Project Report

李华悦　　118010138

1 understanding of this project:

The purpose of this project is to write a MIPS simulator which will simulate executing the code if you take a MIPS code input. The MIPS code will first be translated into machine code through an assembler which was written in project1. Then the machine code will be executed through the simulator and the computer will implement corresponding functions. In short, we will execute MIPS codes through C++ code.

2 big picture idea:

The project can be divided into seven parts:

(1) use malloc and pointers got to simulate memory (which is 6MB). Leave 1MB for the text section through the operation of the pointer. Because registers can only store 32bits and the address got from malloc is 64bits, we need to translate the real address to fake address. Then we can store the fake address into registers.

(2) Using another malloc to simulate registers. Because every register can store 32bits, we allocate 4 bytes for every register.

(3) Using an int variable to simulator PC pointer, the variable stores fake address which is the fake address of next instruction.

(4) Using pointers to store the MIPS machine code in the text section of the malloc memory. Every instruction will be translated into an int and then stored in the text section, so it is enough to allocate 4 bytes for each instruction.

(5) Using pointers to store the data into the data section. For word, use int pointer to

store them. For half, use short pointer to store them. For byte, use char pointer to store them. For ascii and asciiz, store them char by char by using char pointer.

(6) Set the fp and sp register pointer to the end of the simulate memory, which is the position of stack, then we can maintain a stack section.

(7) Write functions for all the instructions to simulate the implementation of instructions.

(8) Using if-else if structure to judge the name of each instruction and then call the function corresponding to it.

(9) Write all the functions for the special instruction “syscall”.

3 how to run my code (in Windows):

(1) Run the C++ code.

(2) Enter the input asm file name in the terminal.

(3) Then you can see the execution of the asm file in the terminal.

(4) Exit.

```
please enter the input file name:E:\term5\CSC3050\many_tests.asm
Testing lb, sb, read/print_char, sbrk
Please enter a char:
a
The char you entered is:
a
Testing for .ascii
aaaabbbbccc
bbbbccc
ccc
You should see aaaabbbbccc, bbbbccc, ccc for three strings
Testing for fileIO syscalls
num of chars printed to file:
41
If you see this, your fileIO is all cool
Testing for .half, .byte
For half, the output should be: 65539 in decimal, and you have:
this is or
65539
For byte, the output should be: 16909059 in decimal, and you have:
this is or
this is or
this is or
16909059
Goodbye
```

4 how to simulate memory:

```
//simulate the memory
void *main_memory = malloc(6291456);

//simulate int pointer of the memory
int *int_memory = (int *)main_memory;

//simulate char pointer of the memory
char *char_memory = (char *)main_memory;
```

I use malloc to get 6MB memory from my computer and then use int and char pointer to store text section and data section.

5 how to simulate registers:

```
//simulate all the registers
void *hi_void = malloc(4);
void *lo_void = malloc(4);
void *registers = malloc(128);
```

I use malloc to store registers in memory and allocate 4 bytes for each register.

6 how to simulate PC:

```
//simulate PC
int pc = 0;
```

I use an int variable to simulate PC. The int variable stores fake address.

7 how to simulate stack:

```
//initializes the value in the special registers
*(register1+29) = new_number + 6291456;
*(register1+30) = new_number + 6291456;
*(register1+28) = new_number + 1048576;
```

I put two registers point to the end of the simulator memory, then the stack is created.

8 how to judge the name of instruction:

I use a big if-else if structure to judge the name of every instruction and then execute the function of it.

```
//add instruction
if ((instruction1=="000000")&&(instruction2=="00000")&&(instruction3=="100000")){
    string rsStr = instruction.substr(6,5);
    string rtStr = instruction.substr(11,5);
    string rdStr = instruction.substr(16,5);
    int rs = binReg_to_decInt(rsStr);
    int rt = binReg_to_decInt(rtStr);
    int rd = binReg_to_decInt(rdStr);
    int *register1 = (int *)registers;
    add_func ((register1+rs), (register1+rt),(register1+rd));
    if ((*(register1+rd)>0)&&(*(register1+rs)<0)&&(*(register1+rt)<0)){
        printf("exception occurs at %d",pc);
        exit(1);
    }
    else if((*(register1+rd)<0)&&(*(register1+rs)>0)&&(*(register1+rt)>0)){
        printf("exception occurs at %d",pc);
        exit(1);
    }
    pc = pc + 4;
}
```

9 how to execute the function of every instruction:

I write a function for every instruction. When the if-else if structure recognize the name of the instruction, it will call the function corresponding to it.

10 something to pay attention to:

In fact, for the io open part, if you want to open a file using my code, you must create the file with the right name in that folder in advance in Linux. Because I write and test my code in Windows, so there may be some problems in Linux if you test it in Linux.

However, I make sure that the result of my code is right in Windows. In Windows, if you want to open file using the string of Linux style, you need to create the folder and file in the same path of the C++ exe in advance, then you can execute the io operation.

```
2845                print_string = print_string + 1;
2846            }
2847        //  str = str.substr(1);
2848            char *p = (char*)str.c_str();
2849            //r
```

11 Summary:

I spent a whole week to write this project. It is very challenging. There is something wrong with my Linux virtual machine, so I can only try my best to make it execute in Windows rightly. I have learnt a lot from this project.