ICS LabA 实验报告

张艺耀 PB2011630

本次实验让我感受到了面向对象编程魅力，了解了命令行操作、STL（如map，pair，string等）的基本操作及面对小型工程文件该如何处理、编译，了解到流式传输文件和字符串的便利。

实验代码分析与阅读

assemble.h

- 头文件的包含，常量、枚举类型、类、函数等的定义。

```
#include <algorithm>
#include <cstring>
#include <fstream>
#include <iostream>
#include <map>
#include <ostream>
#include <sstream>
#include <string>
#include <vector>
...
```

assemble.cpp

- **输入输出流**

  数据输入和输出过程也是数据传输的过程。

```
#include<iostream>
#include<sstream>
```

- **流式传输的好处**

  可以将部分字串忽略。

```
std::string word;
line_stringstream >> word;
if (IsLC3Command(word) == -1 && IsLC3TrapRoutine(word) == -1) {
                // Eliminate the label
  line_stringstream >> word;
}
```

  可以将字符串中以空格分割的子串单独拿出来。

```
std::vector<std::string> parameter_list;
auto parameter_stream = std::stringstream(parameter_str);
while (parameter_stream >> word) {
  parameter_list.push_back(word);
}
auto parameter_list_size = parameter_list.size();
```

- **string转int**

```
int RecognizeNumberValue(std::string s) {
    // Convert string s into a number
    // TO BE DONE
    int num;
    if(s[0] == '#'){
        s.erase(0, 1);
        const char* c = s.data();
        sscanf(c, "%d", &num);
    }else{
        const char* conv_st;
        std::string tmp = "0";
        s = tmp + s;
        conv_st = s.data();
        sscanf(conv_st, "%x", &num);
    }
```

```
    return num;
}
```

本函数将十六进制以**x或X**开头的string转int类型或将十进制以**#**开头的string转int类型，返回十进制数。

使用sscanf以得到便利。

- **int转string**（二进制）

```
std::string NumberToAssemble(const int &number) {
    // Convert the number into a 16 bit binary string
    // TO BE DONE
    std::string s;
    int num_tmp = number, i = 0, flag = 0;
    if(number < 0){
        flag = 1;
        num_tmp = number * (-1);
    }
    while(num_tmp){
        i = num_tmp%2;
        char c = i + 48;
        num_tmp /= 2;
        s.insert(0, 1, c);
    }
    num_tmp = s.length();
    if(num_tmp != 16){
        if(num_tmp > 16)
            s = s.substr(num_tmp - 16, num_tmp - 1);
        else if(!flag){
            for(int i = 16; i > num_tmp; i-- ){
                s.insert(0, 1, '0');
            }
        }
        else{
            for(int i = 0; i < num_tmp ; i++){
                if(s[i] == '0')s[i] = '1';
                else s[i] = '0';
```

```
            }
            for(int i = num_tmp - 1; i >= 0; i--){
                if(s[i] == '0'){s[i] = '1'; break;}
                else {
                    s[i] = '0';
                }
            }
            for(int i = 16; i > num_tmp; i-- ){
                s.insert(0, 1, '1');
            }
        }
    }
    return s;
}
```

本函数将一个十进制数转换为补码表示的16位二进制string。

主要使用string类的length和insert方法。

- **string（二进制）转string（十六进制）**

```
std::string ConvertBin2Hex(std::string bin)  {
    // Convert the 16-bits binary string into a hex string
    // TO BE DONE
    std::string s;
    for(int i = 3; i >= 0; i--){
        int num_tmp = 0;
        for(int j = 3; j >= 0; j--){
            int W[4] = {8, 4, 2, 1};
            if(bin[4*i + j] == '1') num_tmp += W[j];
        }
        char c = (num_tmp <= 9)? (num_tmp + 48): (num_tmp - 10 + 'A');
        s.insert(0, 1, c);
    }
    return s;
}
```

- 操作数的翻译

```
std::string assembler::TranslateOprand(int current_address, std::string str, int opcode_length) {
    // Translate the opreand
    str = Trim(str);
    auto item = label_map.GetValue(str);
    if (!(item.getType() == vAddress && item.getVal() == -1)) {
        // str is a label
        // TO BE DONE
        std::string s_tmp = NumberToAssemble(item.getVal() - current_address - 1);
        return s_tmp.substr(s_tmp.length() - opcode_length, s_tmp.length() - 1);
    }
    if (str[0] == 'R') {
        // str is a register
        // TO BE DONE
        switch(str[1]){
            case '0' : return "000";
            case '1' : return "001";
            case '2' : return "010";
            case '3' : return "011";
            case '4' : return "100";
            case '5' : return "101";
            case '6' : return "110";
            case '7' : return "111";
            default : return "000";
        }
    } else {
        // str is an immediate number
        // TO BE DONE
        str.erase(0, 1);
        str = NumberToAssemble(str);
        return (str.substr(str.length() - opcode_length, str.length() - 1));
```

```
    }
}
```

