CS224 Lab #1

Creating and Running Simple MIPS Assembly Language Programs

Dates:	Section 1 Monday	15 February	08:40-12:30
	Section 2 Friday	19 February	13:40-17:30
	Section 3 Tuesday	16 February	08:40-12:30

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

Part 1. Exercising MIPS in a non-branching non-jumping program with user I/O (30 points)

1. 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 up till now in 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 create a 32-bit large constant "immediate" and put it into register at some point, and use the value in the calculation
-- it must load and use a 32-bit value from memory as part of the calculation (***The TA will specify on the board where in memory to locate this value***.)
--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**.)]

2. Use MARS (editor, assembler, and simulator for execution) to make your program run. Run your program in single-step mode, advancing one instruction at a time under your control. As your program is executing 1 step at a time, look at the values in the registers at each step, including the PC register, and also at the relevant memory locations, to follow and verify its execution. Does it behave as you expected? Why or why not? When it is working correctly, call the TA or Tutor, show your program execution in single-step mode, and explain what the values are as it runs. Now save this MIPS source code, you will need it later. Save it with a filename that includes your name: name_surname_MIPS.txt .

3. While he/she is there, the TA or Tutor will choose 2 lines of the assembly code you have generated (in the column named "Basic"), and the corresponding machine code (in the column named "Code"), and ask you to hand-translate and explain why they are equivalent. [Hint: the register codes are found in Table 6.1, the opcodes and function codes are found on the MIPS Green Card (in Unilica, under

Documents > Resources > MIPS), but everything about MIPS can be found on the Internet (Wikipedia, etc)]

Part 2. Using MIPS for mathematical calculations in a loop (30 points)

4. Now write a more sophisticsated program (including both .text and .data segments) 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***) Your MIPS program must perform the calculation iteratively, using a loop. Your loop must use exactly 1 beq or bne for the test and branch instruction—no other branching or jumping instruction is permitted. In MIPS, test and branch are combined together. For example, beq $t0, $t1, Label means "if the value in $t0 equals the value in $t1, then branch to the instruction at Label". See the MIPS Help for more on the branch instructions. When the computation is finished, the program should print the result along with an explanatory comment to the user via the display.

5. Test your program to make sure it gives correct results for various values of inputs. What happens if one or more inputs are 0? If negative? If very very large? Experiment to see what happens if you try to give non-integer inputs when prompted, such as characters, strings, real numbers, and arithmetic expressions. If unexpected results occur, trace the exection using single-step and breakpoint modes to understand what is happening and why.

6. When your program is working correctly and robustly (no matter what the inputs), call the TA or Tutor to show your program and explain how it calculates and displays the result.

7. Now modify the program, so that it prompts for and accepts the input integers, and runs the calculation using these integers, and *repeats this over and over* until a sentinel value is inputted. Let the sentinel value be 0, so that all integers except zero are accepted and used. But if zero is entered, the program will terminate, ending by printing a goodbye message. [Hint: this program will require a second loop, the assembly language equivalent of *while (input != 0)*].

Show this program, properly working, to the TA or Tutor. Now save the program of part 7, adding it to the name_surname_MIPS.txt file that you created earlier. The file now contains 2 MIPS assembly programs you wrote. Save this MIPS.txt file, you will need it later.

Part 3. Using MIPS for sorting and searching (40 points)

8. Write a MIPS program that does the following: gets N integers from the user via keyboard input, sorts the integers using any sorting algorithm you choose, then searches and finds what the user asks for and gives it as output.

    --the number of inputs N is a variable, so the user should be prompted and respond with a positive integer for N. Then the user is prompted for the values of each of the N inputs. All must be integers for the sort to work, so your program must not accept non-integers. When the

inputting is done, give the user a message. [Note: since the number of inputs is known only at runtime, they should be stored in dynamic data memory. Stack pushing would be convenient, but you may choose to store them in heap instead. A syscall to obtain heap memory can be used for this.]
--perform a sort on the integers, to order them from smallest to largest. Any sort algorithm can do this, but it would be good to explore "sorting algorithms" in Wikipedia to learn what the range of choices are, and what their advantages and disadvantages are. Use of an already-written sorting algorithm implemented in MIPS is allowed. Although MOSS will find this code portion that you didn't write yourself, it will be allowed and you will be exempt from plaigarism.
--tell the user when the numbers are sorted, giving a message. Then ask the user to choose an output from among the choices: mean value, median value, smallest value, largest value, average of {smallest, median, and largest values}, 3rd value*, 2nd-to-last value*, number of negative values, average of lower half-list*, average of upper half-list*, none of these. Until the user selects "None of these", keep searching (and maybe calculating) and outputting values. [* means that this option may not be possible if N is too small. An error message should be the response to the user in that case]

9. When your program is working correctly and robustly (no matter what the inputs), call the TA or Tutor to show your program running, displaying correct results. Explain how it sorts and how it searches (and calculates)

10. Now save the program of part 9, adding it to the name_surname_MIPS.txt file that you created earlier. The file now contains 3 MIPS assembly programs you wrote. Save this MIPS.txt file, you will need it later.

Part 4. 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_MIPS.txt created in Parts 1, 2 and 3. Be sure that the file contains exactly and only the codes which are specifically detailed above. 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 plaigarism). 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 5. Cleanup

1) Erase all the files that you created (MIPS, Verilog, .txt), and that Xilinx created for you. In other words, leave the lab computer exactly as you found it.

2) 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.

CONGRATULATIONS, you are done with Lab1, and one step closer to becoming a Computer Engineer!

LAB POLICIES

1. Each student will earn their own individual lab grade, by how much they do, how much they learn, how much they know. The questions asked by the TA will have an effect on your individual lab score, so be prepared to answer questions concerning the lab topics.
2. Lab score <u>will be reduced to 0</u> if the code is not submitted for similarity testing, or if it is plaigarized. MOSS-testing will be done, to determine similarity rates. Trivial changes to code will not hide plaigarism 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 it MOSS will find it. (The exception to this, for this lab, is the sorting algorithm, as noted above.) The use of the ideas we discussed in the classroom is not a problem.
3. You must be in lab, working on the lab, from the time lab starts until your work is finished and you leave. (Bathroom and snack breaks are the exceptions to this rule).
4. No cell phone usage during lab. Tell friends not to call during the lab hours--you are busy learning how computers work !
5. Internet usage is permitted only to lab-related technical sites. No Facebook, Twitter, email, news, video games, etc--you are busy learning how computers work !