课 程 设 计 报 告

设计题目：C语言到8086汇编语言编译器

班　　级：计算机1606班

组长学号：20164465

组长姓名：马莹莹

指导教师：李晓华

设计时间：2018 年 12 月

设计分工

组长学号及姓名：20164465 马莹莹

分工：

1. 整体架构设计、中间代码设计、中间代码转换为目标代码方法设计
2. 变量声明（包括数组）及符号表设计实现
3. 算数表达式设计实现
4. 函数声明与函数调用设计实现
5. 中间代码优化
6. 目标代码生成中以上部分设计与实现
7. 可视化界面实现

组员 1 学号及姓名：20164504 赵祉怡

分工：

1. 循环结构 while 与 for 设计实现
2. 目标代码生成中循环结构的设计与实现
3. 文法提取
4. 程序测试

组员 2 学号及姓名：20164713 张馨月

分工：

1. 分支结构 if 与 switch 设计实现
2. 目标代码生成中分支结构的设计与实现
3. 可视化界面设计
4. 程序测试

摘要

一个编译器的工作一般可以划分为五个模块：词法分析、语法分析、语义分析及中间代码生成、中间代码优化和目标代码生成。

本文设计并实现了一种基于递归下降法用以将类 C 语言翻译为 8086 汇编语言的简单编译器。它拥有与用户交互的良好界面，所设计的文法支持整数类型及数组的定义和运算，函数的定义与调用，if、switch 分支语句，while、for 循环语句等功能，并允许各模块之间相互嵌套使用。

具体地，我们通过词法分析器将输入的 C 语言源代码输出为可以区分变量名、数字、关键字、字符与字符串的 token 序列并对其中的错误输入进行位置提示。再通过递归下降语法分析器将 token 序列转化为已设计的四元式中间代码，同时填写符号表。该编译器拥有中间代码优化模块，可对四元式以基本块为单位进行常数合并、删除多余运算、删除无用赋值及其它多种方式的优化。最后结合符号表与优化后的四元式，生成可执行的 8086 汇编语言。实验证明，该方法与程序效果良好，具备基本的可扩展性和健壮性。

关键词：编译原理，编译器前端，编译器后端，递归下降，8086 汇编语言

目　录

1 概述

编译原理课程兼有很强的理论性和实践性，是计算机专业的一门非常重要的专业基础课程，在系统软件中占有十分重要的地位。编译原理课程设计是本课程重要的综合实践教学环节，是对平时实验的一个补充。通过编译器相关子系统的设计，使学生能够更好地掌握编译原理的基本理论和编译程序构造的基本方法和技巧，融会贯通本课程所学专业理论知识；培养学生独立分析问题、解决问题的能力，以及系统软件设计的能力；培养学生的创新能力及团队协作精神。

编译程序是一种翻译程序，它特指把某种高级程序设计语言翻译成具体计算机上的低级程序设计语言。本次我们小组设计并实现了一种基于递归下降法用以将类C语言翻译为8086汇编语言的简单编译器，设计的主要方面如下：

1.文法的设计，包括分支、循环、函数、数组等语句

2.符号表的设计，保存用户自定义变量、数组及临时变量，能够区分形参、实参并能通过偏移地址找出各变量，为目标代码生成打下基础。

3.词法分析器，识别变量名、数字、关键字、字符与字符串并生成TOKEN序列

4.语法分析器，识别源代码生成的TOKEN串是否符合文法

5.语义分析器，设计翻译文法，将 token 序列转化为已设计的四元式中间代码，同时填写符号表

6.中间代码优化，基于基本块的局部优化

7.目标代码生成，生成可执行的8086汇编代码

8.GUI，友好的人机交互界面

2 课程设计任务及要求

2.1 设计目的

编译原理课程兼有很强的理论性和实践性，是计算机专业的一门非常重要的专业基础课程，在系统软件中占有十分重要的地位。编译原理课程设计是本课程重要的综合实践教学环节，是对平时实验的一个补充。通过编译器相关子系统的设计，使学生能够更好地掌握编译原理的基本理论和编译程序构造的基本方法和技巧，融会贯通本课程所学专业理论知识；培养学生独立分析问题、解决问题的能力，以及系统软件设计的能力；培养学生的创新能力及团队协作精神。

2.2 设计内容

在下列内容中任选其一：

1、一个简单文法的编译器前端的设计与实现。

2、一个简单文法的编译器后端的设计与实现。

3、一个简单文法的编译器的设计与实现。。

4、自选一个感兴趣的与编译原理有关的问题加以实现，要求难度相当。

2.3 设计要求

1、在深入理解编译原理基本原理的基础上，对于选定的题目，以小组为单位，先确定设计方案；

2、设计系统的数据结构和程序结构，设计每个模块的处理流程。设计合理；

3、编程序实现系统，要求实现可视化的运行界面，界面应清楚地反映出系统的运行结果；

4、确定测试方案，选择测试用例，对系统进行测试；

5、运行系统并要通过验收，讲解运行结果，说明系统的特色和创新之处，并回答指导教师的提问；

6、提交课程设计报告。

3 算法与数据结构

3.1 算法的总体思想

总体思想在编译器的入口函数 Main.java 中得到了很好的体现。

先调用词法分析器将输入的简单 C 语言源代码输出为可以区分变量名、数字、关键字、字符与字符串的 token 序列。

再新建符号表类，用来记录接下来对符号表的一切修改。

随后调用 block.java, block.java 将会把 token 序列划分为各个基本块，然后分发给 exp_four、while_four、for_four、if_four、switch_four、function_four、define_local、define_global 模块。再将返回的四元式中间代码加在一起输出，在这个过程中符号表也已经填好。

优化器 optimization.java 运行三步优化后，object_code.java 结合符号表与优化后的中间代码，生成可执行的 8086 汇编语言。

3.2 词法分析器

3.2.1 功能

输入 C 语言源程序，输出 token 序列。

3.2.2 数据结构

String[] k ： 关键字

List<String> I ： 变量名

List<String> C ： 字符

List<String> S ： 字符串

List<Double> c ： 数字

List<String> p ： 符号

List<List<String>> fr ： 最终返回的结果，包括 token 序列及其对照表

int status ： 代表 ascii 码的识别结果，0 为空格，1 字母，2 数字，3 双引号代表字符串，4 单引号代表字符，5 是其他符号

String result ： 以字符串的形式存放 token。因为在最开始不可能知道共有多少 token，所以不可能使用数组。最开始的想法是使用 list 虽然方便，但写入文件时仍然需要一步 toString，下次调用时也要转化为数组，使用字符串与 concat 来存 token，写入文件时可以直接写入，而调用时 split 即可。后来发现要用 List<List<String>>来返回，还是应该使用 List<String>更合适。

3.2.3 算法

analyzer.java 先对输入的文本进行预处理，去掉换行与多余的空格。再根据读取到的字符的 ascii 码判断它是字母、数字还是符号。

如果当前单词在关键词列表中，则是 k 类型关键字。

如果当前单词第一位为字母，后面是字母或数字或[ ]，则当前单词为 i 类型（变量名）。

如果当前单词第一位为数字且后续都是，则该单词为 c 类型数字。

如果当前单词为符号，先检测它是否是<= >= != ==这种双字节符号，再检测它是否为单双引号作为字符与字符串的开始，如果都不是，则自身为一个 token。

3.3 基本块

3.3.1 功能

该模块对应代码中的 block.java。功能是输入整体的 token 序列，将其分发至变量声明模块、函数声明与调用模块、循环结构模块、分支结构模块、算数表达式模块，获取它们返回的四元式并组合，输出程序整体的四元式和填好的符号表。

