

一、 实验目的与方法

1. 实验目的

- ① 加深对编译器总体结构的理解与掌握
- ② 加深对汇编指令的理解与掌握
- ③ 对指令选择，寄存器分配有较深的理解。

2. 实验方法

将中间代码所给的地址码生成目标代码，采用 X86 的汇编指令。

3. 实验语言

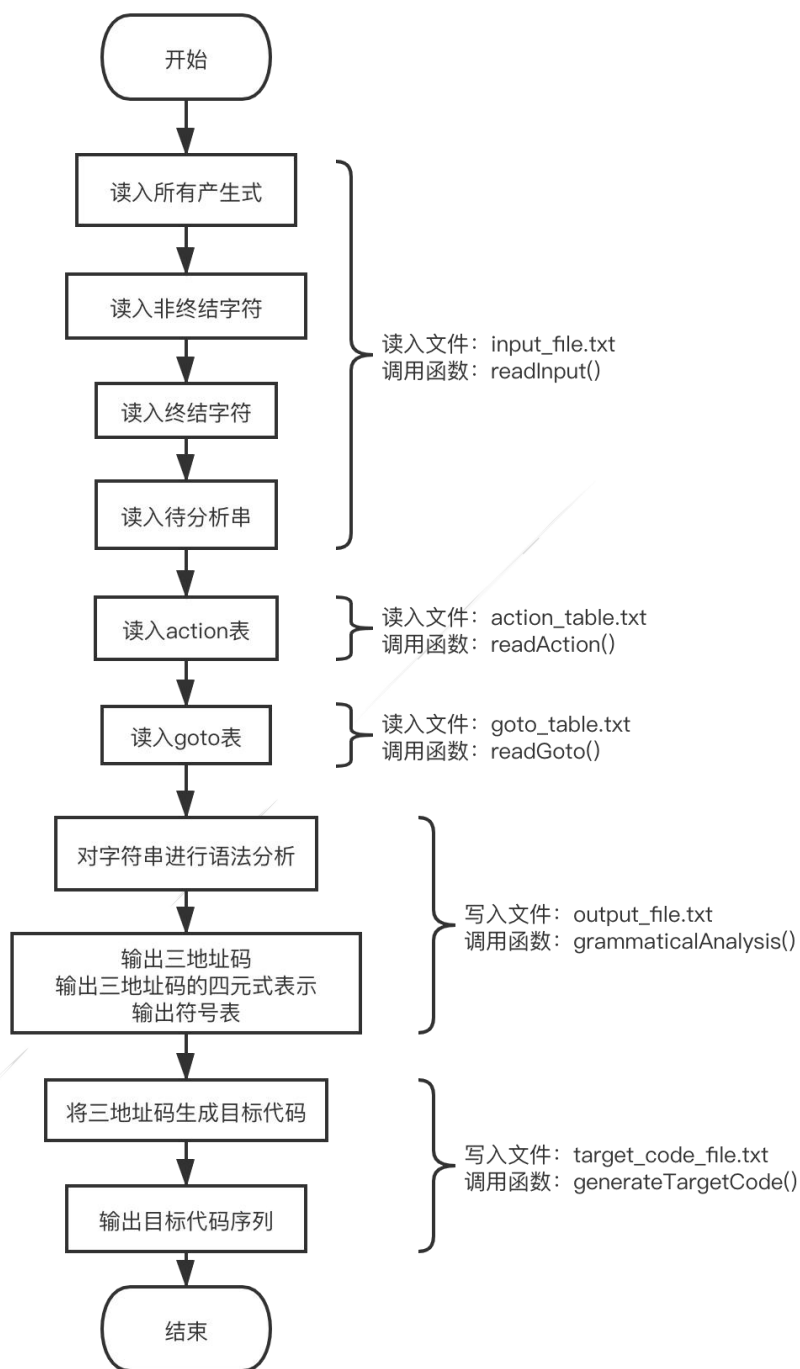
所使用的语言为 C++语言；操作系统环境为 macOS Mojave；软件环境为 CLion。

