# Assembler Lab

姓名：杨涛
学号：PB20020599

# 配置环境

本lab需要使用GNU make进行构建，在linux下直接使用包管理器进行安装。

```
$ sudo apt install make
```

# 代码填充

## LeftTrim&RightTrim

将 `s[0]` 作为 `std::find` 的参数，在 `t` 的字符串中寻找是否存在，若存在则删去 `s[0]` ，直到 `s[0]` 不是 `t` 的字符串中任意一个字符

RightTrim同理

```
// trim from left
inline std::string &LeftTrim(std::string &s, const char *t = " \t\n\r\f\v") {
    std::string sign=t;
    auto begin=sign.begin(),end=sign.end();
    auto temp=std::find(begin,end,s[0]);
    while(temp!=end)
    {
        s.erase(0,1);
        temp=std::find(begin,end,s[0]);
    }
    return s;
}
```

## RecognizeNumberValue

有两种输入方式，一种是前缀带x的十六进制，另一种是前缀带#或无前缀的十进制，根据不同的情况循环读取最低位并乘上基数，同时考虑负号即可

```
int RecognizeNumberValue(std::string s)
{
    // Convert string s into a number
    int sum = 0;
    int weight = 1;
    if (s[0] == 'X')
    {
        s.erase(0, 1);
        int len = s.length(), i = 0;
        if (s[0] == '-')
        {
            weight = -1;
            s.erase(0, 1);
        }
        while (i < len)
        {
            if (s[i] >= '0' && s[i] <= '9')
            {
                sum *= 16;
                sum += s[i] - '0';
            }
            else if (s[i] >= 'A' && s[i] <= 'F')
            {
                sum *= 16;
                sum += s[i] - 'A' + 10;
            }
            else
                return std::numeric_limits<int>::max();
            i++;
        }
    }
    else
    {
        if (s[0] == '#')
            s.erase(0, 1);
        if (s[0] == '-')
        {
            weight = -1;
            s.erase(0, 1);
        }
        int len = s.length(), i = 0;
        while (i < len)
        {
            if (s[i] >= '0' && s[i] <= '9')
            {
                sum *= 10;
                sum += s[i] - '0';
            }
            else
                return std::numeric_limits<int>::max();
            i++;
        }
    }
```

```
    return weight * sum;
}
```

## NumberToAssemble

根据掩码位运算判断某位是否为1之后push_back即可

```
std::string NumberToAssemble(const int &number)
{
    // Convert the number into a 16 bit binary string
    int i = 1 << 15;
    std::string s;
    while (i != 0)
    {
        if (i & number)
            s.push_back('1');
        else
            s.push_back('0');
        i >>= 1;
    }
    return s;
}
std::string NumberToAssemble(const std::string &number)
{
    // Convert the number into a 16 bit binary string
    // You might use `RecognizeNumberValue` in this function
    return NumberToAssemble(RecognizeNumberValue(number));
}
```

## ConvertBin2Hex

按照每四个bits为一组转化为一个十六进制数的规则进行转化

```
std::string ConvertBin2Hex(std::string bin)
{
    // Convert the binary string into a hex string
    static const char *table = "0123456789ABCDEF";
    std::string s;
    int temp;
    for (int i = 0; i < 4; i++)
    {
        temp = 0;
        for (int j = 3; j >= 0; j--)
        {
            temp <<= 1;
            temp += (bin[4 * i + j] - '0');
        }
        s.push_back(table[temp]);
    }
    return s;
}
```

# TranslateOprand

详细补全看注释

```
std::string assembler::TranslateOprand(int current_address, std::string str, int opcode_length)
{
    // Translate the oprand
    str = Trim(str);
    auto item = label_map.GetValue(str);
    if (!(item.getType() == vAddress && item.getVal() == -1))
    {
        // str is a label
        /* 查label表取出地址求offset */
        int offset = item.getVal() - current_address - 1;
        std::string sTemp = NumberToAssemble(offset);
        return sTemp.substr(16 - opcode_length);
    }
    if (str[0] == 'R')
    {
        // str is a register
        /* 求第几个寄存器 */
        int temp = str[1] - '0';
        int mask = 1 << 2;
        std::string s;
        while (mask != 0)
        {
            if (temp & mask)
                s.push_back('1');
            else
                s.push_back('0');
            mask >>= 1;
        }
        return s;
    }
    else
    {
        // str is an immediate number
        /* 将字符串转化为数字即可 */
        return NumberToAssemble(str).substr(16 - opcode_length);
    }
}
```

# assemble

下面选出to be done的部分

将小写转化为大写

```
            // Convert `line` into upper case
            for (int i = 0, length = line.length(); i < length; i++)
            {
                if ('a' <= line[i] && line[i] <= 'z')
                    line[i] += 'A' - 'a';
            }
```

