Some information about the project

The project runs in the MacOS system. Use the instruction “g++ -std=c++11 MIPSsimulator.cpp MIPSassembler.cpp dataHandler.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 folder with input file.

Main idea of the project

The project is to design a simulator. The simulator cooperated with assembler in the project 1 will take an input of MIPS codes and convert it to a program, in the way of computer. Also, the program would generate a text file of machine code for the MIPS code.

The implementation of the program

The main idea of the program is to simulate the machine code like the real computer, which is the basic machine cycle.

First, it maintains a memory block to simulate the real memory. Also, the program allocates some memory for the 32 registers, in the form of int[32].
It also assigns memory for hi and lo registers and LLbit, which are all in the form of integer.

The simulator will first read the .data segment and store the data into the data

segment in the memory block. For all the data, the number of bytes of the memory allocated for them are the multiple of 4. Notice that the input MIPS code must contains the .data and .text segments and the .data should be placed before the .text.

After that the simulator would place the machine code generated by the assembler into the text segment successively in the memory. The program would transfer the data which originally in the form of string to the form of integer and store these integers.

After the machine code and data stored in the memory, the program would execute the machine code in the text segment in the memory. The program also maintains a int * pointer to point the text segment which simulates the program counter, and it would reads the machine codes line by line. To makes the programs fetches efficiently, the program divided the instructions into I, R, J, syscall types and for I type, it separated a special subtype whose opcode is 1. The program will first check the opcode of the machine code. Then it will jump to the type of instruction according to the type, that is, 00000 for R type, 000010 and 000011, 0xc for syscall and the rest of the instructions is I type. After that, the program would check the details of the machine code and find out the exact type of the instructions and execute the related function. And the program makes small functions for all the instructions.

For the file operation instruction, which is in the syscall 13 to 16, the program uses two maps which are map<int, fstream*> and map<string, int> to hold the fstream of the opening file. When open a file with fstream, the program would use an fstream pointer pointing to it and assign an integer as file descriptor. And the program would

insert the pair of ((int) file descriptor, (fstream *) pointer) to the first map. Also, the program would store the file name and the related file descriptor to a map, that is, insert ((string) file name, (int) file descriptor) to the second map.