二、 实验总体流程与函数功能描述

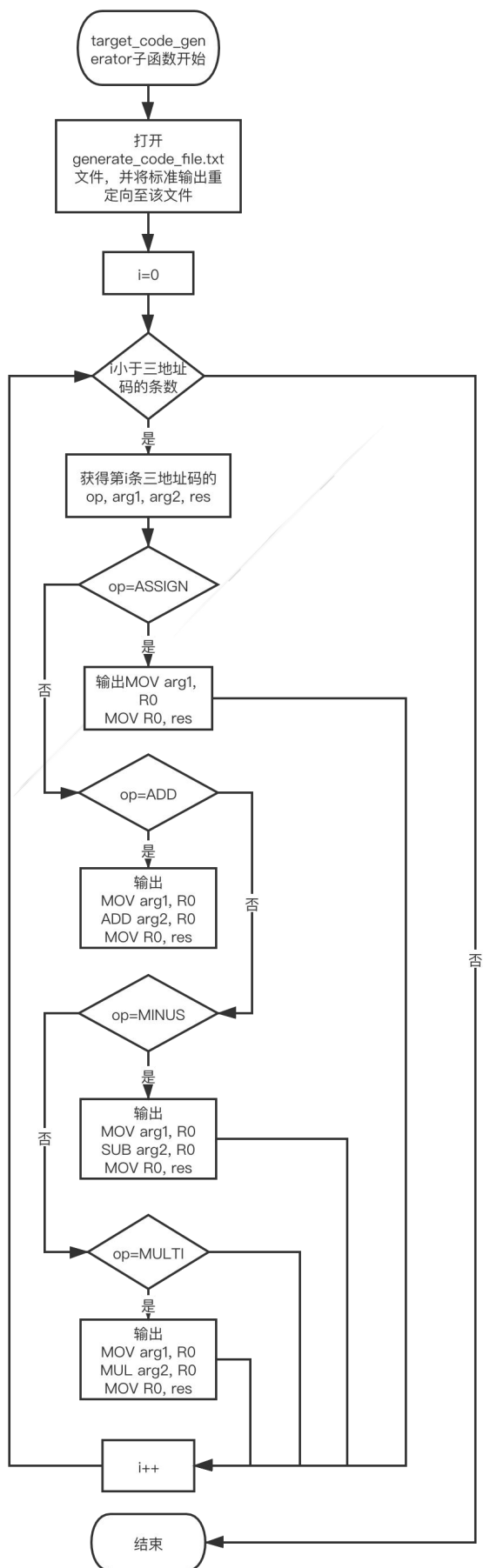
实验四是在实验三的基础上开发的，读入函数、语法分析函数基本不变，主要增加了根据实验三生成的三地址码生成目标代码的的函数。

下面展示整体框架的流程图与目标代码生成函数的流程图。

① 整体框架



② 目标代码生成函数 void target_code_generator()流程图如下:



三、 实验内容

① 四元式的数据结构

```
/* 四元组结构体 */  
struct Quaternion  
{  
    int op;  
    string arg1;  
    string arg2;  
    string res;  
};  
vector<Quaternion> quaternion_table;
```

在实验三中，我将三地址码以更为直观的形式输出，在这样的形式不利于目标代码的生成。于是我在生成三地址码的代码中，增加了将三地址码以四元式存储的功能。op 表示运算符，arg1 表示第一个操作数，arg2 表示第二个操作数，res 表示结果。

每生成一个三地址码，就将其转化为该四元式的形式，插入到四元式表 quaternion_table 中，新增代码如下（只展示赋值语句和减法语句，因为加法、乘法、除法与减法类似）：

```
if (p_num == 1){  
    string fir = tri.back();  
    tri.pop_back();  
    string sec = tri.back();  
    tri.pop_back();  
    quaternion_tmp.op = ASSIGN;  
    quaternion_tmp.arg1 = fir;  
    quaternion_tmp.res = sec;  
    quaternion_table.push_back(quaternion_tmp);  
    cout << sec + " = " + fir << endl;  
}  
  
} else if (p_num == 3){  
    string fir = tri.back();  
    tri.pop_back();  
    string sec = tri.back();  
    tri.pop_back();  
    tnum++;  
    string tmpn;  
    tmpn = to_string(tnum);  
    tmpn = "t" + tmpn;  
    tri.push_back(tmpn);  
    quaternion_tmp.op = MINUS;  
    quaternion_tmp.arg1 = sec;  
    quaternion_tmp.arg2 = fir;  
    quaternion_tmp.res = tmpn;  
    quaternion_table.push_back(quaternion_tmp);  
    cout << tmpn << " = " + sec + " - " + fir << endl;  
} else if (p_num == 5) {
```