3.3.2 数据结构

String[] step, i, C, S, c, k, p ： 输入的 token 序列（step）和对照表

table tb ： 输入的符号表

List<String[]> qt ： 存放四元式的 list

List<String[]> sentence ： 存放被分开的 token 序列

3.3.3 算法

整体来说，block.java 主要分为两大块内容，其一为将 token 序列按照类型划分为数块，其二为按照已划分的块分别调用不同的中间代码生成程序。

划分块是根据关键字与特征划分的。

声明语句以 int/char/int[]/string 开始，如果开始后的第二个字符是“（”，则不是变量声明而是函数声明。变量声明以“；”结尾而函数声明以最开始的{对应的}结尾。

if 语句，以 if 开始，以}为结尾，并且后面可能跟着一个 else。

While、for、switch 语句都以关键字开始而以}结尾。

return、break、continue 则是只有关键字与后面一位的分号。

除此之外，还有算数表达式，有效的算数表达式第一个是变量，第二个是“=”。

在划分完块以后，检查已划分的 token 数量是否等于总 token 数量，相等则已经正确划分，不相等则意味着有未知语句，应该报错。

获取了划分好的基本块以后，则可以调用对应函数了。

3.4 符号表结构

3.4.1 功能

根据声明语句存放变量与函数的基本信息及偏移地址。可做重定义判别。

3.4.2 数据结构

```
public class table {
    class vari {//  全局变量信息
        String name;//  变量名
        String tp;//  类型：int int[] function char string
        int ofad;//  偏移地址
        int other;//（如类型为函数，则指向函数表中的某一个。如类型为数组，则存放数组长度）
    }
```

```
class func {// 函数信息
    String name;// 函数名
    List<String> xctp = new ArrayList<String>();// 形参类型
    List<String> xcname = new ArrayList<String>();// 形参名
    List<vari> vt = new ArrayList<vari>();// 函数中的临时变量
}

List<vari> synbl = new ArrayList<vari>();// 全局变量总表
List<func> pfinfl = new ArrayList<func>();// 函数表
List<String> vall = new ArrayList<String>();// 活动记录，记录当前位置，是子函数还是主函数，来决定调用临时变量还是全局变量
}
```

实例说明：

```
总表:
变量名: x1 类型: int 偏移地址: 0 其他: -1
变量名: x2 类型: int 偏移地址: 1 其他: -1
变量名: x3 类型: int 偏移地址: 2 其他: -1
变量名: a 类型: int 偏移地址: 3 其他: -1
变量名: b 类型: int 偏移地址: 4 其他: -1
变量名: w1 类型: int 偏移地址: 5 其他: -1
变量名: f 类型: function 偏移地址: 0 其他: 0
变量名: i 类型: int 偏移地址: 6 其他: -1


函数表:

函数名: f
形参:
int c
int d
函数的局部变量:
变量名: c 类型: int 偏移地址: 0 其他: 0
变量名: d 类型: int 偏移地址: 1 其他: 0
变量名: e 类型: int 偏移地址: 2 其他: -1
```

3.5 变量声明

3.5.1 功能

变量声明分为全局变量声明与局部变量声明。之所以要区分这两个，是因为全局变量存放在数据段而局部变量存放在堆栈段。该模块输入为类似于"int a;"

的代码对应的 token 序列、iCSckp 表、符号表，无输出但修改符号表。

define_global.java 是全局变量声明，调用此函数声明的变量会被加入符号表总表，而调用 define_local.java 的则是局部变量声明，会被加入符号表-函数表。

3.5.2 数据结构

String[] step, i, C, S, c, k, p ： 输入的 token 序列（step）和对照表

table tb ： 输入的符号表

3.5.3 算法

在符号表设计好以后，变量声明的难点主要是如何区分调用全局变量声明还是局部变量声明。我们的方法是在发现声明函数的语句时向符号表的活动记录加入函数名，再进行函数声明的递归下降，递归返回后再将函数名从符号表的活动记录中清除。每次遇到变量声明则查询符号表的活动记录是否只有“main”，如果不是，那么它就是当前的函数名，正在声明的变量就应该加入这个函数表中函数名的变量 list 里。

3.6 分支结构

3.6.1 功能

能够分析 if 语句，准确无误的生成 if 语句对应的四元式。对 if 语句的条件进行判断是否是复杂的算数表达式，并生成对应的四元式；对 else 进行判断，由此决定是否生成 else 所对应的四元式。

能够分析 switch 语句，准确无误的生成 switch 语句对应的四元式。对 case 和 default 进行判断，（可以有多个 case，只有一个 default，并且 case 和 default 的顺序可以不固定）并生成对应的四元式

3.6.2 数据结构

If:

public static String[] step, i, C, S, c, k, p;//输入

public static List<String[]> qt//存四元式

String T1,T2,P;　　//保存代表左式的临时变量，保存代表右式的临时变量，保存比较的符号

String t;　　//Token 的值

String[] step_son;　//各阶段的子 token 序列

List<String[]> qtt; //各阶段生成的四元式

List<List<String>> anal

Step//token 序列

List<String[]> r1//该方法

Switch：

public static String[] step, i, C, S, c, k, p;//输入

public static List<String[]> qt //存四元式

String T1,T2,P;　　//保存代表左式的临时变量，保存代表右式的临时变量，保存比较的符号

String[] step_son;　//各阶段的子 token 序列

List<String[]> qtt; //各阶段生成的四元式

List<List<String>> anal

Step Step//token 序列

List<String[]>r1//该方法

示例：

```
1   if((a+4)*2<b)
2   {
3       a=a+1;
4   }
5   else
6   {
7       b=b+1;
8   }
```

问题 输出 调试控制台 终端

```
原始四元式：
+ a 4.0 t1
* t1 2.0 t2
< t2 b t3
if t3 _ _
+ a 1.0 a
es _ _ _
+ b 1.0 b
ie _ _ _

优化后的四元式：
+ a 4.0 t1
* t1 2.0 t2
< t2 b t3
if t3 _ _
+ a 1.0 a
es _ _ _
+ b 1.0 b
ie _ _ _
```

图 1 if 示例

图 2 switch 示例

3.6.3 算法

If：对 if 语句的条件进行判断是否是复杂的算数表达式，并生成对应的四元式；对 else 进行判断，由此决定是否生成 else 所对应的四元式。

Switch：对 case 和 default 进行判断，（可以有多个 case，只有一个 default，并且 case 和 default 的顺序可以不固定）并生成对应的四元式

3.7 循环结构

3.7.1 功能

扫描相应的 token 序列，生成 while 和 for 语句的相应中间代码四元式。

3.7.2 数据结构

String[] step,i,C,S,c,k,p :输入的 token 序列(step)和对照表

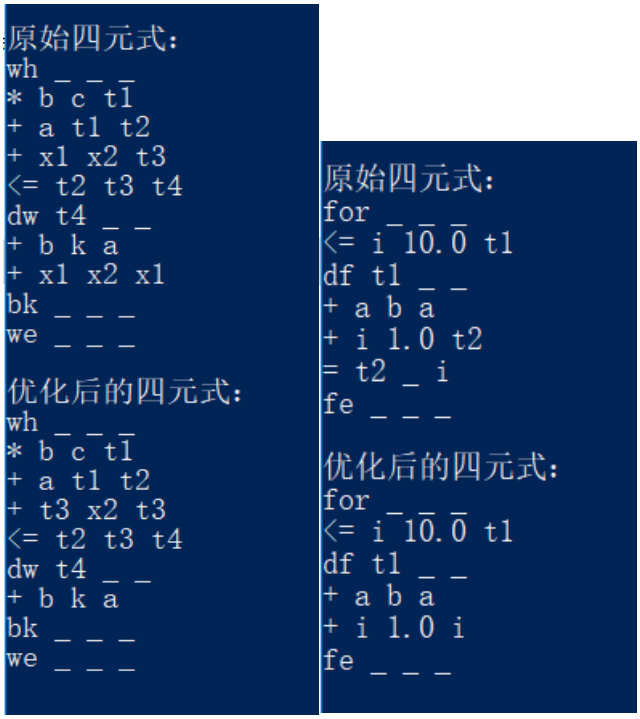
table tb： 输入的符号表

List<String[]>qt :存放四元式的 list

List<String[]>sentence ： 存放被分开的 token 序列，如：while(?){?}的全部 token

序列

示例：



3.7.3 算法

while：

扫描 token 序列，碰到关键字 while 生成四元式(wh,_,_,_)，记录比较左式的开始 token 序号，循环向下扫描直到遇到比较符号，将左式 token 子序列取出调用 exp_four()函数生成四元式，并记录比较符号和代表左式的临时变量；记录比较右式的开始 token 序号，循环向下扫描直到遇到左花括号，将右式 token 子序列取出调用 exp_four()函数生成四元式，并记录代表右式的临时变量，生成(“>”,T 左，T 右，tn)和(de, tn, _, _);记录 while 主程序的开始 token 序号，循环向下扫描直到左右花括号数目相等，将主式 token 子序列取出调用 block()函数生成四元式后生成(we, _, _, _)代表 while 语句结束。

for:

扫描 token 序列，碰到关键字 for 记录 for1 表达式的开始 token 序号，循环向下扫描直到遇到分号，将 for1 表达式子序列取出调用 exp_four()函数生成四元式后生成四元式(for, _, _, _)；记录 for2 表达式左式的开始 token 序号，循环向下扫描直到遇到比较符号，将左式 token 子序列取出调用 exp_four()函数生成四元式，

