MIPS Disassembler
Project 1 – CS 3339

See Canvas for due date and time. Typically, Friday by 11:59pm, plus a 24-hour late period with 10% deduction. Submissions are not accepted after the late period.

PROBLEM STATEMENT

In this project, you will write a disassembler that reads MIPS machine instructions from a (simplified) binary executable file and prints each assembly language instruction to the screen.

Write your disassembler code in the provided C++ code skeleton, `disassembler.cpp`. All of your code will go in the `disassembleInstr` function. You should read and understand the rest of the code, as it will form the basis for the remaining projects. Your disassembler only needs to support the MIPS instructions listed in comments in the code skeleton.

Your disassembler must precisely match the output in the sample output files, including all whitespace and formatting. Note that all constants are in decimal representation except PC values and jump and branch targets, which are in hexadecimal.

Please use the updated "green card" provided in class and posted on Canvas. Note that the green card says the `sll` instruction means '`rd = rt << shamt`' whereas the official MIPS standard says it means '`rd = rs << shamt`'. We will follow the provided MIPS standard for all shift instructions. Make your code match the comments and output files and disregard the "green card" for these two instructions.

ASSIGNMENT SPECIFICS

This project is to be submitted individually, and you should be able to explain all code that you submit. You are encouraged to discuss, plan, design, and debug with fellow students. The code you add and submit must be your own; you may not copy code from another student or any other source.

All provided files are on Canvas. After moving[1] the compressed tar file to your home space on zeus, the following command will extract the files:

```
$ tar xzvf cs3339_project1.tgz
```

This will create a directory called project1 that contains several sample executables (`*.mips`) for testing your disassembler, the corresponding expected output files (`*.dis`), and the disassembler code skeleton (`disassembler.cpp`) to which all your code must be added.

Once you have added the missing code to the `.cpp` file, compile the file with this command:

```
$ g++ -O3 disassembler.cpp -o disassembler -std=c++11
```

[1] Usually this means download to your machine and then use Filezilla to upload to zeus. Safari may try to uncompress the file upon download and rename it. If you download to a Mac the tar command won't work.

Assuming the compilation succeeded, you can run your disassembler on the provided `*.mips` files. For example, to disassemble `test1.mips`, run this command:

```
$ ./disassembler test1.mips
```

To output the disassembly to a file instead of to the screen and then compare the output to the provided expected output, run these 2 commands:

```
$ ./disassembler test1.mips > test1.out
$ diff test1.out test1.dis
```

If the `diff` command produces any output to the screen, then the two files are ***not*** identical and you need to change your disassembler code so that the outputs match. If there is no apparent difference, try the '-w' option. This ignores whitespace, if it matches that means you have an extra space or tab somewhere. If you want a positive confirmation that the files are identical (as opposed to the standard behavior of no output) then you can us the '-s' option. For this an all linux commands the "man pages" are useful. Type 'man diff' to access.

Additional Requirements:

- **Your code must compile with g++ and run on zeus.cs.txstate.edu**
- Do not change any code outside of the `disassembleInstr` function and name block
- Your code must be well-commented, sufficiently to prove you understand its operation
- Make sure your code doesn't produce unwanted output such as debugging messages
- Make sure your code is correctly indented and uses a consistent coding style
- Do not use TAB characters for indentation! (Use a consistent # of spaces per indent level)
- Clean up your code before submitting; i.e., make sure there are no unused variables, unreachable code, etc.

SUBMISSION INSTRUCTIONS

All project files are to be submitted using Canvas. You can "View the assignment submission details to know that you have turned in the assignment. All file submissions also appear in your personal unfiled folder." Per community.canvaslms.com

Fill in the name block in the top of the cpp file with your first name last name and netID.

Any special instructions or comments to the grader, including notes about features you know do not work, should be submitted using assignment comments. Submit only your final `disassembler.cpp` file to Canvas. Do not submit any additional files (i.e., no provided files or generated disassembly output).

You may submit your file as many times as you like before the deadline. Hint: Compile frequently and once you have a partially working solution upload a "safe" copy in case anything unforeseen happens. Only the last submission will be graded. Late assignments will incur a 10% deduction and after the late deadline passes no assignments will be accepted. ***Canvas will not allow submission after the deadline***, so I strongly recommend that you do not wait until the final seconds of the assignment window.