CS224 - Fall 2021- Lab #1 (Version 1: Oct. 6, 10:52 pm)

Creating and Running Simple MIPS Assembly Language Programs

Dates: Section 1, Wednesday, 13 October, 13:30-17:20
Section 2, Thursday, 14 October, 13:30-17:20
Section 3, Wednesday, 13 October, 8:30-12:20
Section 4, Friday, 15 October, 13:30-17:20
Lab Location: EA-Z04

TAs:

Section 1: Kenan Çağrı Hırlak, Pouya Ghahramanian
Section 2: Kenan Çağrı Hırlak, Soheil Abadifard
Section 3: Ege Berkay Gülcan, Sepehr Bakhshi
Section 4: Alper Şahıstan, Soheil Abadifard

TA name (x No of labs) email address:

Alper Şahıstan (x1): alper.sahistan@bilkent.edu.tr
Ege Berkay Gülcan (x1): berkay.gulcan@bilkent.edu.tr
Kenan Çağrı Hırlak (x2): cagri.hirlak@bilkent.edu.tr
Pouya Ghahramanian (x1): ghahramanian@bilkent.edu.tr
Sepehr Bakhshi (x1): sepehr.bakhshi@bilkent.edu.tr

Purpose: to learn how to write, edit, debug and run simple MIPS assembly language programs, using MARS[1].

Summary

Preliminary Work: 30 points

1. Create and array and reverse it (10 points)
2. Check if an array is a palindrome (10 points)
3. Calculate an arithmetic expression (10 points)

Lab Work: 70 points

1. Using MARS with "hello world" (15 points)
2. Fixing errors in a Fibonacci program (jal: jump and link, jr: jump register instructions) (20 points)
3. Implementing an arithmetic expression given in lab (15 points)
4. Implementing a simple menu with loops (20 points)

Important Note for All Labs

1. Try to complete the lab part at home before coming to the lab. Make sure that you show your work to your TA and answer his questions to show that you know what you are doing before uploading

[1] For another tool like MARS see https://arxiv.org/pdf/2104.09502v3.pdf.

your lab work or follow the instructions of your TAs. In all labs if you are not told you may assume that inputs are corrects. For all works when needed please provide a simple user interface for inputs and outputs.

2. 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 (purpose etc.) and must have a neat syntax in terms of variable names, comments, blank lines etc.

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

DUE DATE OF PRELIMINARY WORK: SAME FOR ALL SECTIONS

No late submission will be accepted.

a. Please upload your programs of preliminary work to Moodle by 9:30 am on Wednesday Oct 13 for similarity testing by MOSS.

b. For preliminary work upload a file that includes all programs in proper order with the filename **StudentID_FirstName_LastName_SecNo_PRELIM_LabNo.txt** Only a NOTEPAD FILE (txt file) is accepted. Any other form of submission receives 0 (zero).

c. Note that the Moodle submission closes sharp at 9:30 am and no late submissions will be accepted. You can make resubmissions before the system closes, 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.

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.

DUE DATE OF 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 Part 3 & 4 & 5** to the Moodle Assignment, for similarity testing by MOSS. See Part 6 below for details.

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

d. We use zoom to track your lab attendance.

Part 1. Preliminary Work

1. **CreateArrayAndReverse (10 points)**: It creates an array of maximum size of 20 elements (allocate memory using: .space 80) that asks the user 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. **CheckIfPalindrome (10 points)**: 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 and checks if it is a palindrome (an array with 0 or 1 entry is by definition a palindrome, and for example the array containing 1, 2, 2, 1 is a palindrome, etc.).

3. **ArithmeticExpression (10 points)**: Implement the following expression without using div.

x= (a x (b - c)) % 8

Part 2. Lab Work

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

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.

3. Using MIPS for mathematical calculations (15 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 = (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
```

4. Using MIPS for implementing a program with a simple menu that involve loop (20 points)

Create an array of maximum size of 100 elements (400 bytes). 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.

a. Find summation of numbers stored in the array which is greater than an input number.
b. Find summation of even and odd numbers and display them.
c. Display the number of occurrences of the array elements divisible by a certain input number.
d. Quit.

Optional: Is it possible to have arrays with a size determined during execution time according to the desired array size. If so learn it and do it in that way too. Hint. Look at available syscalls.

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 !

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. 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.
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.