by getting the array elements one by one from the user.
- Displays array contents.
- Decides if it is a palindrome array or not. Palindrome array is an array where its numbers are in the same order backward and forward like {1, 2, 2, 1} or {1, 2, 3, 2, 1}.

Note that maximum array size is fixed and 20 elements which can be achieved by a declaration like as follows

```
array:    .space   80      # 20 words each one is 4 bytes.
```

Part 2. (25 points) Writing a MIPS program that

- implements the following arithmetic expression for integer numbers.
  x= a * (b - c) % d

In your program

- Read a, b, c, d in the main program (main program: the part that the execution begins).
- Pass a to d to a subprogram using the argument registers $a0, $a1, $a2, and $a3.
- Perform the computation in the subprogram using $t (temporary) registers and return the result to the caller/main program in $v0.
- Print the result in main.
- In doing these provide a reasonable user interface for input and output.

Table 1. Explanation for divide and multiplication instructions.

Instruction	How it Works
div $t1,$t2	Divides $t1 by $t2 then set LO to quotient and HI to remainder (use mfhi to access HI, mflo to access LO)
mult $t1,$t2	Multiplies $t1 and $t2 and sets HI to high-order 32 bits, LO to low-order 32 bits of the product of $t1 and $t2 (use mfhi to access hi, mflo to access lo)
mfhi $t1	Moves from HI register : Set $t1 to contents of HI
mflo $t1	Moves from LO register : Set $t1 to contents of LO

Part 3 & 4. Lab Work (50 points)

Part 3. 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.

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]. Or pay attention to the value of the PC where it points to the instruction to be executed.

Part 4. Using MIPS for Mathematical Calculations (25 Points)

Write a program that prompts the user for one or more integer input values, reads these values from the keyboard, and computes a mathematical formula such as (***a different formula may be given by your TA***):

A= (B / C + D * B - C) Mod B

When the computation is finished, the program should print the result along with an explanatory comment to the user via the display.

Part 5. Submit Your Code for MOSS Similarity Testing

1. Submit your 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. Be sure that the file contains exactly and only the codes which are specifically detailed Part 1 to Part 4, yes including Part 1 preliminary work programs (your paper submission for preliminary work must match MOSS submission). Check the specifications! *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 !

Part 6. Cleanup for Physical Lab

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. For people who are 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.

Part 7. Lab Policies for Physical Lab

1. You can do the lab only in your section. Missing your section time and doing in another day is not allowed.
2. Students will earn their own individual lab grade. The questions asked by the TA will have an effect on your individual 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. 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.

SELF STUDY: ASSEMBLING AND DEBUGGING A PROGRAM

A. Examining 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, Coproc 1 and Coproc 2). 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"), 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 <a different name> , and make it print out that name.

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.