LabA 汇编器

实验内容

- 实现LC3汇编器，输入汇编代码，输出二进制机器码

代码架构

- 汇编器的主体通过assemble函数实现，其中涉及了关键的几个函数，譬如修剪行信息、识别字符串数字等。
- 汇编的过程由三次扫描实现：
    - 第一次扫描时整理行信息、将所有的小写字母转换为大写字母，并对每一行的性质进行标记；
    - 第二次扫描时寻找行标签，并建立标签——地址查询表；
    - 第三次扫描时开始汇编，将需要汇编操作的行进行“翻译”，输出二进制机器码。

核心代码

- 第0次扫描

```
    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);//string
            if (line.size() == 0) {
                // Empty line
                continue;
            }
            std::string origin_line = line;
            // Convert `line` into upper case
            for(auto p = line.begin(); p != line.end(); p++){
                if(*p >= 'a' && *p <= 'z') *p -= 'a' - 'A';
            }

            // 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);
                comment_str = line.substr(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;
    }
```

- 第1次扫描

```
// 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) {
        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 word;
                line_stringstream >> word;
                if (word[0] != '\"' || word[word.size() - 1] != '\"') {
                    // @ Error String format error
                    return -6;
                }
                auto num_temp = word.size() - 1;
                line_address += num_temp;
            } else if (pseudo_command == ".FILL") {
                file_address[line_index] = line_address;
                std::string value;
                line_stringstream >> value;
                int fill_value = RecognizeNumberValue(value);
                if (fill_value== std::numeric_limits<int>::max()){
                    // @ Error value
                    return -2;
                }
                line_address++;
            }
            else if (pseudo_command == ".BLKW") {
                file_address[line_index] = line_address;
                std::string value;
                line_stringstream >> value;
                int length = RecognizeNumberValue(value);
                if(length == std::numeric_limits<int>::max()){
                    // @ Error length
                    return -2; //??????
                }
                line_address += length;
            } 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
            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 || line_stringstream.eof()) {
            // a label used for jump/branch
            label_map.AddLabel(label_name, value_tp(vAddress, line_address - 1));
            if (IsLC3Command(word) != -1 || IsLC3TrapRoutine(word) != -1) file_tag[line_index] = lOperation;
            else line_address--;
            continue;
        } 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
                line_stringstream >> word;
                auto length_temp = RecognizeNumberValue(word);
                if(length_temp == std::numeric_limits<int>::max()){
                    // @ Error Invalid Number input @ BLKW
                    return -4;
                }
                if (length_temp > 65535 || length_temp < -65536) {
                    return -5;///??????
                }
                label_map.AddLabel(label_name, value_tp(vAddress, line_address - 1));

                line_address += length_temp - 1;
            }
            if (word == ".STRINGZ") {
                // modify label map
                // modify line address
                line_stringstream >> word;
                if (word[0] != '\"' || word[word.size() - 1] != '\"') {
                    // @ Error String format error
                    return -6;
                }
                label_map.AddLabel(label_name, value_tp(vAddress, line_address -
1));
                auto num_temp = word.size() - 1;
                line_address += num_temp - 1;
            }
        }
    }

    if (gIsDebugMode) {
        // Some debug information
        std::cout << std::endl;
        std::cout << "Label Map: " << std::endl;
        std::cout << label_map << std::endl;

        for (auto index = 0; index < file_content.size(); ++index) {
            std::cout << std::hex << file_address[index] << " ";
            std::cout << file_content[index] << std::endl;
        }
    }
```

