Program #1

Due Friday, Oct. 26, 2018 at 4:59pm

Collaboration

You must work on this assignment by yourself. You may not look up solutions for any part of this project from any resources (e.g. previous years' solutions, the Internet, other textbooks, etc.) including code you have written for similar assignments in other courses. You are allowed to look up C++ standard library information, and you are allowed to look online for explanations of compiler errors. You are allowed to discuss this project with the instructor, and peer tutors. You are not allowed to discuss the program with any other students or non-students until after the submission due date and time.

MIPS Decoding

For this assignment, you will decode 32 bit binary encodings of MIPS instructions and print out their MIPS assembly representations.

Your main program should accept the name of a file to parse from as a command line argument. Each line of the file should contain a single 32 bit binary string which is the encoding of a string. (However, you should check to make sure that this format is followed.)

After reading in your file and making sure that there are no errors (meaning, each line is formatted correctly and each line holds a valid encoding of an instruction), you should print out the 32 bit binary encoding followed by the MIPS assembly version of that instruction to stdout. See the files input.mach and input.asm in ~kshaw/share/cs301/prog1. The formatting of your output should exactly match the sample output. If there are errors in the input file, you should print out an error to stderr and exit before printing any other output.

The instructions you must properly decode are:

- ADD
- ADDI
- XOR
- MULT
- MFLO
- SLL
- SLT
- SLTI
- LB
- BEQ
- J

You should expect immediate fields to be encoded in two’s complement and for there to be both positive and negative numbers.

Registers should simply be encoded as their number preceded by a “$”.

Addresses (for BEQ and J) should be encoded in hexadecimal and preceded by a 0x. (You are welcome to use the stream operator to convert to hexadecimal.)

Deliverables and grading

- Your grade will be based 50% on correctness, 15% on design, 15% on testing, 10% on style (including commenting), and 10% on your design document.
- I am requiring you to use the OpcodeTable and Instruction classes from lab 4, although you will likely need to modify them. I recommend that you create different parsing functions for R, I, and J type instructions. I strongly recommend that you consider reusing the format of the ASM.cpp file and ASMParser class from lab #4. (Note: I would be happy to discuss your design with you before you start coding your solution.)
- Your design document (named DESIGN) should explain at a high level your design; it should closely follow the example design write-up provided to you for lab #4. It should describe what each class represents logically and it should describe how the instances of the classes fit together to produce the final application. Another section (not shown in the lab #4 design document) should very specifically explain how your code would need to be modified in order to add a new MIPS instruction. (Note, the easier it is to do so, the likelier you'll achieve more points for good design.)
- You need to submit at least two different test files. At least one of the test files must be formatted correctly and consist only of correct binary encodings which demonstrate that your code works. Specifically, you should make sure to show that your code can handle positive and negative values for all instructions that have immediates encoded in them. (For SLL and J, you do not need to test negative values.) (Note that testing is worth 15% of your grade, so you want to make sure you test exhaustively and intelligently.)
- Your code needs to compile with the –Wall flag without any warnings or errors. Additionally, you must use a Makefile.
- You are expected to have removed all memory leaks from your code.
- Methods and class declarations need to be commented extensively in both header and code files.
- Finally, make sure you follow good object oriented design principles as described in the "Coding Standards" document. For example, you should keep methods relatively short and focused on a single functionality. You should create private helper methods to perform some functionality (e.g. converting from binary strings to integer values, checking that an input string is 32 bits long, checking that an input string contains only "0" and "1" characters). You should define constant variables. You should make sure to hide data. Hiding data includes not returning private data structures from a class, but instead adding methods to the class to allow users of the class to enquire about the data stored in those private data structures.

Submitting Your Work

You must submit your work using a script called turnin **by Friday, Oct. 26 at 4:59pm**.

Remember to comment your code, to put your name in each file, and make sure you deallocate any dynamically allocated memory.

Remeber to turn in a `README` file with your name.

While you're in your solution directory, type the following command:

```
turnin -v -c cs301 -p prog1 *
```