并记录比较符号和代表左式的临时变量；记录比较右式的开始 token 序号，循环向下扫描直到遇到分号，将右式 token 子序列取出调用 exp_four()函数生成四元式，并记录代表右式的临时变量，生成(“>”,T 左，T 右，tn)和(de, tn , _, _)；记录 for3 表达式开始位置，循环向下扫描直到遇到左花括号，记录 for 主程序的开始 token 序号，循环向下扫描直到左右花括号数目相等，将主式 token 子序列取出调用 block()函数生成四元式，将 for3 表达式子序列取出调用 exp_four()函数生成四元式后生成(fe, _, _, _)代表 for 语句结束。

3.8 函数定义与调用

3.8.1 功能

输入函数声明与调用对应的 token 序列和 iCSckp 表，输出四元式中间代码。

3.8.2 数据结构

String[] step, i, C, S, c, k, p ： 输入的 token 序列（step）和对照表

table tb ： 输入的符号表

String fnm ： 从符号表-活动记录读取函数名

List<String> xcname ： 存放形参名

String result ： 存放最终返回的四元式

3.8.3 算法

函数定义分为声明与基本块两个部分，即 int f（int a,int b）和{？}。

其中声明部分调用全局变量声明程序 define_global.java, {?}部分则去大括号调用 block.java。在这个过程中，需要插入以下说明性四元式：

```
[ fun, f, _, _] ———————— 函数名为 f 的函数定义开始，声明前加入
[ rt , c, _, _] ———————— 返回 c 作为函数的定义终结，调用 block 后加入
```

而函数调用模块主要内容则是生成以下四元式：

```
[ sf, f , _, _] ———————— f 开始
[ xc, a, 10, _] ———————— 传递 10 给形参
[ esf, f, a, _] ———————— 函数 f 结束调用，返回值存为 a
```

具体的，在 function_four.java 中，先查询得到函数名，再向 result 加入一条[sf,

函数名 , _, _]，再找到所有实参，以[xc，形参，实参, _]的格式加入 result，最后加入一条[esf，函数名，函数返回, _]代表调用的结束。

z.c语言代码输入.txt

```
int f(int a,int b){
    int c;
    c=a+b;
    return c;
}
int d;
d=f(10,15);
```

```
原始四元式:
fun f _ _
+ a b c
rt c _ _
sf f _ d
xc a 10.0 _
xc b 15.0 _
esf f d _

优化后的四元式:
fun f _ _
+ a b c
rt c _ _
sf f _ d
xc a 10.0 _
xc b 15.0 _
esf f d _
```

```
总表:
变量名: f 类型: function 偏移地址: 0 其他: 0
变量名: d 类型: int 偏移地址: 1 其他: -1


函数表:

函数名: f
形参:
int a
int b
函数的局部变量:
变量名: a 类型: int 偏移地址: 0 其他: 0
变量名: b 类型: int 偏移地址: 1 其他: 0
变量名: c 类型: int 偏移地址: 2 其他: -1


活动记录: main
```

```
DATAS SEGMENT
  d DB ?
DATAS ENDS
STACKS SEGMENT
  STK DB 20 DUP (0)
STACKS ENDS
CODES SEGMENT
  ASSUME CS:CODES,DS:DATAS,SS:STACKS
START:
  MOV AX,DATAS
  MOV DS,AX
  MOV SP,BP
  JMP JMPF0
f PROC NEAR
  MOV AL,SS:[BP]
  ADD AL,SS:[BP+1]
  MOV SS:[BP+2],AL
  MOV AL,SS:[BP+2]
  RET
f ENDP
JMPF0:
  MOV AL,OFFSET 10D
  MOV SS:[BP],AL
  MOV AL,OFFSET 15D
  MOV SS:[BP+1],AL
  CALL f
  MOV [d],AL
  MOV AH,4CH
  INT 21H
CODES ENDS
  END START
PS D:\code\Compiler-NEU-2018>
```

3.9 算数表达式

3.9.1 功能

输入加减乘除和赋值语句的 token 序列，输出对应四元式

3.9.2 数据结构

int now = 0, flag = 1, num = 1, brackets = 0，分别用来记录当前进行到了哪个 token、是否有语法错误（有的话，flag 置 0，每个递归函数都在开始时判断 flag）、临时变量使用到了 t 几（如果用完了 t1，num 应该为 2）、括号是否成对（brackets 在遇到“（”时+1，“）”时-1，若最终 brackets 不为 0，则括号不成对错误）

String[] step, i, C, S, c, k, p ： 输入的 token 序列（step）和对照表

Stack<String> st1 ： 元素栈，用来存放 a+b 中的 a 和 b

Stack<String> st2 ： 符号栈，用来存放 a+b 中的+

List<String[]> qt ： 存四元式

3.9.3 算法

符号栈与元素栈，遇到元素就加进元素栈，遇到符号就加进符号栈。

遇到符号的时候要跟符号栈的栈顶元素比较，操作是：

1．遇+- 栈顶+-*/，出栈　空或 入栈

2．遇*/ 栈顶+-（空 入栈

3．遇） 出栈到（

出栈的意思是，符号栈栈顶为新四元式的第一位，元素栈出栈两个元素作为四元式的第二位第三位，然后新建一个临时变量 tn 为四元式结果，再把这个新临时变量入元素栈。

入栈的意思是，符号入符号栈。

遇到元素全直接入栈。

```
原始四元式：
* b 8.0 t1
+ 10.0 t1 t2
- t2 7.0 t3
- 10.0 c t4
* t4 d t5
+ t3 t5 t6
* b 8.0 t7
+ t6 t7 t8
- 10.0 c t9
* t9 6.0 t10
- t8 t10 t11
* t11 4.0 t12
- t12 e a

优化后的四元式：
* b 8.0 t1
+ 10.0 t1 t2
- t2 7.0 t3
- 10.0 c t4
* t4 d t5
+ t3 t5 t6
+ t6 t1 t8
* t9 6.0 t10
- t8 t10 t11
* t11 4.0 t12
- t12 e a
```

```
总表：
变量名：a 类型：int 偏移地址：0 其他：-1
变量名：b 类型：int 偏移地址：1 其他：-1
变量名：c 类型：int 偏移地址：2 其他：-1
变量名：d 类型：int 偏移地址：3 其他：-1
变量名：e 类型：int 偏移地址：4 其他：-1


函数表：


活动记录：main
```

```
DATAS SEGMENT
  a DB ?
  b DB ?
  c DB ?
  d DB ?
  e DB ?
DATAS ENDS
STACKS SEGMENT
  STK DB 20 DUP (0)
STACKS ENDS
CODES SEGMENT
  ASSUME CS:CODES,DS:DATAS,SS:STACKS
START:
  MOV AX,DATAS
  MOV DS,AX
  MOV SP,BP
  MOV AL,DS:[1]
  MOV AH,OFFSET 8D
  MUL AH
  MOV SS:[BP+1],AL
  MOV AL,OFFSET 10D
  ADD AL,SS:[BP+1]
  MOV SS:[BP+2],AL
  MOV AL,SS:[BP+2]
  SUB AL,OFFSET 7D
  MOV SS:[BP+3],AL
  MOV AL,OFFSET 10D
  SUB AL,DS:[2]
```

```
    MOV SS:[BP+3],AL
    MOV AL,OFFSET 10D
    SUB AL,DS:[2]
    MOV SS:[BP+4],AL
    MOV AL,SS:[BP+4]
    MOV AH,DS:[3]
    MUL AH
    MOV SS:[BP+5],AL
    MOV AL,SS:[BP+3]
    ADD AL,SS:[BP+5]
    MOV SS:[BP+6],AL
    MOV AL,SS:[BP+6]
    ADD AL,SS:[BP+1]
    MOV SS:[BP+8],AL
    MOV AL,SS:[BP+7]
    MOV AH,OFFSET 6D
    MUL AH
    MOV SS:[BP+9],AL
    MOV AL,SS:[BP+8]
    SUB AL,SS:[BP+9]
    MOV SS:[BP+10],AL
    MOV AL,SS:[BP+10]
    MOV AH,OFFSET 4D
    MUL AH
    MOV SS:[BP+11],AL
    MOV AL,SS:[BP+11]
    SUB AL,DS:[4]
    MOV AH,4CH
    INT 21H
CODES ENDS
    END START
```

