Project 1 Description

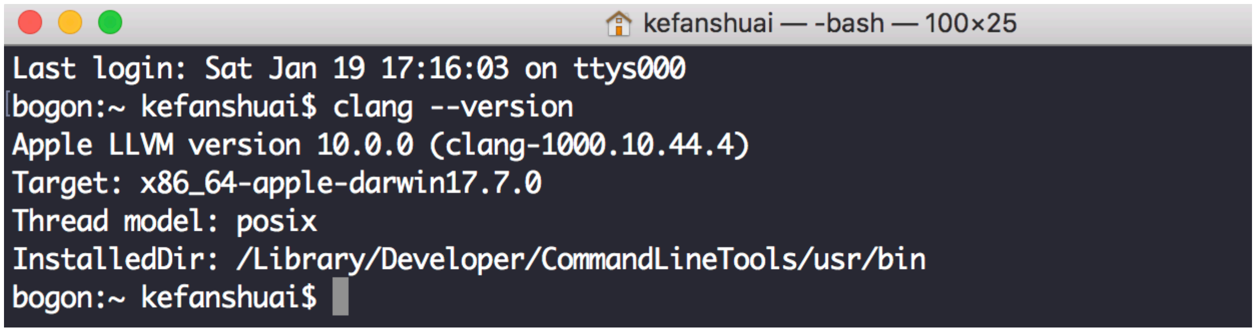
The first project of this course is to write an assembler for MIPS assembly language. The flow of the project is to take an input of MIPS assembly language file, assemble it, then generate a simple output file which is “executable”. The output file should contain machine code for each instruction. The file should be able to run on the MIPS Simulator, which is the second project of the course. You may write your code in either C Language or Python.

The output should be a .txt file which contains the machine code of the corresponding input .asm file. If your write in C Language, the program will be tested on linux system, so make sure your program correctly runs on linux system. I will provide you with two input files and their corresponding output, so that you can debug your program.

Basics of Linux:

For Mac Users:

Check whether you have C Language installed on your mac. Open up your terminal, then type the following command:



```
Last login: Sat Jan 19 17:16:03 on ttys000
bogon:~ kefanshuai$ clang --version
Apple LLVM version 10.0.0 (clang-1000.10.44.4)
Target: x86_64-apple-darwin17.7.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
bogon:~ kefanshuai$
```

If your Mac shows the version of C Language, that means you already have the environment to write this program. Otherwise, you will have to install Xcode on your Mac. Go to App Store on your Mac and search for Xcode.

After doing that, check again to see if you have the environment for C. If you do, you can now begin to write your code.

For Windows Users:

You will need to install a virtual linux system. Virtualbox is recommended, since it’s free. While you can still purchase VMWare if you would like to. Here below is a tutorial link teaching you how to do that.
https://itsfoss.com/install-linux-in-virtualbox/

If you would like to write your program in Visual Studio or other IDE, you may. However, make sure you test your code on linux environment before you turn in your code.

After you have the C environment on your Mac or installed virtual system on your Windows, you have only one more thing to prepare before you start to write your code. The thing is to download the package I have provided. In the package, I have a couple of files.

1. **tester.c** This is where the main function is located. I will use this function to test your program and so can you. I will provide two test cases in the form of .asm file, and the output file as .txt file.
2. **testfile.asm, testfile2.asm, output.txt, output2.txt** These are the test files you will look into. You can always create more test cases your selves just to make sure you are doing it correctly.

The Program's Logic

We already know the things to achieve in this project, the question is, how do we do it?

The basic idea is to break this project into two parts: the first part is to scan through the file, and find out where are the tags, such as "main", or "loop". For each tag you find, you will need to save the data of what is the tag, and what's its memory address. Then you will need proper data structure to store it. Therefore, you will need to write a program to do the phase 1 job. Let's call it phase1.c.

In order to store the information of tags, you will need to write data structures and corresponding functions. Let's call it labelTable.c.