- 第2次扫描

```
// Scan #2:
    // Translate

    // Check output file
    if (output_filename == "") {
        output_filename = "binary.txt";
        if (output_filename.find(".") == std::string::npos) {
            output_filename = output_filename + ".txt";
        } else {
            output_filename = output_filename.substr(0,
output_filename.rfind("."));
            output_filename = output_filename + ".txt";
        }
    }

    std::ofstream output_file;
    // Create the output file
    output_file.open(output_filename);
    if (!output_file) {
        // @ Error at output file
        return -20;
    }

    for (int line_index = 0; line_index < file_content.size(); ++line_index) {
        if (file_address[line_index] == -1 || file_tag[line_index] == lComment)
{
            // * This line is not necessary to be translated
            continue;
        }

        auto line = file_content[line_index];
        auto line_stringstream = std::stringstream(line);

        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);
                if (gIsDebugMode)
                    output_file << std::hex << file_address[line_index] << ": ";
                output_file << output_line << std::endl;
            } else if (word == ".BLKW") {
                // Fill 0 here
                std::string length_str;
                line_stringstream >> length_str;
                int length = RecognizeNumberValue(length_str);
                for(int i = 0; i < length; i++){
                    if (gIsDebugMode)
                        output_file << std::hex << file_address[line_index] <<
": ";

                    if (gIsHexMode)
                        output_file << "0000" << std::endl;
                    else
                        output_file << "0000000000000000" << std::endl;
                }

            } else if (word == ".STRINGZ") {
                // Fill string here
                std::string s;
                line_stringstream >> word;
                s = word.substr(1, word.length() - 2);
                for(auto p = s.begin(); p != s.end(); p++){
                    int n = *p;
                    if (gIsDebugMode)
                        output_file << std::hex << file_address[line_index] + p
- s.begin() << ": ";
                    if (gIsHexMode)
                        output_file << ConvertBin2Hex(NumberToAssemble(n)) <<
std::endl;
                    else
                    output_file << NumberToAssemble(n) << std::endl;
                }
                if (gIsDebugMode)
                    output_file << std::hex << file_address[line_index] +
 s.length() << ": ";
                if (gIsHexMode)
                    output_file << "0000" << std::endl;
                else
                    output_file << "0000000000000000" << std::endl;
            }
            continue;
        }

        if (file_tag[line_index] == lOperation) {
            std::string word;
            line_stringstream >> word;
            if (IsLC3Command(word) == -1 && IsLC3TrapRoutine(word) == -1) {
                // Eliminate the label
                line_stringstream >> word;
            }
            if (gIsDebugMode)
                output_file << std::hex << file_address[line_index] << ": ";


            std::string result_line = "";
            auto command_tag = IsLC3Command(word);
            auto parameter_str = line.substr(line.find(word) + word.size());
            parameter_str = Trim(parameter_str);

            // Convert comma into space for splitting//TOBEDONE
            for(auto p = parameter_str.begin(); p != parameter_str.end(); p++){
                if(*p == ',') *p = ' ';
            }

            auto current_address = file_address[line_index];

            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();
            if (command_tag != -1) {
                // This is a LC3 command
                switch (command_tag) {
                case 0:
                    // "ADD"
                    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);
                    }
                    break;
                case 1:
                    // "AND"
                    result_line += "0101";
                    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);
                    }
                    break;
                case 2:
                    // "BR"
                    result_line += "0000111";
                    if (parameter_list_size != 1) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    result_line += TranslateOprand(current_address,
parameter_list[0], 9);
                    break;
                case 3:
                    // "BRN"
                    result_line += "0000100";
                    if (parameter_list_size != 1) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    result_line += TranslateOprand(current_address,
parameter_list[0], 9);
                    break;
                case 4:
                    // "BRZ"
                    result_line += "0000010";
                    if (parameter_list_size != 1) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    result_line += TranslateOprand(current_address,
parameter_list[0], 9);
                    break;
                case 5:
                    // "BRP"
                    result_line += "0000001";
                    if (parameter_list_size != 1) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    result_line += TranslateOprand(current_address,
parameter_list[0], 9);
                    break;
                case 6:
                    // "BRNZ"
                    result_line += "0000110";
                    if (parameter_list_size != 1) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    result_line += TranslateOprand(current_address,
parameter_list[0], 9);
                    break;
                case 7:
                    // "BRNP"
                   result_line += "0000101";
                    if (parameter_list_size != 1) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    result_line += TranslateOprand(current_address,
parameter_list[0], 9);
                    break;
                case 8:
                    // "BRZP"
                    result_line += "0000011";
                    if (parameter_list_size != 1) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    result_line += TranslateOprand(current_address,
parameter_list[0], 9);
                    break;
                case 9:
                    // "BRNZP"