使用这段代码之前得到的comment_position进行切割

```
            // Split content and comment
            auto content_str = line.substr(0, comment_position);
            auto comment_str = line.substr(comment_position);
```

判断数据是否正常并记录地址

```
        else if (pseudo_command == ".FILL")
        {
            std::string word;
            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;
            }
            file_address[line_index] = line_address;
            line_address++;
        }
        else if (pseudo_command == ".BLKW")
        {
            file_address[line_index] = line_address;
            std::string number;
            line_stringstream >> number;
            auto num = RecognizeNumberValue(number);
            if (num == std::numeric_limits<int>::max())
            {
                // @ Error Invalid Number input @ BLKW
                return -7;
            }
            if (num <= 0)
                return -7;
            //@error
            line_address += num;
        }


    if (IsLC3Command(word) != -1 || IsLC3TrapRoutine(word) != -1)
    {
        // * This is an operation line
        file_tag[line_index] = lOperation;
        continue;
    }
```

存储label，若该行只有一个label则需要特殊考虑

```
        word = "";
        line_stringstream >> word;
        if (IsLC3Command(word) != -1 || IsLC3TrapRoutine(word) != -1 || word == "")
        {
            // a label used for jump/branch
            file_tag[line_index] = lOperation;
            label_map.AddLabel(label_name, value_tp(vAddress, line_address - 1));
            if (word == "")
            {
                file_tag[line_index] = lComment;
                line_address--;
            }
        }


if (word == ".BLKW")
            {
                // modify label map
                // modify line address
                label_map.AddLabel(label_name, value_tp(vAddress, line_address - 1));
                std::string number;
                line_stringstream >> number;
                auto num = RecognizeNumberValue(number);
                if (num == std::numeric_limits<int>::max())
                {
                    // @ Error Invalid Number input @ BLKW
                    return -7;
                }
                if (num <= 0)
                    return -7;
                //@error
                line_address += num - 1;
            }
            if (word == ".STRINGZ")
            {
                // modify label map
                // modify line address
                label_map.AddLabel(label_name, value_tp(vAddress, line_address - 1));
                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 - 1;
            }
```

将伪指令变为机械码

```
else if (word == ".BLKW")
            {
                // Fill 0 here
                std::string number_str;
                line_stringstream >> number_str;
                auto num = RecognizeNumberValue(number_str);
                std::string output_line;
                if (gIsHexMode)
                    output_line = "0000";
                else
                    output_line = "0000000000000000";
                output_file << output_line << std::endl;
                if (gIsDebugMode == 1)
                    for (int i = 1; i < num; i++)
                    {
                        output_file << std::hex << file_address[line_index] + i << ": ";
                        output_file << output_line << std::endl;
                    }
                else
                    for (int i = 1; i < num; i++)
                    {
                        output_file << output_line << std::endl;
                    }
            }
            else if (word == ".STRINGZ")
            {
                // Fill string here
                std::string str;
                line_stringstream >> str;
                int num = str.length();
                if (gIsHexMode)
                {
                    output_file << ConvertBin2Hex(NumberToAssemble((int)str[1])) << std::endl;
                    if (gIsDebugMode == 1)
                        for (int i = 2; i < num - 1; i++)
                        {
                            output_file << std::hex << file_address[line_index] + i << ": ";
                            output_file << ConvertBin2Hex(NumberToAssemble((int)str[i])) << std
                        }
                    else
                        for (int i = 2; i < num - 1; i++)
                        {
                            output_file << ConvertBin2Hex(NumberToAssemble((int)str[i])) << std
                        }
                }
                else
                {
                    output_file << NumberToAssemble((int)str[1]) << std::endl;
                    if (gIsDebugMode == 1)
                        for (int i = 2; i < num - 1; i++)
                        {
                            output_file << std::hex << file_address[line_index] + i << ": ";
                            output_file << NumberToAssemble((int)str[i]) << std::endl;
                        }
```

```
            else
                for (int i = 2; i < num - 1; i++)
                {
                    output_file << NumberToAssemble((int)str[i]) << std::endl;
                }
        }
    }



    // Convert comma into space for splitting
    replace(parameter_str.begin(), parameter_str.end(), ',', ' ');
```

对每个指令的翻译

```
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 += NumberToAssemble(parameter_list[0]).substr(8);
        break;
    default:
        // Unknown opcode
        // @ Error
        break;
    }
```

# 总结

通过本次lab，我深刻了解了汇编器的作用原理，并且能够近距离了解一个好的代码框架，并且在框架下可以最终实现一个汇编器