Then, there is the second phase, where you will need to use the information generated in phase 1 (stored in a labelTable) to output the final answer. To provide a little more detail, you will need to look read the instruction, and look into your table if needed. This is named phase2.c.

There are three .c files that needed to be turned in in this project. Also, for each .c file, please write a .h file (header file) corresponding to it. The reason why we need a header file is that we need to declare data structures and function prototypes, so that the compiler can see the DS/function before we declare one. You definitely can write everything in .c file, like define DS/functions before they are declared, however, that is not a good way to write code. The code written in that way will be chubby and noisy. Moreover, if you are writing a huge project, this will make your debug process super long. Please note that the name has to be an exact match or you will not get your grade.

Linux Commands

If you are using a linux system or virtual system, you can ignore this. If you are using Terminal on your Mac or sub-system on windows, there are some useful commands you will need to know.

cd directory : the change directory command, followed by the directory under current directory that you would like to go to. Or use "cd .." to go back.

pwd : print the current directory

ls : list file. Print all the files under the current directory

For more commands, please refer to this page: https://searchdatacenter.techtarget.com/tutorial/77-Linux-commands-and-utilities-youll-actually-use

How to Write Your Code:

You can just use your IDE if wanted to. There are other options such as text editors. I would recommend you a text editor named Sublime. You can find it for both Windows and Mac, and it's simple to use, developer friendly, and beautiful.

There are other options such as vim, which is a Linux editor. Once you are in the terminal, you can simply type "vim file.c" to open a file named "file.c", if it's not found under the directory, it will create one for you. Then hit i for insert, and write code. When you are done, press "esc" to exit insert mode, and use ":q" to quit vim, or ":wq" to save and quit.

Make sure all your files related to the program are under the same directory, including the testing input and output. After you are done editing, you can use the command "gcc tester.c phase1.c phase2.c labelTable.c -o assembler" to include all the needed .c files, and produce an executable file named "assembler". Any errors or warnings will be shown on the terminal. If you successfully compiled the program and produced an executable file (Use "ls" command to see whether you have it under the directory), you can use "./assembler" to run the program and test it.

How to Turn in The Code

When you are done, you will have to submit all the files you have modified. You do not have to turn in tester.c and input/output file. Submit the files on BlackBoard, as there will be an assignment created.

Academical Dishonesty

According to the University's policies, cheating is prohibited. Anyone who was caught cheating will obtain an automatic zero on this project. Please do not copy your fellow students' codes, or share your code with other students. Automatic zero goes to both sides.

Grading

Phase1 of the project - 20%

labelTable - 20%
- construct reasonable data structures - 10%
- work correctly to save labels and find label - 10%

Phase2 of the project - 40%
- find corresponding instruction type - 10%
- map correct machine code to instructions, registers, etc - 25%
- correctly output a .txt file - 5%

Coding style - 10%
- clean and neat coding style and comments - 5%
- correctly write header files with style - 5%

Project report -10%

NEWLY ADDED!!!

The MIPS instructions your code will have to support are stated in the file posted on blackboard. You do not need to support any other instructions and I will not grade your code based on assembly language files with other instructions. The test files will not contain extreme end cases just to test your carefulness. This program is not aimed to test that kind of ability. The only things will be in the test files are instructions, labels, and comments.

The tester.c is provided, if you wanted to use your own tester, or the tester provided does not fit your code, REMEMBER TO SUBMIT YOUR OWN TESTER.C. If your submission does not have a tester.c, I will test it with the SAME tester provided to you. Your phase2.c must write the answer to a file named “output.txt”, which will be compared with the corresponding expected output by the tester.

To test your program, do the compilation like mentioned above (with gcc). Then you should have an “output.txt” file under the same directory along with your other code. Run the command “./assembler testfile.txt output.txt expectedoutput.txt”, then you will see whether you pass the test or not.

IF you are using your own tester, it has to follow the same testing process with the tester provided: I will only test your program using the command stated above with different input file and expected output file. In fact, you can just make changes you need to my “tester.c” to avoid this kind of problems.

IF you are writing your code in python, you will have to provide me your own tester.