3.10 中间代码优化

3.10.1 功能

1. 删去无效赋值
2. 替换常数
3. 相同四元式替换

3.10.2 数据结构

List<String[]> qt ： 存放四元式

List<String> replace ：存放需要替换的东西，如要把 t1 换成 6.28，则在 replace 中存入 t1 和 6.28。replace 的偶数位是要被替换的，奇数位是前一位应该被替换成为的。

3.10.3 算法

1. 删去无效赋值： 如果某个四元式的结果在后续被重新赋值过并且没被引用过，那么它是无效赋值可以删掉。实现方法：对每个四元式查看它在后序的四元式里是否有被引用过（flag_appear12），和是否被重新赋值过（flag_appear3），如果未被引用过且被重新赋值过，则删去无效赋值
2. 替换常数： 如果某四元式第二位第三位都是数字且第一位是加减乘除，则在替换列表加入类似于 t1 6.28，代表等会把所有 t1 换成 6.28。并且判断 replace 是否为空，也就是检查当前四元式是否要执行把 t1 替换成 6.28 这种操作。
3. 相同四元式替换： 如果 t2 和 t4 的符号、操作数、目标操作数都一样，那么 t4=t2，t4 那条四元式删除

   总体的，如果执行 1/2/3 后都没有可以替换的东西，才当做优化完毕，否则从头重新执行一遍 1/2/3.之所以这么做，是因为后序的优化可能使得前面可以再次优化。

```
原始四元式:
* b 8.0 t1
+ 10.0 t1 t2
- t2 7.0 t3
- 10.0 c t4
* t4 d t5
+ t3 t5 t6
* b 8.0 t7
+ t6 t7 t8
- 10.0 c t9
* t9 6.0 t10
- t8 t10 t11
* t11 4.0 t12
- t12 e a

优化后的四元式:
* b 8.0 t1
+ 10.0 t1 t2
- t2 7.0 t3
- 10.0 c t4
* t4 d t5
+ t3 t5 t6
+ t6 t1 t8
* t9 6.0 t10
- t8 t10 t11
* t11 4.0 t12
- t12 e a
```

3.11 目标代码生成

3.11.1 功能

输入优化后的四元式，输出 8086 汇编语言目标代码

3.11.2 数据结构

List<String> code ： 存放目标代码

boolean[][] active ： 存与四元式 qt 一一对应的活跃信息

HashSet<String> t ： 一个集合，为了把每个非数字的变量名加入 set 去重，获取变量表

List<String> variable ： set 转化为 list 的变量表

boolean[] symbl ： 存放变量对应的活跃信息

String RDL ： 表示寄存器此时存的是哪个变量。我们用的是单寄存器。

String[] kt ： 四元式中的关键字，为了区分变量

内容为：

{ "if","es","ie","wh","dw","bk","ct","we","for","df","do","fe","fun","rt","xc","esf","sw","dft","cs"};

Stack<String> whn ： 保存 while 该怎么跳，遇到 while 新建 WH 就 push，遇到 WE 就 pop

Stack<String> wen ： 遇到 CMP 新建 WE 就 push，遇到 WE 就 pop

Stack<String> esn ： 遇到 if 新建 ES 就 push，遇到 ie 就 pop

Stack<String> ien ： 遇到 es 新建 IE 就 push，遇到 ie 就 pop

Stack<String> jmpfn ：为了避免汇编程序运行到函数时不跳过而引起死循环，需要在函数前加一句无条件跳转到函数后的语句。

3.11.3 算法

1. 求活跃信息：给变量的活跃信息赋初值，临时变量初值 false，非临时变量初值 true。再逆序扫描四元式组，添加变量在每个四元式的时候的活跃信息。
2. 根据符号表生成程序前部：定义数据段堆栈段等。需要变化的只有根据符号表总表的全局变量来生成数据段的部分，其他地方都是写死的。

3. 根据四元式生成程序中部：

算数表达式：

+ - * / ——> MOV AL,t1　　ADD AL,t2

比较语句：

> < == >= <= != ——> MOV AL,t1　　MOV AH,t2　　CMP AL,AH

如果下一句是 dw 或 df，则根据比较符号取反 J? WEN(新建)或 FEN(新建)

分支结构：

if ——> 如果这个 if 有 else 则根据上一句的比较符号取反 J? ESN(新建)

没有 else 则根据上一句的比较符号取反 J? IEN(新建)

es ——> JMP IEN(新建)　　ESN:

ie ——> IEN:

循环结构：

wh ——> MOV CX,02H　　WHN(新建):　　INC CX

we ——> LOOP WHN　　WEN:

for ——> MOV CX,02H　　FORN:(新建)　　INC CX

fe ——> LOOP FORN　　FEN:

bk ——> JMP WEN 或 FEN

ct ——> 根据上一句的比较符号 J? WHN 或 FORN

子程序：

fun ——> fname PROC NEAR　　在活动记录中加入函数

rt ——> MOV AL,结果　　RET　　fname ENDP　　在活动记录中把该函数

及其临时变量全 pop 掉

sf ——> 向活动记录加入 f 以供传参寻找偏移地址

xc ——> MOV 形参的位置,想传的参数

esf ——> CALL fname　　活动记录 pop f

4. 添加上程序尾部 MOV AH,4CH　INT 21H　CODES ENDS　　END START

3.12 可视化界面

我们的可视化界面是用 html+css+js 前端和 django 后端执行命令调用 java 生成的可执行 jar 文件完成的。可通过 http://www.maocaoqiu.cn/compiler/访问。

图 1 源代码示例

图 2 中间代码生成


Compiler_NEU_2018
原代码
中间代码
目标代码
目标代码
DATAS SEGMENT
x1 DB ?
x2 DB ?
x3 DB ?
a DB ?
b DB ?
w1 DB ?
i DB ?

