CS224 – Fall 2017 - Lab #1

Creating and Running Simple MIPS Assembly Language Programs

Dates:

Section 1:	Wed, Oct. 4th	13:40-17:30 in EA-Z04
Section 2:	Fri, Oct. 6th	13:40-17:30 in EA-Z04
Section 3:	Wed, Oct. 4th	08:40-12:30 in EA-Z04
Section 4:	Thu, Oct. 5th	13:40-17:30 in EA-Z04

Purpose: to learn how to write, edit, debug and run simple MIPS assembly language programs, using MARS, a MIPS simulator.

Summary

Part 1 (30 points): Preliminary Report/Preliminary Design Report: array, palindrome, arith. expression and object code generation for la (Due date of this part is the same for all groups)

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 Dear students please bring and drop your preliminary work into the box provided in front of the lab before 3:59 pm on Tuesday October 3rd. For late submissions there will be a substantial penalty.

LAB WORK SUBMISSION TIMING: You have to show your lab work to your TA by **12:15** in the morning lab and by **17:15** in the afternoon lab. Note that you cannot wait for the last moment to do this. If you wait for the last moment and show your work after the deadline time 20 points will be taken off.

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 (for some Latex is the science fiction editor for undergraduates, you may challenge yourself by trying it). 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 2017
Lab No.
Your Full Name/Bilkent ID

1. 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
Calculates the average of the array contents. For each array element, display the difference from the average. (for example if values are 1, 2, 3; average is 2 and it displays -1, 0, 1).

2. Write a MIPS program that
Gets an input string and checks if it is a palindrome. (Study load byte, store byte and some other instructions if need).

3. Write a MIPS program that
Implements the following expression without using div. If necessary use instructions other than we have seen in the class.

a= (x - y) % 4

4. Generate the object code (in hex) for the following la instructions. Show your work for the intermediate steps (please remember that la is a pseudo instruction and implemented by two instructions).

```
        la      $s0, z
        la      $t1, t
        .....
        .data
        .space  40
z:      .word   5,4,3,2,1
t:      .word   1,2,3
```

5. Write a simple program (both .text and .data segments) which will sequentially make operations (arithmetic, logical, shift and load-store) on data values, using all the MIPS operations that have been introduced in the first 2 weeks of class. You may put them in any order, and may use any registers and memory and small constants (immediate values) as needed. But your MIPS program should meet the following conditions:

-- it must never create a result of 0 at any step

-- it must load a 32-bit constant "immediate" into register at some point, and use the value in the calculation

-- it must use a 32-bit value from memory as part of the calculation (***Pick a proper memory location***.)

--it must prompt the user for an input with a message, and use that value as part of the calculation

--at the end, it must output "The final result is:" followed by the value of the result (on the same line)

[Hint: use MARS Help to learn about how to use syscalls, esp. for getting input and output values, printing messages, stopping the program, and to learn about assembler directives (like .text and .word and more) and to learn about pseudo-instructions, esp. **la** and **li**.)]

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)

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.

Part 4. Using MIPS for mathematical calculations Program 4 (15 points)

Write a program that prompts the user for an integer input value, reads this value 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 involve all four operations and the mod operator.

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

Now modify the program, so that it prompts for and accepts an integer input N, and runs the calculation on N, and *repeats this over and over* until a sentinel value is inputted. Let the sentinel value be 0, so that all integers for N except zero are accepted and used. But if zero is entered, the program will terminate, ending by printing a goodbye message.

Part 6. Submit your code for MOSS similarity testing

Submit your MIPS codes for similarity testing to the Unilica > Assignment specific for your section. You will upload one file: **name_surname_SecNo_MIPS.txt** created in the relevant parts. 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.* 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 ! All students must upload their code to Unilica > Assignment while the TA watches. Submissions made without the TA observing will be deleted, resulting in a lab score of 0.

Part 6. 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.
7. For labs that involve hardware for design you will always use the same board provided to you by the lab engineer.