ICS LABA 实验报告

1. 内容简介

本实验要求我们根据现有框架搭建LC-3汇编器，将输入的标准汇编代码转化成对应的机器码。

2. 内容概述

2.2.1 程序实现

根据我们在 ICS 课内所学知识，汇编器需要对汇编代码进行两次扫描。第一遍扫描为了建立符号表，建立整个程序的相互跳转关系；第二遍扫描则根据已有的符号表进行翻译。本实验采用了三遍扫描的模式：

第一遍扫描： 清除注释，将汇编码进行初步处理后以行为单位存入临时容器中；

第二遍扫描： 建立符号表，实现汇编行地址与程序行地址之间的一一对应；

第三遍扫描： 进行翻译。

下面我将按照上述三个步骤进行代码的介绍与分析。

2.2.2 第一遍扫描

程序如下：

```
// 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.
    std::transform(line.begin(), line.end(), line.begin(), ::toupper);    //转换
为大写

    // 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, content_str;
            content_str = line.substr(0, comment_position);     //根据;位置分割
            comment_str = line.substr(comment_position + 1);    //后面的都是注释

            // 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);
        }
    }
```

在打开汇编文件后，程序按行读入汇编指令，检查其中是否有分号。分号前的文本为程序正文，分号后的为注释。该部分程序会将每一行的正文部分依次存入临时空间,为后续的操作做准备。

2.2.3 第二遍扫描

```
// Scan #1:
// Scan for the .ORIG & .END pseudo code
// Scan for jump label, value label, line comments
int line_address = -1;
for (int line_index = 0; line_index < file_content.size(); ++line_index) {
    //开始处理读入的每一行
    if (file_tag[line_index] == lComment) {
        // This line is comment
        continue;
    }

    auto line = file_content[line_index];

    // * Pseudo Command
    if (line[0] == '.') {
        file_tag[line_index] = lPseudo;
        // This line is a pseudo instruction
        // Only .ORIG & .END are line-pseudo-command
        auto line_stringstream = std::istringstream(line);
        std::string pseudo_command;
        line_stringstream >> pseudo_command;

        if (pseudo_command == ".ORIG") {
            // .ORIG
            std::string orig_value;
            line_stringstream >> orig_value;
            orig_address = RecognizeNumberValue(orig_value);
            if (orig_address == std::numeric_limits<int>::max()) {
                // @ Error address
                return -2;
            }
            file_address[line_index] = -1;
            line_address = orig_address;

        } else if (pseudo_command == ".END") {
            // .END
            file_address[line_index] = -1;
            // If set line_address as -1, we can also check if there are
programs after .END
            // line_address = -1;
        } else if (pseudo_command == ".STRINGZ") {
            file_address[line_index] = line_address;
            std::string tword, tline;
            line_stringstream >> tline;

            while (line_stringstream) {
                line_stringstream >> tword;
                if (tword == "" || !line_stringstream) break;
                tline += ' ';
                tline += tword;
            }

            /*f (word[0] != '\"' || word[word.size() - 1] != '\"') {
                    // @ Error String format error
                    return -6;
                }*/
            auto num_temp = tline.size() - 1;
            line_address += num_temp;
        } else if (pseudo_command == ".FILL") {
            // TO BE DONE.
            file_address[line_index] = line_address;
            line_address += 1;


        } else if (pseudo_command == ".BLKW") {
            // TO BE DONE.
            file_address[line_index] = line_address;
            std::string length_s;
            line_stringstream >> length_s;
            auto length_int = RecognizeNumberValue(length_s);
            line_address += length_int;

        } else {
            // @ Error Unknown Pseudo command
            return -100;
        }

        continue;
    }

    if (line_address == -1) {
        // @ Error Program begins before .ORIG
        return -3;
    }

    file_address[line_index] = line_address;
    line_address++;

    // Split the first word in the line
    auto line_stringstream = std::stringstream(line);
    std::string word;
    line_stringstream >> word;
    if (IsLC3Command(word) != -1 || IsLC3TrapRoutine(word) != -1) {
        // * This is an operation line
        // TO BE DONE.
        file_tag[line_index] = lOperation;
        continue;
    }

    // * Label
    // Store the name of the label
    auto label_name = word;
    // Split the second word in the line
    line_stringstream >> word;
    if (IsLC3Command(word) != -1 || IsLC3TrapRoutine(word) != -1 || word ==
label_name) {
        // a label used for jump/branch
        // TO BE DONE.
        file_tag[line_index] = lOperation;
        label_map.AddLabel(label_name, value_tp(vAddress, line_address - 1));

    } else {
        file_tag[line_index] = lPseudo;
        if (word == ".FILL") {
            line_stringstream >> word;
            auto num_temp = RecognizeNumberValue(word);
            if (num_temp == std::numeric_limits<int>::max()) {
                // @ Error Invalid Number input @ FILL
                return -4;
            }
            if (num_temp > 65535 || num_temp < -65536) {
                // @ Error Too large or too small value  @ FILL
                return -5;
            }
            label_map.AddLabel(label_name, value_tp(vAddress, line_address -
1));

        }
        if (word == ".BLKW") {
            // modify label map
            // modify line address

            // TO BE DONE.
            file_address[line_index] = line_address;
            std::string length_s;
            line_stringstream >> length_s;
            auto length_int = RecognizeNumberValue(length_s);
            label_map.AddLabel(label_name, value_tp(vAddress, line_address -
1));
            line_address += length_int - 1;
        }
        if (word == ".STRINGZ") {
            // modify label map
            // modify line address
            // TO BE DONE.
            file_address[line_index] = line_address;
            std::string tword, tline;
            line_stringstream >> tline;

            while (line_stringstream) {
                line_stringstream >> tword;
                if (tword == "" || !line_stringstream) break;
                tline += ' ';
                tline += tword;
            }

            auto num_temp = tline.size() - 1;
            label_map.AddLabel(label_name, value_tp(vAddress, line_address));
            line_address += num_temp - 2;

        }
    }
}
```