图 3 目标代码生成

4. 程序设计与实现

4.1 程序流程图

图 1 整体流程

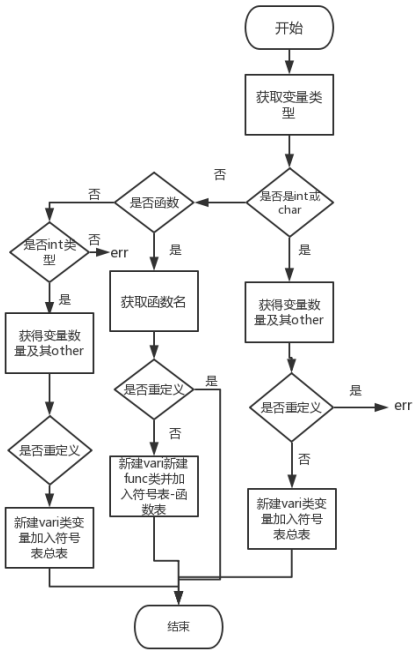


图 2 全局变量声明

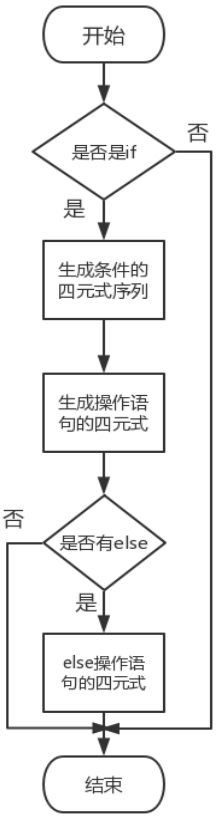


图 3 if 语句流程

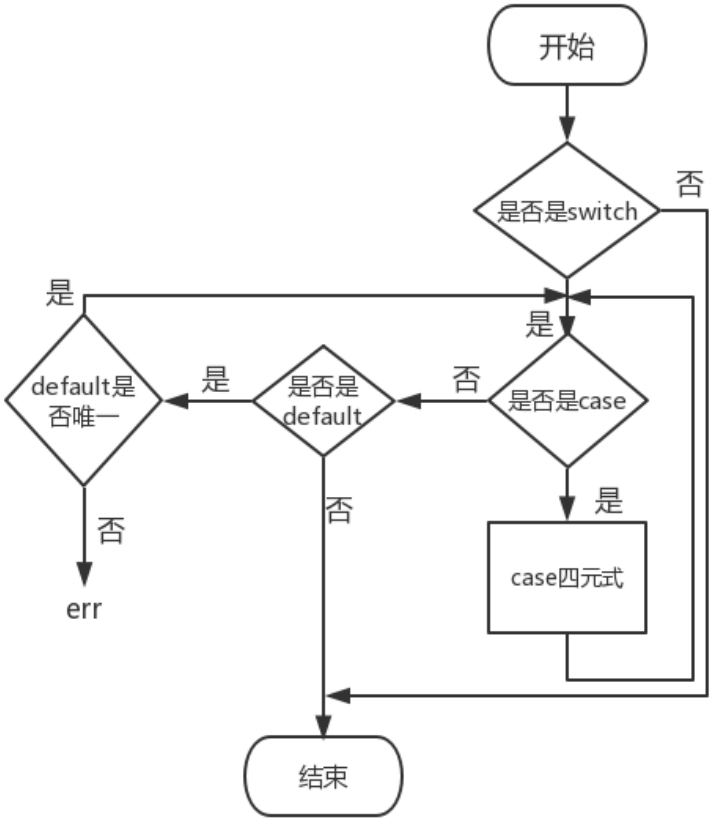


图 4　Switch 语句流程图

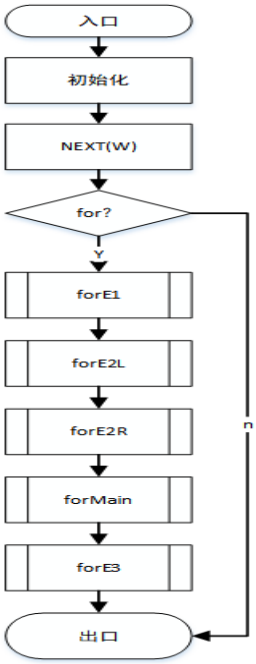


图 5　For 整体流程

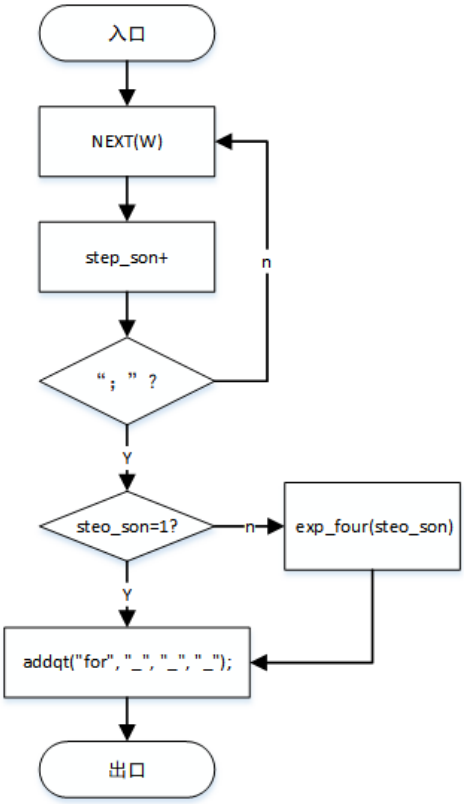


图 6　For 四元式

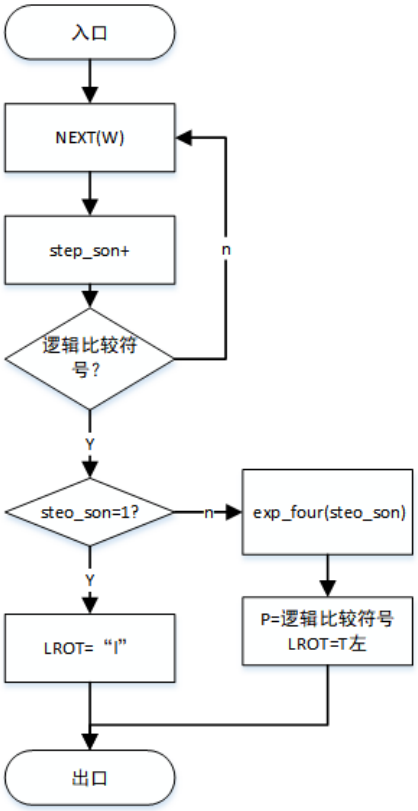


图 7　For 左四元式

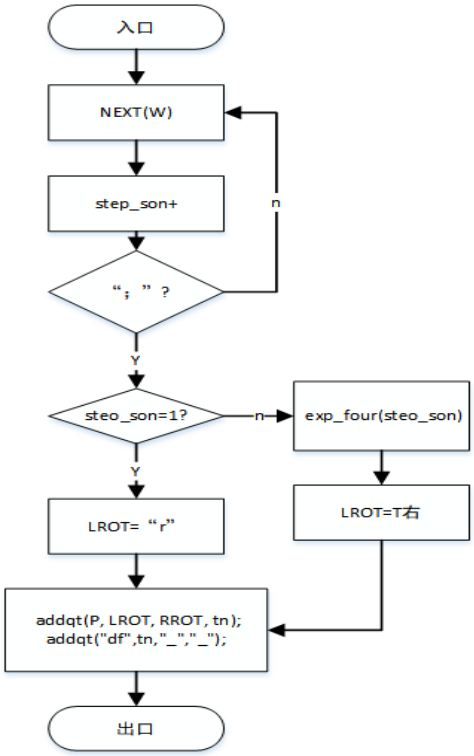


图 8　For 生成四元式

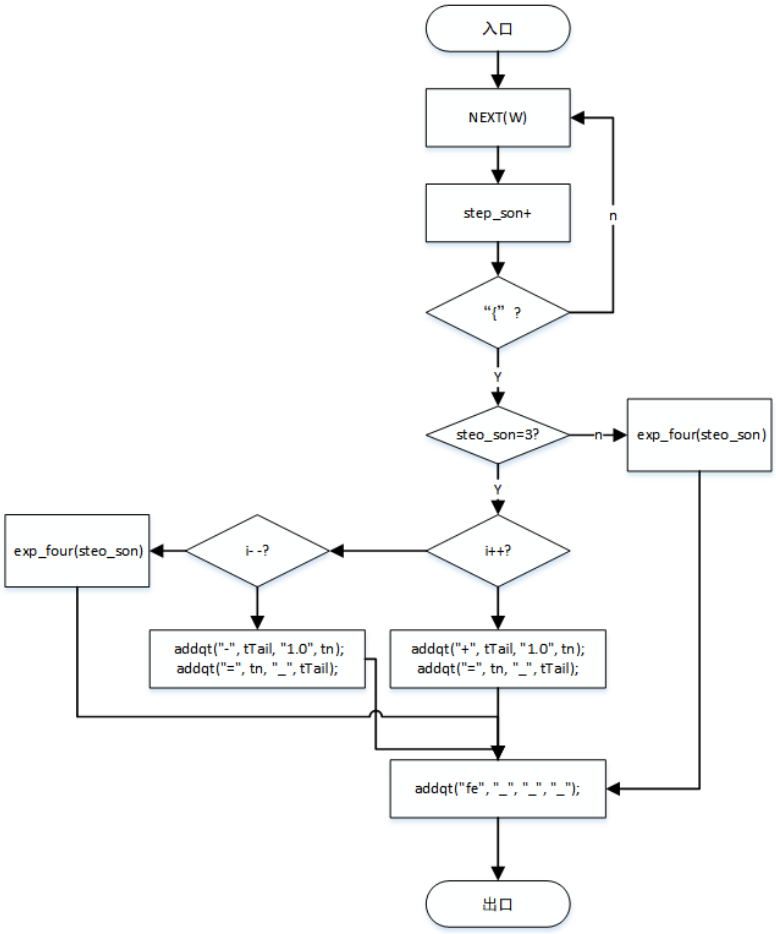


图 9　for 主体

