CS224 - Fall 2019 - Lab #1 (Version 2: October 2, 19:35)

Creating and Running Simple MIPS Assembly Language Programs

Dates: Section 1, Wednesday, 9 October, 13:40-17:30
Section 2, Friday, 11 October, 13:40-17:30
Section 3, Wednesday, 9 October, 8:40-12:30
Section 4, Thursday, 10 October, 13:40-17:30
Section 1, 2, 3, 4 Prelim. Report Deadline: Wednesday October 9, 10:40
Lab Location: EA-Z04

NO LATE SUBMISSION IS ACCEPTED

Purpose: **1**. Introduction to MIPS assembly language programming and MARS environment. **2**. 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 and must have a neat syntax in terms of variable names and spacing. In all labs if it is not specified assume that inputs are always correct.

Summary
Preliminary Report/Preliminary Design Report:
Part 1 (30 points): array, palindrome, division without division instruction, code generation and defining some terms. (Due date of this part is the same for all groups).

Lab: (Study lab part at home and try to finish it before coming to the lab. Make sure that you show the lab work to your TA before uploading in the lab.)
Part 2 (15 points): Using MARS with "hello world".
Part 3 (20 points): Using breakpoints, fixing errors in a Fibonacci program
using jal (jump and link) and jr (jump register) instructions.
Part 4 (15 points): Implementing an arithmetic expression given in lab.
Part 5 (20 points): Implementing a simple menu with loops.

DUE DATE OF PART 1: SAME FOR ALL SECTIONS:

a. Please bring and drop your hardcopy (printed copy) of the preliminary work into the box(es) provided in the lab by 10:40 on Wednesday October 9.

Please **upload your programs of Part 1 (PRELIMINARY WORK)** to the Unilica by 10:40 on Wednesday October 9. Use filename **StudentID_FirstName_LastName_SecNo_PRELIM_LabNo.txt** Upload only the programs. Only a NOTEPAD FILE (txt file) is accepted. Your paper submission for preliminary work must match MOSS submission. Any format other than txt gets 0 points.

b. To get credit for preliminary work you have to submit its hard copy and upload its txt version to unilica. **No late submission will be accepted**.

DUE DATE PART 2-5: (different for each section) YOUR LAB DAY

You have to demonstrate your lab work to the TA for grade by **12:15** in the morning lab and by **17:15** in the afternoon lab. Your TAs may give further instructions on this. If you wait idly and show your work last minute, 20 points may be taken off from your grade.

At the conclusion of the demo for getting your grade, you will **upload your entire program work** to the Unilica Assignment, for similarity testing by MOSS. See Part 6 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. Preliminary Work / Preliminary Design Report
(30 points, contains sections 1 to 5 & each section 6 points)

You have to provide a neat presentation prepared by Word or a word processor with similar output quality such as latex. At the top of the paper on left provide the following information and staple all papers. In this part provide the program listings with proper identification (please make sure that this info is there for proper grading of your work, otherwise some points will be taken off).

CS224
Section No.: Enter your section no. here
Fall 2019
Lab No. 1
Your Full Name/Bilkent ID

1. Input an Array: Write a MIPS program that
Creates an array of maximum size of 20 elements that asks the user first the number of elements and then enters the elements one by one.
Displays array contents
Reverses the array contents and display the array (for example 1, 2, 3 becomes 3, 2, 1).

2. Palindrome: Write a MIPS program that
Read an integer array as defined above and check if it is a palindrome. (For example, an array with four elements like 1, 4, 4, 1 is a palindrome: the order of the elements from the beginning to the end and from the end to the beginning are the same.

3. Perform Division Without Division Instruction: Write a MIPS program that
Inputs two integer numbers n1 and n2 and performs integer division n1/n2 without using division instruction and display both the division and the remainder of the division.

4. Object Code Generation: Generate the object code for the following instructions first in binary then give it in hex.

```
add     $t0, $t1, $t2
addi    $s0, $s3, 15
mult    $a0, $a1
sw      $t1, 8($t2)
lw      $t2, 8($t1)
```

5. Define Terms: Define the following terms and provide an answer etc as described.

a. Symbolic machine instruction: give two examples

b. Machine instruction: give two examples and write their symbolic equivalents

c. Assembler directive: give two examples.

d. Pseudo instruction: give two examples and provide its implementation using real instructions. Note in your answer only use symbolic machine instructions.

Part 2. 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, 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 <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.

Part 3. Using breakpoints in MARS, fixing logic errors and using jal and jr instructions (20 points)

A. Fix the errors of the Fibonacci Program

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 caluclated 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.

B. Extend the Fibonacci Program

Extend the corrected Fibonacci program such that it gets a positive input integer number if it is an odd number like 7 it finds the corresponding 7th Fibonacci number, if it is an even number like 4 it finds the factorial of that number like 4!

Part 4. Using MIPS for mathematical calculations Program 4 (15 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 will be given on the board by the TA that includes all four operations and Mod***):

$$X= (A * B + C - D) \text{ Mod } E$$

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

Part 5. Using MIPS for implementing a program with a simple menu that involve loops Program 5 (20 points)

Create an array of maximum size of 100 elements. Ask the user to enter the number of elements and then the elements one by one. Perform the following operations by providing a menu (no GUI just comment based menu) to the user. Implement the following in the form of separate subprograms. The use of $s registers is optional

a. Find summation of numbers stored in the array which is less than an input number.
b. Find summation of numbers out of a value range specified by two numbers and display that value.
c. Display the number of occurrences of the array elements divisible by a certain input number.
d. Quit.

Optional: Use syscall with 9 in $v0 for dynamic storage allocation for the array.

Part 6. Submit your code for MOSS similarity testing

1. Submit your MIPS codes for similarity testing to the Unilica > Assignment specific for your section.
2. You will upload one file. Use filename **StudentID_FirstName_LastName_SecNo_LAB_LabNo.txt**
3. Upload only the programs.
4. Only a NOTEPAD FILE (txt file) is accepted.
5. Be sure that the file contains exactly and only the codes which are specifically detailed Part 1 to Part 6, including Part 1 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 Unilica Assignment for similarity checking.*
6. 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!
7. All students must upload their code to Unilica > Assignment while the TA watches.
8. Submissions made without the TA observing will be deleted, resulting in a lab score of 0.

Part 7. 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.