Design Document

1. Overall

In general, this program is divided into two part, which are assembler and simulator. In the assembler part, the major part is to translate MIPS instructions into machine code. The comments are eliminated. And data are ignored. Every MIPS instruction has its own form, which is R, I or J. Then according to the different registers used, the machine code can be written. And offset, immediate number and address are stored in two's complement into machine code.

In the simulator part, the program has two steps. In the first step, the program allocates memory and simulates registers. The text segment (machine code) is stored from 0x400000 to 0x500000. And the static data (in the .data part) is stored from 0x500000. The size of whole allocated memory is 0x600000, which means the top of stack ($sp points at) is 0x1000000. Notice that the program maps the beginning of allocated memory with 0x400000. Thus, there is a 1-to-1 mapping relationship between the simulated address and real address. In the second step, the program executes the MIPS instructions (machine code). The MIPS instructions are stored in text segment in the form of machine code. With the pointer "pc" moving from 0x400000 (simulated memory), the program read instructions one by one (pc = pc + 4). And every time the program first distinguishes the type of instruction and then call the responding instruction execute function.

2. Program structure

2.1 Assembler

In this part of coding, the program need to translate the MIPS instruction into machine code. In order to achieve this, we can break the problem, into serval sections.

i. Remove the comments

removeComments(istream & is, ostream & os) is used to remove the comments. It distinguish "#" and delete the content after it.

ii. Record the address

recordAddress(istream & is) scan the input file and record the address of all labels

iii. Translate instructions to machine code

toMachineCode(istream & is, ostream & os) is used to translate instructions to machine codes and store them into additional document "code_put.txt". Notice that when translating, toTwosComplement16(int num) is used to turn offsets, immediate numbers and addresses into two's completement form. decimalToBinary(string str, int l) is used to turn decimal numbers into binary numbers in length "l".

2.2 Simulator

In this part of coding, the program needs to simulate the running of MIPS compiler. All in all, it has two parts, which are store and simulate the memory and execute the machine code. In each part, we can divide them into serval sections.

2.2.1 Store and simulate the memory

i. Allocate memory

allocateMemory() is used to allocate memory of size 0x600000 and then point the pointers to the specific positions.

ii. Store machine code and data

storeData(istream & is) can read the data after label ".data" and store them in static data segment (from 0x500000 in virtual memory). Notice that it can read escape characters and store them correctly.

storeMachineCode(istream & is) can read machine codes in additional document "code_put.txt" and store them in text segment (from 0x400000 in virtual memory). Notice that we call readTwosComplement(string binaryCode, int l) to turn machine codes into integers then store in the memory.

2.2.2 Execute the machine codes

i. Implementation of seventy-eight instructions functions

With textbook and reference sources, this program has completed all seventy eight instructions functions.

ii. Distinguish different types of instructions

typeofInstruction(string functCode, string opCode, string rtCode, string decimalCode, istream& is, ostream& os) is to distinguish different machined code by “op” code, “funct” code, etc. Then it calls corresponding instruction functions.

iii. Start simulating

simulate(istream& is, ostream& os) is to realize the execution process. It reads machine code from text segment and call the corresponding instruction function. Every time pc increases by 4, then the program reads next instruction. Notice that every time the program reads an instruction, it will call toTwosComplement(int num) to change it back to binary number.