LABA 实验报告

整体思路

- 进行两次扫描。
- 第一次整体扫描得到代码总体长度以及LABEL，伪操作等占用的实际地址，为第二次扫描中的跳转做铺垫。
- 第二次扫描中如遇LABEL或地址的跳转则可以根据第一次扫描得到的位置进行作差得到PC应该跳转的值，其余部分直接进行翻译即可。

具体实现

assembler.h

- 枚举型变量储存每段代码状态，vector容器储存编译指令，方便第二次扫描进行比对并进行转换。
- 使用类储存每段代码对应状态及所带有具体值的类型。
- 定义函数：
  - TRIM（去除输入字符串前后多余字符）

```
// trim from left
inline std::string &LeftTrim(std::string &s, const char *t = "
\t\n\r\f\v")
{
    // TO BE DONE
    s.erase(0, s.find_first_not_of(t));
    return s;
}

// trim from right
inline std::string &RightTrim(std::string &s, const char *t = "
\t\n\r\f\v")
{
    // TO BE DONE
    s.erase(s.find_last_not_of(t) + 1);
    return s;
}
```

  - 判断是LC3指令抑或是TRAP指令

```
inline int IsLC3Command(const std::string &str);
inline int IsLC3TrapRoutine(const std::string &str);
```

  - 十进制数与字符的相互转化，二进制与十六进制的转化

```
inline char DecToChar(const int &num);
inline int CharToDec(const char &ch);
std::string ConvertBin2Hex(std::string bin);
```

  - 分析数字并转换为汇编语言

```
int RecognizeNumberValue(std::string str);
std::string NumberToAssemble(const int &number);
std::string NumberToAssemble(const std::string &number);
```

assembler.cpp

- 将某一字符串s转化为数字
  - 可以根据LC-3汇编语言的输入规范分为三类：#开头的十进制数，x开头的十六进制数，以及除此之外的二进制数，以此进行对应转换。

```
switch (s[0])
    {
    case '#':
        if (s[1] == '-')
        {
            for (int i = len; i > 1; i--)
            {
                val += CharToDec(s[i]) * a;
                a *= 10;
            }
            val = -val;
        }
        else
        {
            for (int i = len; i > 0; i--)
            {
                val += CharToDec(s[i]) * a;
                a *= 10;
            }
        }
        break;
    case 'x':
    case 'X':
        if (s[1] == '-')
        {
            for (int i = len; i > 1; i--)
            {
                val += CharToDec(s[i]) * a;
                a *= 16;
            }
            val = -val;
        }
        else
        {
            for (int i = len; i > 0; i--)
            {
                val += CharToDec(s[i]) * a;
                a *= 16;
            }
        }
        break;
    default:
        if (s[1] == '-')
        {
            for (int i = len; i > 1; i--)
            {
```

```
                val += CharToDec(s[i]) * a;
                a *= 2;
            }
            val = -val;
        }
        else
        {
            for (int i = len; i > 0; i--)
            {
                val += CharToDec(s[i]) * a;
                a *= 2;
            }
        }
        break;
    }
```

- 将某一数值转化为16位2进制数
    - 由于使用补码储存，故先判断数值的正负性，然后再针对处理。

```
std::string NumberToAssemble(const int &number)
{
    // Convert the number into a 16 bit binary string
    // TO BE DONE
    if (number > 65535 || number < -65536)
    {
        // @ Error Too large or too small value  @ FILL
        return NULL;
    }
    std::string res = "";
    if (number < 0)
    {
        int ans_ = -number - 1;
        while (ans_)
        {
            res.insert(res.begin(), DecToChar(ans_ % 2));
            ans_ /= 2;
        }
        int len = 16 - res.size();
        for (int i = 0; i < len; i++)
        {
            res.insert(res.begin(), '0');
        }
        for (int i = 0; i < 16; i++)
        {
            if (res[i] == '1')
                res[i] = '0';
            else
                res[i] = '1';
        }
    }
    else if (number == 0)
    {
        res += "0000000000000000";
    }
    else
    {
```

```
        int ans = number;
        while (ans)
        {
            res.insert(res.begin(), DecToChar(ans % 2));
            ans /= 2;
        }
        int len = 16 - res.size();
        for (int i = 0; i < len; i++)
        {
            res.insert(res.begin(), '0');
        }
    }
    return res;
}
```

- 将某一代表数字的字符串转换为16位2进制数

```
std::string NumberToAssemble(const std::string &number)
{
    int num = RecognizeNumberValue(number);
    return NumberToAssemble(num);
}
```