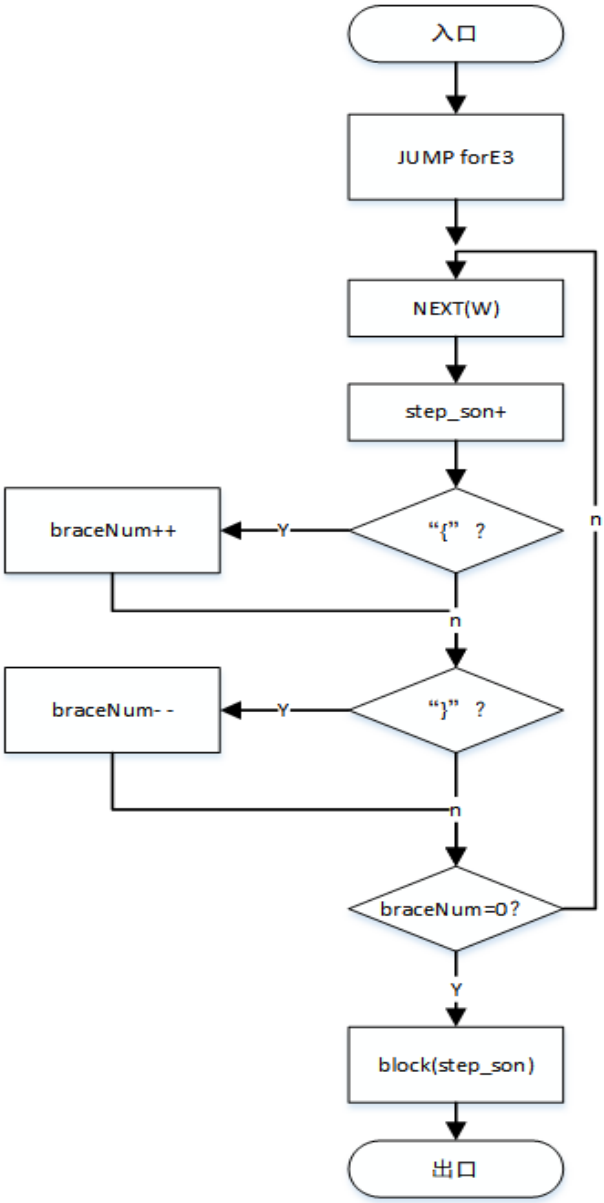

入口
JUMP forE3
NEXT(W)
step_son+
“{” ?
Y
braceNum++
n
“}” ?
Y
braceNum- -
n
braceNum=0?
n
Y
block(step_son)
出口

图 10　For 主体

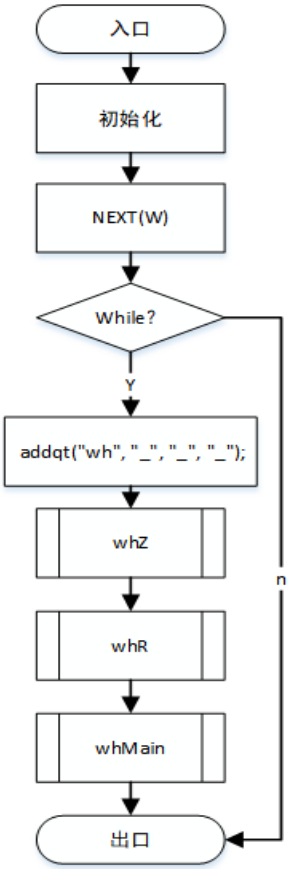


图 11　while 总体

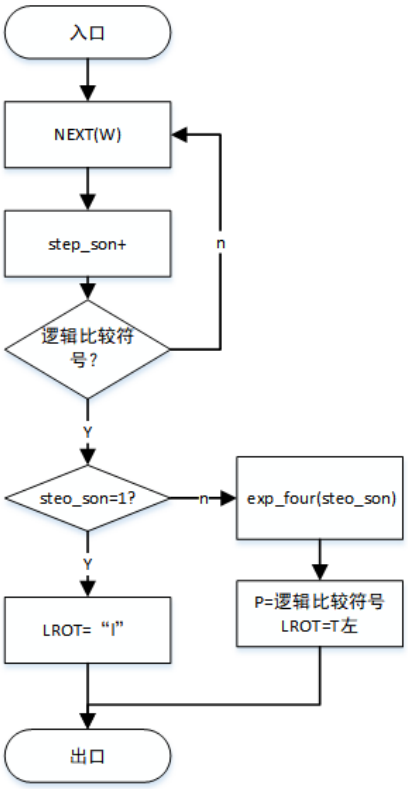


图 12　while 左四元式

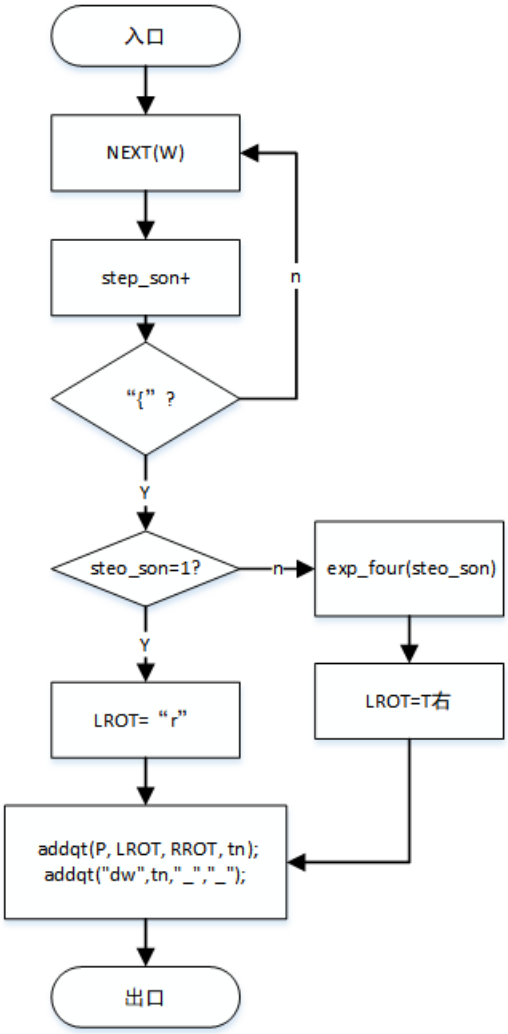


图 13　while 右四元式

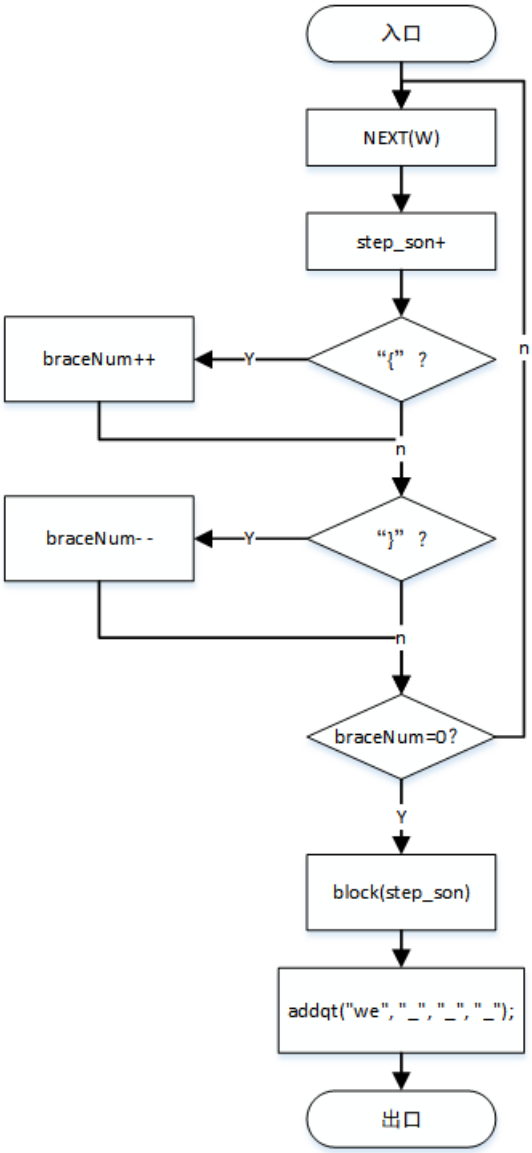

入口
NEXT(W)
step_son+
“{” ?
Y
braceNum++
n
n
“}” ?
Y
braceNum- -
n
braceNum=0?
n
Y
block(step_son)
addqt("we", "_", "_", "_");
出口

