Assignment 2 Implementation Guide

I recommend that you accomplish assignment 2 in steps. After you implement each step, compare your output with the output generated by my executable. This process will ensure that you are on the right track and will allow you to get partial credit if you cannot complete the assignment.

1. Test to see if you can read the .text and .data directive from standard input. Do not include any instructions or data. You should be able print out the object code header to standard output that consists of just one line indicating there are 0 instructions and 0 data.
2. Translate an assembly program with one instruction. This means that you would have a .text directive followed by a single syscall instruction. See if you can recognize the instruction mnemonic. If you cannot, then print out an error message to standard error so you can determine if your code did not work or if you incorrectly entered the instruction. Setup a struct representing an R-type instruction that includes the R-type fields: op, rs, rt, rd, shamt, and funct. Assign the appropriate value to the funct field by looking up the value in Appendix A of your textbook. Assign zero to the other fields. Print out the number of instructions you have read (1) and the number of data values you have read (0). Either produce a single unsigned integer value representing the instruction by a series of left shift and or operations or use a union with an unsigned int field and another field that is a struct containing unsigned int bit fields for each of the fields of the R-type instruction. Print the value out to standard output as a hexadecimal value.
3. Extend your solution to test an assembly program with R-type instructions that have one operand: mfhi and mflo. Check to make sure you have successfully recognized each instruction. Assign both the appropriate value for the funct field in Appendix A for these instructions and update the rd field by reading the register name and converting it to an integer value. Use the table in the MIPS Integer Registers slide to determine what the appropriate integer value should be for the register. Assign zero to the other fields. Store each of the instructions. Print out the number of instructions and the number of data words (still 0). Print out each of the instructions as a hexadecimal value.
4. Extend your solution to test an assembly program with R-type instructions that have two operands: div and mult. Assign the values for the funct, rs, and rt fields. Assign zero to the other fields and store each of the instructions. Print out the number of instructions and the number of data words (still 0). Print out each of the instructions as a hexadecimal value.
5. Extend your solution to test an assembly program with R-type instructions that have three operands: addu, and, or, slt, and subu. Check to make sure you have successfully recognized each instruction. Assign the values for the funct, rs, rt, and rd fields. Complete the translation of the assembly program.
6. Extend your solution to test an assembly program with I-type instructions that can use integer offsets: addiu, lw, and sw. Extend your solution to also recognize data directives: .word and .space. Check to make sure you have successfully recognized each instruction. Assign the values for the op, rs, rt, and immed fields. Complete the translation of the assembly program by printing both the instructions and data as hexadecimal values.
7. Extend your solution to test an assembly program with I-type instructions that can use named offsets: beq, bne, lw, and sw. This will require using labels for some instructions and data. Include a label to be stored with each instruction and each data word. Also include a label that could be referenced by each instruction. When recognizing each of these instructions store the

label of associated with the instruction if one exists and the label when it is referenced by the instruction. Assign the values for the op, rs, rt fields. Make another pass for the instructions that referenced a label. Alternatively, you can calculate the label values first, then process the instructions together in the second pass. Appropriately translate labels to offsets referenced by branches (beq and bne) and memory references (lw and sw) and update the immed field. Complete the translation of the assembly program.

8. Extend your solution to test an assembly program with J-type instructions: j, which also references a label. Appropriately translate the referenced label as an absolute address within the instructions. Complete the translation of the assembly program.

9. Test your solution with other MIPS assembly test programs.