该部分逻辑是，首先判断该行是否为伪指令，并根据需要进行空间预分配，计算该行在机器码中对应的行地址。如果该行包含有标签，则会将标签与行地址一起存入map容器中。以此为基础构建了程序的符号表。

2.2.4 第三遍扫描

该部分为翻译模块。对于伪指令部分，相关的操作如下：

```
if (file_tag[line_index] == lPseudo) {
    // Translate pseudo command
    std::string word;
    line_stringstream >> word;
    if (word[0] != '.') {
        // Fetch the second word
        // Eliminate the label
        line_stringstream >> word;
    }

    if (word == ".FILL") {
        std::string number_str;
        line_stringstream >> number_str;
        auto output_line = NumberToAssemble(number_str);
        if (gIsHexMode)
            output_line = ConvertBin2Hex(output_line);
        output_file << output_line << std::endl;
    } else if (word == ".BLKW") {
        // Fill 0 here
        // TO BE DONE.
        std::string number_str = "0";
        auto output_line = NumberToAssemble(number_str);
        if (gIsHexMode)
            output_line = ConvertBin2Hex(output_line);
        output_file << output_line << std::endl;
    } else if (word == ".STRINGZ") {
        // Fill string here
        // TO BE DONE.
        std::string tword, tline;
        line_stringstream >> tline;

        while (line_stringstream) {
            line_stringstream >> tword;
            if (tword == "" || !line_stringstream) break;
            tline += ' ';
            tline += tword;
        }
        for (auto text_index = 0; text_index < tline.size(); text_index++) {
            if (tline[text_index] == '\"')
                continue;
            auto output_line = NumberToAssemble(int(tline[text_index]));
            if (gIsHexMode)
                output_line = ConvertBin2Hex(output_line);
            output_file << output_line << std::endl;
        }

    }

    continue;
}
```

对于指令部分，我们以ADD函数为例，对其的操作如下：

```
//result_line为对应的机器码输出序列
result_line += "0001";
if (parameter_list_size != 3) {
    // @ Error parameter numbers
    return -30;
}
result_line += TranslateOprand(current_address, parameter_list[0]);
result_line += TranslateOprand(current_address, parameter_list[1]);
if (parameter_list[2][0] == 'R') {
    // The third parameter is a register
    result_line += "000";
    result_line += TranslateOprand(current_address, parameter_list[2]);
} else {
    // The third parameter is an immediate number
    result_line += "1";
    // std::cout << "hi " << parameter_list[2] << std::endl;
    result_line += TranslateOprand(current_address, parameter_list[2], 5);
}
```