图 14　while 主体

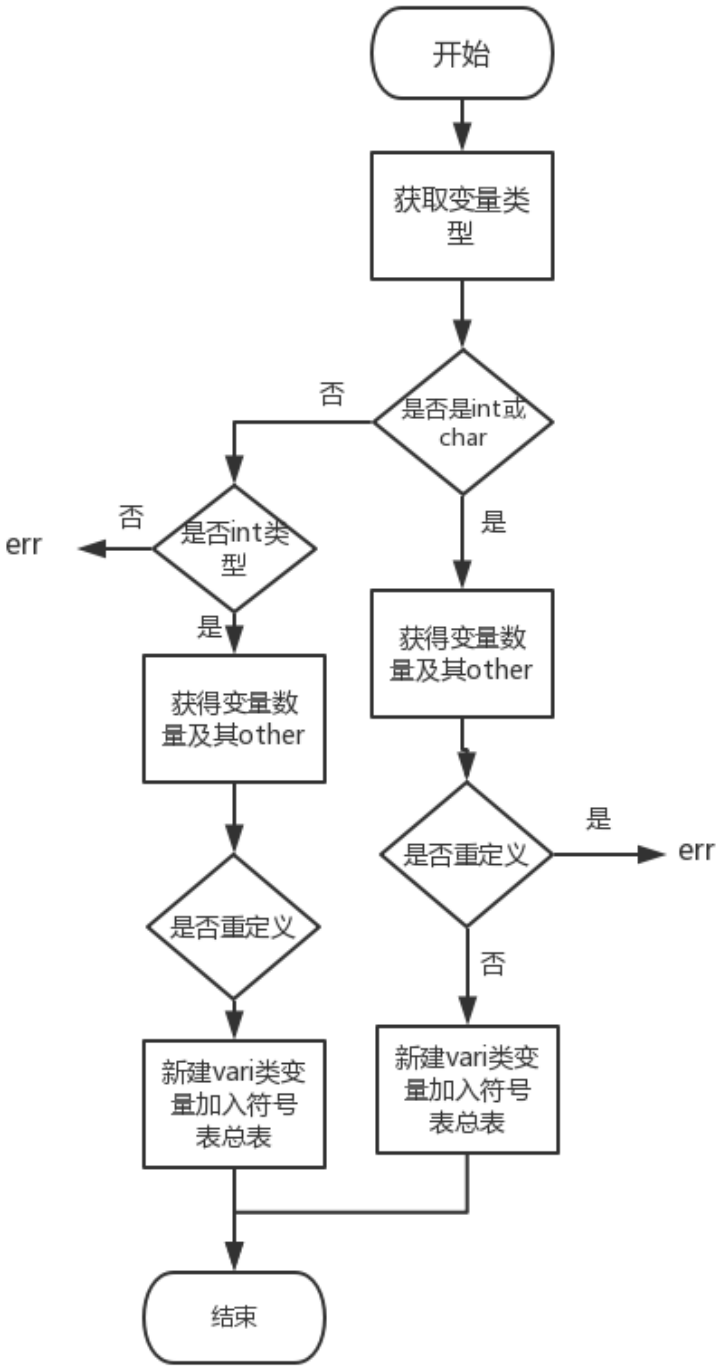

开始
获取变量类型
是否是int或char
否
是
否
是否int类型
err
是
获得变量数量及其other
获得变量数量及其other
是否重定义
是
err
否
是否重定义
新建vari类变量加入符号表总表
新建vari类变量加入符号表总表
结束

图 15　局部变量声明

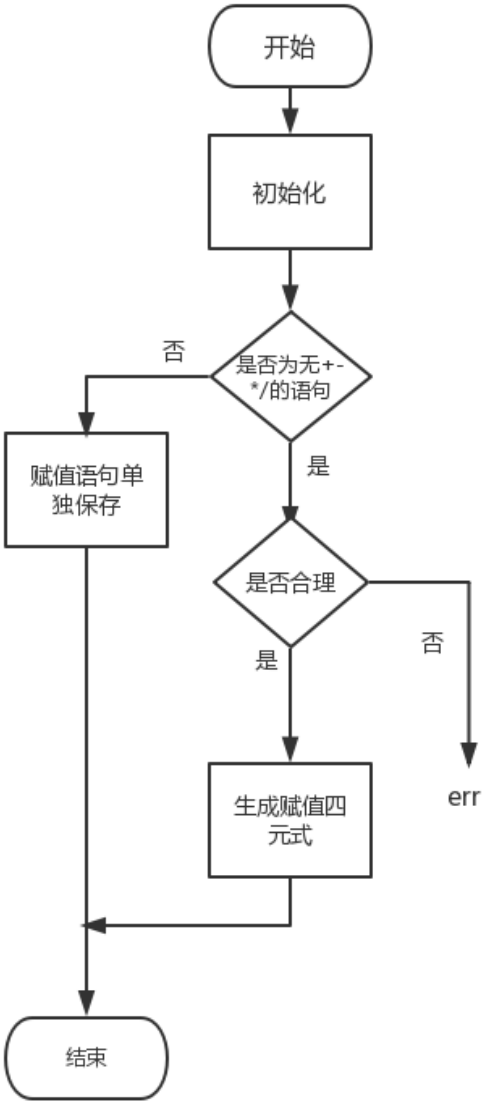


图 16　算数表达式

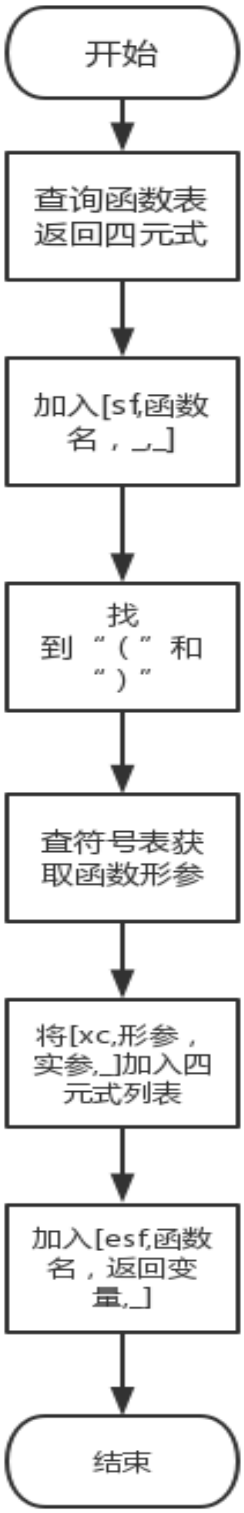

开始
查询函数表
返回四元式
加入[sf,函数
名，_,_]
找
到“(”和
“)”
查符号表获
取函数形参
将[xc,形参，
实参,_]加入四
元式列表
加入[esf,函数
名，返回变
量,_]
结束

