## Some information about the project

The project runs in the Ubuntu-16.04.6-desktop-i386 system. Use the instruction "g++ -std=c++11 filename.cpp -o assembler" to assemble the file. And using the "./assembler input_file out_file" to run the program. Notice that the input_file should be a filename with absolute path, the output_file should be the name of the file, which does not contain the path. The output_file would be generated in the same dictrectory.

## Main idea of the project

The project is to design an assembler which takes in input file of MIPS code in the form of .asm file, and translates the MIPS instruction to machine code according to related format, and finally output the result in a text file.

## The implementation of the program

The main idea of the program is to translate the MIPS instructions line by line. However, some instructions have the function to jump to different labels. To translate this kind of instruction, the addresses of the labels are needed. However, it is quite troublesome to translate these instructions since the program needs to scan the whole file again to search for the addresses. To solve this problem, the program will scan the file twice.

During the first scanning of the file, the program would scan the whole file line by line. The program would check whether the first word of each line is label, if so, the program would use map<string, int> to store (label, address) in the map structure, which makes it more convenient for the translation. Also, during the first scanning, the program would store the line numbers of some instructions, which follow the rows of non-instruction, like null line or comments line. These line numbers are stored in a map<int, int> in the form of (counter number, line number). This map can be used to correct the line number of the instructions during the second scan, because the when using the operator “>>” to get next word, it might directly jump through some null lines, which might causing the line number of the instructions inaccurate. By doing so, the program can inform the user the line number of incorrect instructions during the second scanning.

In the second scanning, the program will transfer the MIPS instruction to machine code line by line. To make the translation easier, the program divided the whole instructions into 16 types, according to the input of the instruction. For example, the program groups the instructions “add rd, rs, rt”, “addu rd, rs, rt” into a group, because they all take the same input “rd, rs, rt”. Grouping in this way makes the translation easier, because instructions in each group have nearly the same format of machine code, their formats of machine code only differ in one or two parts, for example, the group members of “xxx rd, rs, rt” only only differ from each other in the “funct” part (except “mul” have the different op “0x1c”). Also, classifying the instructions in this way makes it easier to store the information of the instructions, because it is only needed to store the information of the part where they are different from each other. In each group, the program stores the different part of the instructions using the map<string, int> to store (instruction name, difference). During the scanning, the programs would scan each line, and check whether the first word is label, instruction or comments. If it is label, the program would skip the label. If it is a comment, the program would skip the whole line. If it is an instruction, the program would check which type the instruction belongs and do the translation of the rest of the line. If any wrong instruction occurs, the instruction would print out the line number of wrong instruction and continue to translate the next part of the file. To translate the instructions, the program use four char[] to store the information of the line. To be specific, the three or four char[] would hold the instruction, first to second or third arguments of the instruction. After convert the different arguments to number, the program would translate the number to binary and write them into the output file.