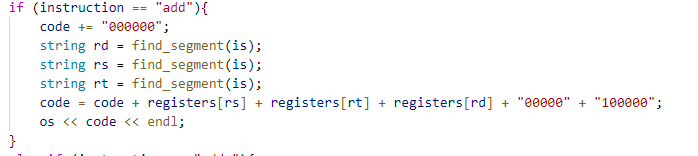
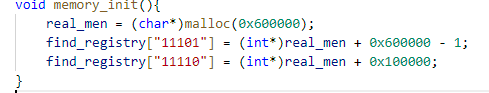
Report – Project 1

[Big picture thoughts and ideas]

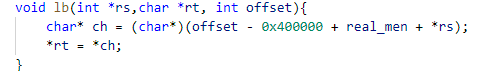
The execution as well as the basic thought of my program contains two major parts: 1, Use assembler to convert MIPS instructions to machine code. 2, Use simulator to simulate the execution of machine code.

First, it’s to know how to convert the MIPS instructions into machine code. The process of converting start with remove the comments and labels in MIPS file. During this process, my program also should divide the MIPS file into .data part and .text part and store them separately. Plus, the program should store the address of the label. Getting a more recognizable code, the next step is to recognize its MIPS instructions. The instruction can be classified into R type, I type, and J type. By recognizing their op code, the program can distinguish them. Then it will recognize what the instruction of it as well as the registers and offset it involves and convert it to binary. After converting, several 32 bits binary codes are stored in another file and the part of assembler is done. 

In simulation part, the program just simulates how the machine code execute in computer. In reality, the information will store in the main memory. The code will be stored in the text segment and static data and dynamic data will be stored in the data segment. Therefore, the first step of simulation is to simulate the memory. The program malloc a size of 0x600000 memory and use int pointer to simulate the registers in CPU. The machine code will be stored into the *text segment* of simulated memory and data will be stored into the *data segment* of simulated memory. When executing the code, they will be loaded for use, like what happens in CPU.



After the memory simulation, the next step is to load the machine code store in memory out and execute them. There will be a pointer pointing to the location of execution and make sure that the machine code is executed one by one. If there is jump all branch operation, make sure that it will point the place which should be executed. Next, the program should get the machine code and recognize what MIPS code it corresponds to, and what registers it need to use. To simulates the MIPS code, this program writes different functions to implement the operation of MIPS code. During the simulation,



(for example, the lb operation is implemented like this)

1. The memory of CPU is simulated by malloc memory in heap.
2. the data stored in the register is stored in the pointer instead.
3. the execution of machine code is implemented by the function written in C++.

Finally, assembler and simulator should be linked together to get a complete program.

[High level implement Ideas]

I break the assembler part into:

1, Remove comments and store the address of label. In this part, I read the characters one by one and put them in one line. Once I meet the “#”, I will ignore the character after that. When there is nothing but ‘ ’ in the whole line, I won’t store this line.

2, Recognize the operation of MIPS code. I get the first segment of the one line of the code, the first segment is used to recognize what operation it is.

3, Base on its type of operation, the program will know what’s the following segments of the code means. They are registers, imm, shamt, or offset. Convert them to their corresponding binary code and combine the binary code in order, the 32 bits binary code is got.

I break the simulator part into:

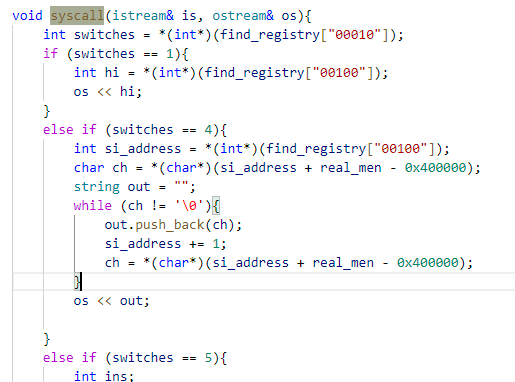
1, Memory simulation and register simulation. I simulate the memory by mallocing a 0x600000 bytes memory. I simulate the register by creating several pointers.

2, Store data and code. I use a pointer to store the 32 bits machine code into the simulated memory line by line. I recognize the type of data, and stored them into the simulated data segment.

3, Function implemented part. In order to simulate the function of MIPS code. I write different functions to implement them. That’s the key part of the simulation. for every function except syscall, the arguments of function are the registers (in the simulator, actually pointers). It does the operation base on the function of MIPS code on the registers or the real memory.

4, Execute the code line by line. I use a pointer “pc” to simulate where the program is executing now. To recognize what the operation of MIPS code it is, and so that to call the corresponding function I write in the third part.

5, Finally, I implement the IO part, that’s the implement of syscall, I read the value store in register $v0. And switches to each IO operations such as write, read\_int.



(part of implementation of syscall)

[Implement details]

1, Map <string, \*int> is used to store the pointers simulating the registers.

2, “malloc” to simulate the memory and “new int” to simulate the registers.

3, “tentotwo” use to convert the decimal number into binary number, it will calculate the twos’ complement code if the decimal number is negative. “twototen\_u” and “twototen\_s” are used to convert the binary number to the usigned decimal number or signed decimal number.

4, several integers used to represent the simulated registers or pointers. For example, pc is initialized to be 0x400000, \*$zero is initialized to be 0;