易弘鹏 117020343

Project Report

1. Objective

The objective of this project is to write a MIPS simulator which takes in a file of machine code and then run it on the console.

2. Logic

This program consists of three files: “main.c”, “functions.h” and “functions.c”:
“main.c” file is to do some initialization and read the file;
“functions.c” file is to run the code read in “main.c” file. It contains dozens of function needed to run the code.

The main logic of this project can be summarized as:

1) Assign memory for 32 registers and HI, LO registers as global variables so that it can be reference at any function.

```
// Container to store code and '\n'
    currentLine = (char*)malloc(33 * sizeof(char));
// Container of registers, so that register[x] can represent the register with register number x.
    registers = (char**)malloc(32 * sizeof(char*));
// Container of HI, LO register,
    registerHI = (char*)malloc(32 * sizeof(char));
    registerLO = (char*)malloc(32 * sizeof(char));
```

2) Read the machine code file line by line.

```
// Continue reading line by line and run the instrucions for every line.
    while (!feof(fp)){
        fgets(currentLine, 34, fp); // Read to the end of the line(include '\n')
        run();
    }
```

For every line, extract the code as a string and store it as a global variable and then run the code (all functions used for running code are defined in file “functions.c”):

a. Read opcode, identify which type of instruction it is: R type, I type or J type.

b. For each type, identify the value of each required component of instructions – rs, rt, rd, sa, function for R type; rs, rt, immediate for I type; target for J type.

```
// allocate memory to store the value of rs, rt, rd, sa, function
char* rs = (char*)malloc(5 * sizeof(char));
char* rt = (char*)malloc(5 * sizeof(char));
```

```
// copy the code of rs, rt, rd, sa and function from the currentLine
strncpy(rs, currentLine + 6, 5);
strncpy(rt, currentLine + 11, 5);
```

c. Identify the specific instruction, and call the corresponding function by pass in the data we got from step b.

```
// Identify the particular instruction and call the corresponding function to simulate execution
if (!strcmp(function, "100000")) {add(bin2dec5(rd), bin2dec5(rs), bin2dec5(rt));}
else if (!strcmp(function, "100001")) {addu(bin2dec5(rd), bin2dec5(rs), bin2dec5(rt));}
```

d. The specific function does the execution.
(There should be 43 functions related to 43 instructions, but due to time limit only 23 are supported)

```
// Functions that havn't been supported yet.
void andR(int rd, int rs, int rt);
void divR(int rs, int rt);
void divu(int rs, int rt);
void mfhi(int rd);
void mflo(int rd);
void mthi(int rs);
void mtlo(int rs);
void mult(int rs, int rt);
void multu(int rs, int rt);
void nor(int rd, int rs, int rt);
void orR(int rd, int rs, int rt);
void subu(int rd, int rs, int rt);
void xorR(int rd, int rs, int rt);
```

```
// Functions that havn't been supported yet.
void andi(int rt, int rs, int immediate);
void lb(int rt, int immediate, int rs);
void lbu(int rt, int immediate, int rs);
void lui(int rt, int immediate);
void ori(int rt, int rs, int immediate);
void sb(int rt, int immediate, int rs);
void xori(int rt, int rs, int immediate);
```

e. Finish the running of the current line, read the next line.

3. How to run the program

1) Make all the files related to the program are under the same directory, including the input file of machine code.
2) Use the command “gcc main.c functions.c -o simulator” to include all the needed .c files, and produce an executable file named “simulator”.
3) If the program was compiled successfully, use “./assembler inputfile.txt” to run the program and test it, where “inputfile.txt” is the machine code file which will be run.

As shown below, the program will run successfully by following the steps provided above. By the way, the input files here are the test files given on BlackBoard：

```
yihongpengdeMacBook-Pro:project2 yihongpeng$ gcc main.c functions.c -o simulator
yihongpengdeMacBook-Pro:project2 yihongpeng$ ./simulator adder.txt
123
321
444
yihongpengdeMacBook-Pro:project2 yihongpeng$ ./simulator fibonacci.txt
10
55
yihongpengdeMacBook-Pro:project2 yihongpeng$ ./simulator string.txt
hello world
hel
yihongpengdeMacBook-Pro:project2 yihongpeng$ ./simulator char.txt
t
u
yihongpengdeMacBook-Pro:project2 yihongpeng$
```