主要使用string类的substr和erase方法。

- **assemble函数**

  Step 0

  首先对每行做Trim操作 删去前导后导空格，将所有字母转大写 删去注释。然后打标签、记录Label和地址，最后转换。

```
if (input_file.is_open()) {
    // Scan #0:
    // Read file
    // Store comments
    while (std::getline(input_file, line)) {
        // Remove the leading and trailing whitespace
        line = Trim(line);
        if (line.size() == 0) {
            // Empty line
            continue;
        }
        std::string origin_line = line;
        // Convert `line` into upper case
        // TO BE DONE
        for(int i = 0; i < line.size(); i++){
            if('a' <= line[i] && 'z' >= line[i]){
                line[i] = line[i] - 32;
            }
        }

        // Store comments
        auto comment_position = line.find(";");
        if (comment_position == std::string::npos) {
            // No comments here
            file_content.push_back(line);
            origin_file.push_back(origin_line);
```

```
            file_tag.push_back(lPending);
            file_comment.push_back("");
            file_address.push_back(-1);
            continue;
        } else {
            // Split content and comment
            // TO BE DONE
            std::string comment_str = line.substr(comment_position);
            std::string content_str = line.substr(0,comment_position);
            // Delete the leading whitespace and the trailing whitespace
            comment_str = Trim(comment_str);
            content_str = Trim(content_str);
            // Store content and comment separately
            file_content.push_back(content_str);
            origin_file.push_back(origin_line);
            file_comment.push_back(comment_str);
            if (content_str.size() == 0) {
                // The whole line is a comment
                file_tag.push_back(lComment);
            } else {
                file_tag.push_back(lPending);
            }
            file_address.push_back(-1);
        }
    }
} else {
    std::cout << "Unable to open file" << std::endl;
    // @ Input file read error
    return -1;
}
```

Step 1

查找pseudo操作码并处理；

查找label并记录。

此部分对地址的处理较为关键，由于代码行数较多，故不做代码展示。

Step 2

是整个程序最重要的部分：汇编码转机器码。

每个命令的转换操作大体如一，故不再赘述。

main.cpp

- 命令行参数提取

```
std::pair<bool, string> getCmdOption(char **begin, char **end,
const std::string &option) {
    char **itr = std::find(begin, end, option);
    if (itr != end && ++itr != end) { //++itr表示make_pair的对象是
下一个字符串
        return std::make_pair(true, *itr);
    }
    return std::make_pair(false, "");
}
```

其中 `std::find` 查找给定数字范围内的元素。返回指向范围 [first,last) 中第一个元素的迭代器。

- 文件名和参数输入

```
auto input_info = getCmdOption(argv, argv + argc, "-f");
string input_filename;
auto output_info = getCmdOption(argv, argv + argc, "-o");
string output_filename;

// Check the input file name
if (input_info.first) {
  input_filename = input_info.second;
```

```
} else {
  input_filename = "input.txt";
}

if (output_info.first) {
  output_filename = output_info.second;
} else {
  output_filename = "";
}

if (cmdOptionExists(argv, argv + argc, "-d")) {
  // * Debug Mode :
  // * With debug mode, we can show comments and actual address
  SetDebugMode(true);
}
if (cmdOptionExists(argv, argv + argc, "-e")) {
  // * Error Log Mode :
  // * With error log mode, we can show error type
  SetErrorLogMode(true);
}
if (cmdOptionExists(argv, argv + argc, "-s")) {
  // * Hex Mode:
  // * With hex mode, the result file is shown in hex
  SetHexMode(true);
}
```

- 运行

```
auto ass = assembler();
auto status = ass.assemble(input_filename, output_filename);

if (gIsErrorLogMode) {
  cout << std::dec << status << endl;
}
return 0;
```

程序运行

Mac系统下默认使用的不是c++11标准，故需要将makefile文件更改如下：

```
CC=g++ -std=c++11
CFLAGS=-I. -g
DEPS = assembler.h
OBJ = assembler.o main.o

%.o: %.cpp $(DEPS)
  $(CC) -c -o $@ $< $(CFLAGS)

assembler: $(OBJ)
  $(CC) -o $@ $^ $(CFLAGS)

all: assembler

.PHONY: clean

clean:
  rm -rf assembler
  rm *.o
```

Debug

在删除逗号时遇到Bug。最初使用string类寻找并替换会造成在遇到如 `R1，R2,#3` （“R2”和“#3”之间没有空格）的情况无法解析，原因未知，故引入cstdlib头文件，代码更改如下：（其中注释为更改前内容）：

```
// Convert comma into space for splitting
// TO BE DONE
/*
while(word.find(',') != -1){
  word.replace(word.find(','), 1, " ");
}
*/
std::replace(parameter_str.begin(), parameter_str.end(), ',', ' ');
```