- 将二进制转换为十六进制
  - 将二进制数前加上b然后利用上述函数进行转换。

```
std::string ConvertBin2Hex(std::string bin)
{
    bin.insert(bin.begin(), 'b');
    int num = RecognizeNumberValue(bin);
    std::string res;
    while (num)
    {
        res.insert(res.begin(), DecToChar(num % 16));
        num /= 16;
    }
    return res;
}
```

- 翻译操作码
  - 如果得到的是一个LABEL，则利用第一次扫描中储存的位置减去当前扫描到LABEL时的位置得到一个跳转值。

```
if (!(item.getType() == vAddress && item.getVal() == -1))
    {
        // str is a label
        // TO BE DONE
        int num_temp = item.getVal() - current_address - 1;
        std::string res = NumberToAssemble(num_temp);
        std::string ans;
        for (int i = 15 - opcode_length + 1; i <= 15; i++)
        {
            ans += res[i];
        }
        return ans;
    }
```

- 若为寄存器类型，转换为三位二进制数

```
    if (str[0] == 'R')
    {
        // str is a register
        // TO BE DONE
        std::string ans;
        std::string res = NumberToAssemble(str[1]);
        for (int i = 13; i <= 15; i++)
            ans += res[i];
        return ans;
    }
```

- 若为立即数，则直接转换

```
    else
    {
        // str is an immediate number
        // TO BE DONE
        auto num = RecognizeNumberValue(str);
        std::string ans;
        std::string res = NumberToAssemble(num);
        for (int i = 15 - opcode_length + 1; i <= 15; i++)
            ans += res[i];
        return ans;
    }
```

- 第一遍读取
  - 将指令全部转换为大写以方便匹配。但值得注意的是，若直接对文件的整行读入直接进行transform则会将.STRINGZ后所携带的字符串一并转换为大写，这显然是不合理的，我们需要对.STRINGZ进行特殊处理，即只将这一伪操作转为大写而不处理后续“”中的内容。

```
            // Convert `line` into upper case
            // TO BE DONE
            std::string tem = line;
            std::transform(line.begin(), line.end(), line.begin(),
::toupper);
            auto i = line.find(".STRINGZ");
            if (i != std::string::npos)
            {
                i += 8;
                line = line.replace(i, line.size() - i, tem.substr(i));
            }
```

- 若读取到注释，将注释与指令分开

```
                // Split content and comment
                // TO BE DONE
                std::string comment_str;
                std::string content_str;
                int i;
                for (i = 0;; i++)
                {
                    if (line[i] == ';')
                        break;
                    else
                        content_str += line[i];
                }
                for (int j = i + 1; i < line.size(); i++)
                {
                    comment_str += line[j];
                }
```

- 若读取到对应的伪操作，根据伪操作的类型对当前读取到的地址进行跳转。若读取到直接进行的操作则进行标记，若读取到跳转指令则标记并且将跳转标签加入map。

- 第二遍扫描
  - 从头进行读取，根据当前地址与第一次储存的地址的差值计算PC跳转值。
  - 根据不同的伪操作向LC-3填入对应值。
    - 值得说明的是，对于STRINGZ，若采用框架所给出的方式——即将每行代码作为一个输入流然后以空格为间隔逐段读取——将造成后续的“”中的内容第一次读取只会读到“XXX，进而导致立即报错，故需要将.STRINGZ后的全部字符包括中间的空格一并读取。

```
            else if (word == ".STRINGZ")
            {
                // Fill string here
                // TO BE DONE
                std::string number_str;
                std::getline(line_stringstream, number_str);
                number_str = Trim(number_str);
                if (number_str[0] != '\"' ||
number_str[number_str.size() - 1] != '\"')
                    return -10;
                int num_temp;
                for (int i = 1; i < number_str.size() - 1; i++)
                {
```

```
                    num_temp = (int)number_str[i];
                    output_file << NumberToAssemble(num_temp) <<
std::endl;
                }
            }
```

- 对于一般的操作，将逗号转换为空格方便处理
- 此后就是对对应的汇编指令转换为对应的机器码，例如
  - ADD

```
                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;
```

- .ORIG与.END的处理。当读取到.ORIG时，将后部数值作为初始地址；当读取到.END时，地址为-1，宣告读取的结束。

main.cpp

- 调整输出模式，进行文件的读取，然后转换为机器码后输出到对应文件中。

makefile

```
CC=g++
CFLAGS=-I. -g                       //编译设置
DEPS = assembler.h                  //直接包含对应文件，自动生成依赖关系
OBJ = assembler.o main.o            //标记目标
```

- 在目标文件目录下使用make指令，最后生成三个文件

  ```
  assembler.exe
  assembler.o
  main.o
  ```

  与预期一致。

- 最后执行exe文件，则会自动读取main中所指定格式的输入文件并以指定格式输出得到机器码。