CENG 351: Computer Architecture I

Lab 3: Assembly

Labs should be completed in pairs.

For computer hardware, the instruction set defines a specific architecture. It is important to understand how assembly language (code written using individual instructions) can translate higher-level code to run on a specific processor. Understanding this will help us understand the hardware design.

In this lab, you will explore a pre-written assembly program using an online MIPS simulator. You will use the simulator to figure out what the assembly code is doing and then write the program using high-level code (or pseudocode). Include answers to the lab questions in the report.

Simulating Assembly

1. There are many software packages available for simulating MIPS. We will use the UNC miniMIPS Architecture simulator because it is a web-based tool, it is easy to use and learn, and it has more than enough capability to simulate any assembly we will work with in this class. The MIPS simulator can be found here: http://www.csbio.unc.edu/mcmillan/miniMIPS.html
2. The first step is to enter in your assembly. Type (or copy-paste) in the following program into the window. *Note the following*: this program uses the "mul" instruction, which we have not covered in class. You can look up what this instruction does (hint: it is an R-type instruction). The words on the left followed by colons are labels. They are used to reference the memory location of the instruction they are labelling (such as in the beq instruction). The instruction names should all be aligned, along with the instruction operands.

```
main:   addi    $16,$0,5
        addi    $17,$0,1

top:    beq     $16,$0,done
        mul     $17,$17,$16
        addi    $16,$16,-1
        j       top
done:   sw      $17,0x1234($0)
```

3. Before running the program, you first must use the "Assemble" button. The webpage will give you a pop-up notifying you whether or not this process was successful. There should not be any errors in the assembly program, so if you

see an error you may have typed in something wrong.

4. To run the program, you can use the "Step" button to step through individual instructions (recommended) or the "Multistep" button to step through more than one instruction. Additionally, if you begin an assembly line with an asterisk (*), that will set a breakpoint. If you set a breakpoint, you can use the "Run" button to run the program until you hit that breakpoint.

5. As you run your program, you will notice that the assembly instructions are translated to machine language. The memory address of each instruction and the machine language code (in hexadecimal) are shown along with each assembly instruction. Notice what happens when you step from the jump (j) instruction to the next instruction.

Address	Contents	Instruction
0x80000000	0x20100005	main: addi $16,$0,5
0x80000004	0x20110001	addi $17,$0,1

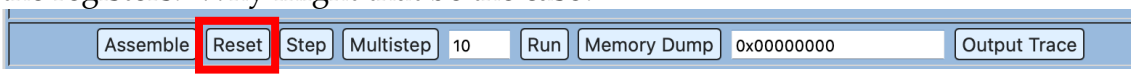
6. The MIPS simulator also keeps track of the values stored in registers. Note that these registers are (mostly) defined by their address number and not their names ($s5, $t0, etc.). Watch how the values change as the program executes. Furthermore, notice that the PC (program counter) is included with the other registers in the top-right corner. What values are stored in the PC?

Registers, Instruction Count = 2, Memory References = 2			[K]$pc: [0x80000008]
$0: [0x00000000]	$1: [0x00000000]	$2: [0x00000000]	$3: [0x00000000]
$4: [0x00000000]	$5: [0x00000000]	$6: [0x00000000]	$7: [0x00000000]
$8: [0x00000000]	$9: [0x00000000]	$10: [0x00000000]	$11: [0x00000000]
$12: [0x00000000]	$13: [0x00000000]	$14: [0x00000000]	$15: [0x00000000]
$16: [0x00000005]	$17: [0x00000001]	$18: [0x00000000]	$19: [0x00000000]
$20: [0x00000000]	$21: [0x00000000]	$22: [0x00000000]	$23: [0x00000000]
$24: [0x00000000]	$25: [0x00000000]	$26: [0x00000000]	$27: [0x00000000]
$gp: [0x00000000]	$sp: [0x00000000]	$fp: [0x00000000]	$ra: [0x00000000]

7. The last instruction in the program stores a value to memory. In order to see the contents of memory, we can use the "Memory Dump" button. Enter in the address of memory you would like to see and click the button to get a popup of memory contents starting at that address. Take a screenshot of a memory dump of the memory location in the final instruction and include it in your lab report.

8. Try to understand what the program is doing. You may need to re-run the program to understand it and answer the questions below. If you need to do

that, you can use the "Reset" button to reload the instructions into memory and start from the beginning. Note that this does not reset the contents of memory or the registers. Why might that be the case?



High-level translation

1. After you understand the operation of the program, write a short code snippet in a high-level language of your choice (or pseudocode) that would result in the assembly language program.
2. Keep in mind that registers are used as the processor's working storage (for things like variables). You can use comments or other notes to indicate the link between variables in your code and registers.
3. The final instruction of the program is a sw instruction. You do not need to include high-level code that would result in this instruction, but may do so if you wish.
4. Make a small change to the high-level code (changing a variable value, an operator, etc.) and show how the corresponding assembly language program would change to match.
5. Make the assembly code change in the simulator and simulate the new program to check that you are correct.

Compile a short report that contains screenshots of the instructions (machine and memory address, below the assembly) at the end of program execution, along with the memory dump of the final value stored at the end of the program. Include the high-level code you wrote to represent the assembly, as well as your changes to the high-level code and assembly. Describe any problems you had during the lab and what you did to troubleshoot. Describe each partner's contribution to the lab. Include answers to the lab questions below

Lab questions:

1. Include answers to questions posed in steps 6 and 8 of the assembly simulation.
2. In plain English, what is this assembly program doing? You can reference your high-level code to help describe the operation if it is helpful.
3. What are the MIPS names of the registers used in this program?

Submission

Please submit a digital copy of your lab report. If taking screenshots, please save them into the lab report document instead of as separate files. The lab should be submitted

through canvas and only one report needs to be submitted per group. In the lab report, please indicate all lab partners and their contributions.

Due Date

This assignment is due on the date shown on Canvas by 11:59pm. Submit via Canvas in PDF format.

Grading

Grading will be based on correctness and completeness of the solution (i.e. show all your work).