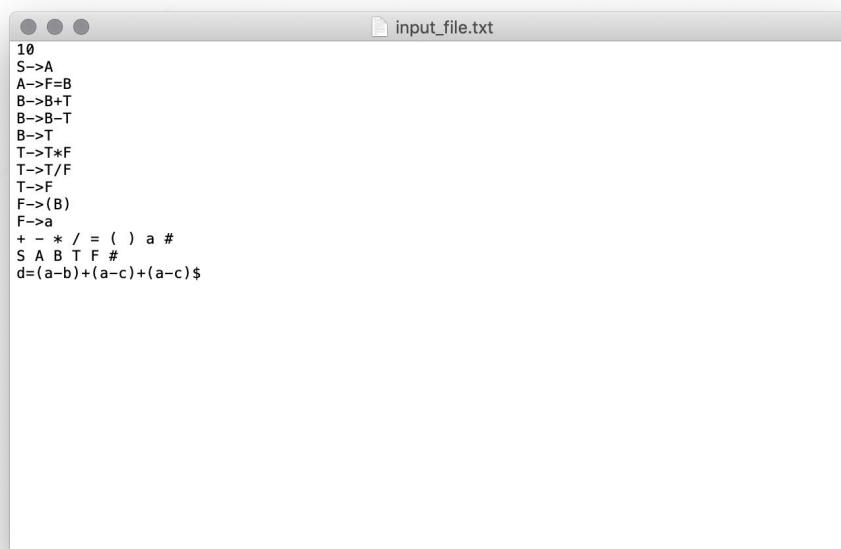
② 生成代码序列

```
MOV a, R0
SUB b, R0
MOV R0, t1
MOV a, R0
SUB c, R0
MOV R0, t2
MOV t1, R0
ADD t2, R0
MOV R0, t3
MOV a, R0
SUB c, R0
MOV R0, t4
MOV t3, R0
ADD t4, R0
MOV R0, t5
MOV t5, R0
MOV R0, d
```

四、实验结果与分析

① 实验输入

action_table.txt 和 goto_table.txt 的输入与实验三相同，有所不同的是 input_file。将待分析的字符串修改为： $d=(a-b)+(a-c)+(a-c)$



```
10
S->A
A->F=B
B->B+T
B->B-T
B->T
T->T*F
T->T/F
T->F
F->(B)
F->a
+ - * / = ( ) a #
S A B T F #
d=(a-b)+(a-c)+(a-c)$
```

② 实验输出

```
output_file.txt
***** 三地址码 *****
t1 = a - b
t2 = a - c
t3 = t1 + t2
t4 = a - c
t5 = t3 + t4
d = t5
***** 三地址码的四元式表示 *****
op      arg1      arg2      result
2       a         b        t1
2       a         c        t2
1       t1        t2        t3
2       a         c        t4
1       t3        t4        t5
0       t5                d
***** 符号表 *****
名字    符号种类  类型    地址    扩展属性指针
c       变量      int     0       NULL
b       变量      int     4       NULL
a       变量      int     8       NULL
d       变量      int    12       NULL
```

```
target_code_file.txt
***** 目标代码序列 *****
MOV a, R0
SUB b, R0
MOV R0, t1
MOV a, R0
SUB c, R0
MOV R0, t2
MOV t1, R0
ADD t2, R0
MOV R0, t3
MOV a, R0
SUB c, R0
MOV R0, t4
MOV t3, R0
ADD t4, R0
MOV R0, t5
MOV t5, R0
MOV R0, d
```