2.2.5 相关辅助函数

I. 汇编码数字识别函数

该函数的功能是将输入的汇编数字转化成对应的short型整数。

```
int RecognizeNumberValue(std::string s) {
    // Convert string s into a number
    // TO BE DONE.
    //常用的格式有16进制 x12, 10进制 #23等。下面开始转换
    std::string temp;
    bool ifminus = false;
    int answer = 0;
    if (s[0] == 'X') {
        // base 16
        if (s[1] == '-') {
            temp.append(s, 2, s.size() - 2);
            ifminus = true;
        }
        else
            temp.append(s, 1, s.size() - 1);

        std::stringstream ss;
        ss << std::hex << temp;
        ss >> answer;

    }
    else if (s[0] == '#') {
        // base 10
        if (s[1] == '-') {
            temp.append(s, 2, s.size() - 2);
            ifminus = true;
        }
        else
            temp.append(s, 1, s.size() - 1);

        answer = std::stoi(temp);
    }
    else {
        std::cout << "数值错误!" << std::endl;
        system("pause");
        return -1;
    }

    if (ifminus)
        answer = 65536 - answer;
    return answer;

}
```

基本思路是，先判断最高位是 0 还是 1，再将其剩下的位上补 0 或 1 。

II. 数码转换函数

该函数的功能是将十进制数/字符串转化成16位2进制字符串。

```
std::string NumberToAssemble(const int &number) {
    // Convert the number into a 16 bit binary string
    // TO BE DONE.
    int tnumber = number;
    if (tnumber < 0)
        tnumber = 65536 + tnumber;
    std::bitset<16> temp(tnumber);
    std::string answer = temp.to_string();
    return answer;

}

std::string NumberToAssemble(const std::string &number) {
    // Convert the number into a 16 bit binary string
    // You might use `RecognizeNumberValue` in this function
    // TO BE DONE.
    int number_int = RecognizeNumberValue(number);
    std::bitset<16> temp(number_int);
    std::string answer = temp.to_string();
    return answer;
}
```

III. 无意义字符清除函数

该函数的功能是按照需求定向清除字符串首尾的无意义字符。

```
// trim from left
inline std::string &LeftTrim(std::string &s, const char *t = " \t\n\r\f\v") {
    // TO BE DONE.
    bool flag = false;
    int i = 0;
    while (!flag && i < s.size()) {
        if (s[i] == t[0] || s[i] == t[1] || s[i] == t[2] || s[i] == t[3] || s[i]
== t[4] || s[i] == t[5]) {
            s = s.erase(i, 1);
        }
        else {
            flag = true;
        }
    }
    //s.erase(s.begin(), std::find_if(s.begin(), s.end(),
std::not1(std::ptr_fun<int, int>(std::isspace))));
    return s;
}

// trim from right
inline std::string &RightTrim(std::string &s, const char *t = " \t\n\r\f\v") {
    // TO BE DONE.
    bool flag = false;
    int i = s.size() - 1;
    while (!flag && i >= 0) {
        if (s[i] == t[0] || s[i] == t[1] || s[i] == t[2] || s[i] == t[3] || s[i]
== t[4] || s[i] == t[5]) {
            s = s.erase(i, 1);
            i--;
        }
        else {
            flag = true;
        }
    }
    //s.erase(std::find_if(s.rbegin(), s.rend(), std::not1(std::ptr_fun<int,
int>(std::isspace))).base(), s.end());
    return s;
}

// trim from left & right
inline std::string &Trim(std::string &s, const char *t = " \t\n\r\f\v") {
    return LeftTrim(RightTrim(s, t), t);
}
```

3.小结

本实验成功实现了 LC-3 的汇编器，进一步加深了我对于汇编语言的相关理解，也提高了我c++代码的编程能力。