                    result_line += "0000111";
                    if (parameter_list_size != 1) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    result_line += TranslateOprand(current_address,
parameter_list[0], 9);
                    break;
                case 10:
                    // "JMP"
                    result_line += "1100000";
                    if (parameter_list_size != 1) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    result_line += TranslateOprand(current_address,
parameter_list[0]);
                    result_line += "000000";
                    break;
                case 11:
                    // "JSR"
                    result_line += "01001";
                    if (parameter_list_size != 1) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    result_line += TranslateOprand(current_address,
parameter_list[0], 11);
                    break;
                case 12:
                    // "JSRR"
                    result_line += "0100000";
                    if (parameter_list_size != 1) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    result_line += TranslateOprand(current_address,
parameter_list[0]);
                    result_line += "000000";
                    break;
                case 13:
                    // "LD"
                    result_line += "0010";
                    if (parameter_list_size != 2) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    result_line += TranslateOprand(current_address,
parameter_list[0]);
                    result_line += TranslateOprand(current_address,
parameter_list[1], 9);
                    break;
                case 14:
                    // "LDI"
                    result_line += "1010";
                    if (parameter_list_size != 2) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    result_line += TranslateOprand(current_address,
parameter_list[0]);
                    result_line += TranslateOprand(current_address,
parameter_list[1], 9);
                    break;
                case 15:
                    // "LDR"
                    result_line += "0110";
                    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]);
                    result_line += TranslateOprand(current_address,
parameter_list[2], 6);
```

```
                    break;
                case 16:
                    // "LEA"
                    result_line += "1110";
                    if (parameter_list_size != 2) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    result_line += TranslateOprand(current_address,
parameter_list[0]);
                    result_line += TranslateOprand(current_address,
parameter_list[1], 9);
                    break;
                case 17:
                    // "NOT"
                    result_line += "1001";
                    if (parameter_list_size != 2) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    result_line += TranslateOprand(current_address,
parameter_list[0]);
                    result_line += TranslateOprand(current_address,
parameter_list[1]);
                    result_line += "111111";
                    break;
                case 18:
                    // RET
                    result_line += "1100000111000000";
                    if (parameter_list_size != 0) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    break;
                case 19:
                    // RTI
                    result_line += "1000000000000000";
                    if (parameter_list_size != 0) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    break;
                case 20:
                    // ST
                    result_line += "0011";
                    if (parameter_list_size != 2) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    result_line += TranslateOprand(current_address,
parameter_list[0]);
                    result_line += TranslateOprand(current_address,
parameter_list[1], 9);
                    break;
                case 21:
                    // STI
                    result_line += "1011";
                    if (parameter_list_size != 2) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    result_line += TranslateOprand(current_address,
parameter_list[0]);
                    result_line += TranslateOprand(current_address,
parameter_list[1], 9);
                    break;
                case 22:
                    // STR
                    result_line += "0111";
                    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]);
                    result_line += TranslateOprand(current_address,
parameter_list[2], 6);
                    break;
                case 23:
                    // TRAP
                    result_line += "11110000";
                    if (parameter_list_size != 1) {
                        // @ Error parameter numbers
                        return -30;
                    }
                    result_line += TranslateOprand(current_address,
parameter_list[0], 8);
                default:
                    // Unknown opcode
                    // @ Error
                    break;
                }
            } else {
                // This is a trap routine
                command_tag = IsLC3TrapRoutine(word);
                switch (command_tag) {
                    case 0:
                    // x20
                    result_line += "1111000000100000";
                    break;
                    case 1:
                    // x21
                    result_line += "1111000000100001";
                    break;
                    case 2:
                    // x22
                    result_line += "1111000000100010";
                    break;
                    case 3:
                    // x23
                    result_line += "1111000000100011";
                    break;
                    case 4:
                    // x24
                    result_line += "1111000000100100";
                    break;
                    case 5:
                    // x25
                    result_line += "1111000000100101";
                    break;
                    default:
                    // @ Error Unknown command
                    return -50;
                }
            }

            if (gIsHexMode)
                result_line = ConvertBin2Hex(result_line);
            output_file << result_line << std::endl;
        }
    }
```

- 在第二次扫描中，我们需要对每种不同的指令进行单独翻译，具体的各指令翻译模式比较基本，故不在此展开。

实验总结

通过本次实验，我了解了汇编语言翻译成机器语言的过程，同时也对汇编器有了更加深刻的认识。