图 17　函数调用

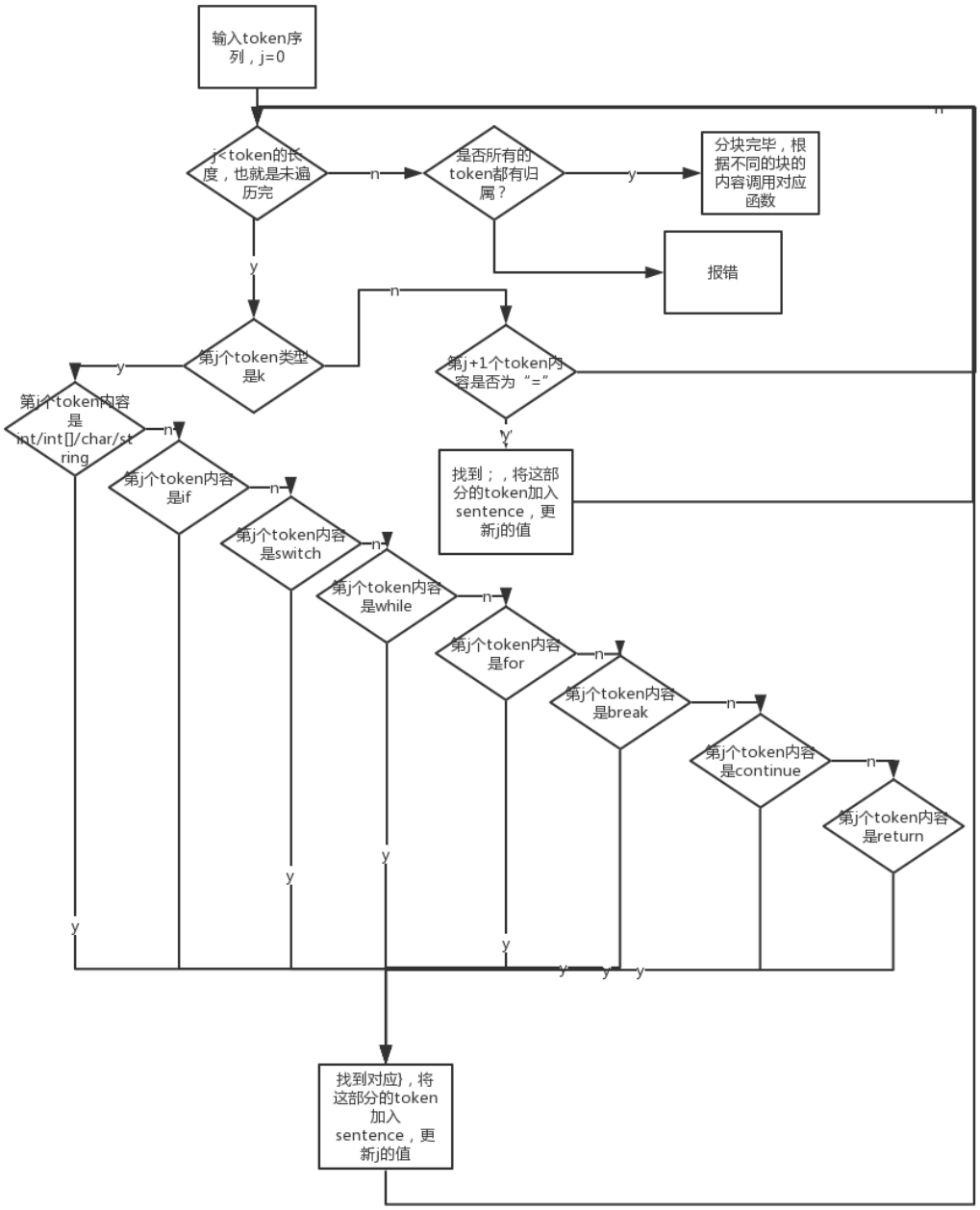


图 17　Block 流程图

图 18　入口函数 main

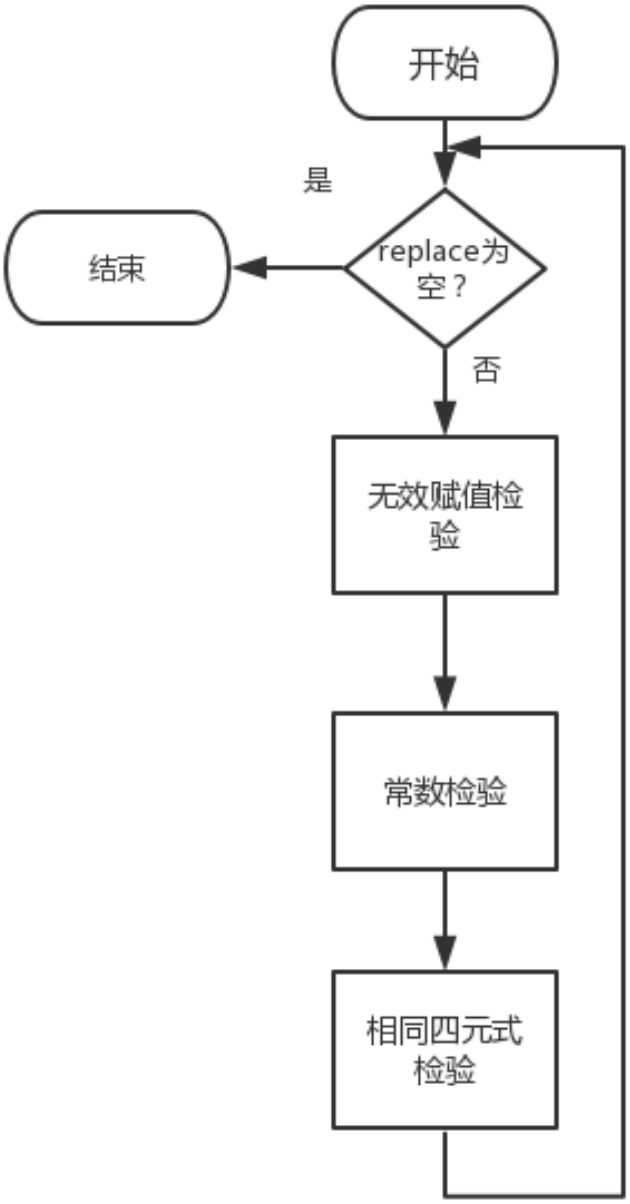


图 19　optimization 优化

4.2 程序说明

在程序的执行中，我们封装了如下函数:

1. boolean is_c(String c)判断单词是否为数字
2. boolean is_t(String c)判断单词是否为临时变量
3. boolean is_arr(String s)判断单词是否为数组
4. int reset_t(List<String[]> qtt, int n)　避免临时变量的冲突，在每次往 qt 加 qtt 的时候都要重置所有 t 后面的值。在所有的四元式生成模块中都要用到。
5. void addqt(String a1,String a2,String a3,String a4)四个字符串生成一条四元式并送入四元式区
6. void printLS(List<String[]> r)打印 List<String[]>，一般用来打印四元式组
7. String getf(table tb) 查符号表，获取正在执行的函数的函数名。在局部变量声明时用到。
8. int getofad(List<table.vari> vt) 在新建变量时，求新变量的偏移地址。在变量声明时用到。
9. String getTraceInfo() 输出代码行号的函数。在算数表达式模块使用递归下降法时 debug 用。
10. String getv(String v,table tb) 在目标代码生成时取源操作数使用，输入形如"a"这样的变量名，输出形如 DS:[1]或 SS:[BP+2]的地址。
11. String getv1(String v,table tb) 在目标代码生成时取目标操作数使用，输入形如"a"这样的变量名，输出形如[a]或 SS:[BP+2]的地址。
12. String cmpp(String cmp) 意思是 cpm_positive，将">"转化为"JA"，别的同理
13. String cmpn(String cmp) 意思是 cpm_negative，将">"转化为"JBE"，别的同理
14. int getfnum(table tb) 获取最新函数在活动记录中的位置
15. int getflocalnum(table tb) 获取最新函数的局部变量的数量
16. boolean hves(List<String[]> qt,int i) 判断四元式当前的 if 语句是否有 else
17. int index(List<String> variiable, String str) 找到 str 在 variiable 中的位置
18. void print(table tb) 打印符号表

4.3 实验结果

```
int main()
{
    int x1,x2,x3;
    x1=1;
    x2=2;
    x3=3;
    x1=x2;
    x2=x3;
    x3=x1;
    x1=x3+x2;

    int a;
    int b;
    int w1;
    b=10;
    a=1+2+3*4*5+b;
    int f(int c,int d)
    {
        int e;
        e=c+d;
        return e;
    }
    a=f(10,15);
```

```
    w1=1;
    while(w1<5)
    {
        a=b+2;
        w1=w1+1;
        if(w1>3)
        {
            a=a+200;
            break;
        }
        else
        {
            a=a+2;
        }
    }
    int i;
    for(i=0; i<=6; i++)
    {
        a=a+9;
    }
    int shuzu[20];
    shuzu[1]=20;
    shuzu[2]=20;
    shuzu[3]=20;
    shuzu[1]=shuzu[2]+shuzu[3];
}
```

图 1　源输入程序

```
优化后的四元式:
= 2.0 _ x2
= 3.0 _ x3
= x2 _ x1
= x3 _ x2
= x1 _ x3
+ x3 x2 x1
= 10.0 _ b
+ 63.0 b a
fun f _ _
+ c d e
rt e _ _
sf f _ a
xc c 10.0 _
xc d 15.0 _
esf f a _
= 1.0 _ w1
wh _ _ _
< w1 5.0 t5
dw t5 _ _
+ b 2.0 a
+ w1 1.0 w1
> w1 3.0 t6
if t6 _ _
+ a 200.0 a
bk _ _ _
es _ _ _
+ a 2.0 a
ie _ _ _
we _ _ _
= 0.0 _ i
for _ _ _
<= i 6.0 t7
df t7 _ _
+ a 9.0 a
+ i 1.0 i
fe _ _ _
= 20.0 _ shuzu[2]
= 20.0 _ shuzu[3]
+ shuzu[2] shuzu[3] shuzu[1]
```

图 2　优化四元式

```
原始四元式:
= 1.0 _ x1
= 2.0 _ x2
= 3.0 _ x3
= x2 _ x1
= x3 _ x2
= x1 _ x3
+ x3 x2 x1
= 10.0 _ b
+ 1.0 2.0 t1
* 3.0 4.0 t2
* t2 5.0 t3
+ t1 t3 t4
+ t4 b a
fun f _ _
+ c d e
rt e _ _
sf f _ a
xc c 10.0 _
xc d 15.0 _
esf f a _
= 1.0 _ w1
wh _ _ _
< w1 5.0 t5
dw t5 _ _
+ b 2.0 a
+ w1 1.0 w1
> w1 3.0 t6
if t6 _ _
+ a 200.0 a
bk _ _ _
es _ _ _
+ a 2.0 a
ie _ _ _
we _ _ _
= 0.0 _ i
for _ _ _
<= i 6.0 t7
df t7 _ _
+ a 9.0 a
+ i 1.0 t8
= t8 _ i
fe _ _ _
= 20.0 _ shuzu[1]
= 20.0 _ shuzu[2]
= 20.0 _ shuzu[3]
+ shuzu[2] shuzu[3] shuzu[1]
```

图 3　原始四元式

```
总表:
变量名: x1 类型: int 偏移地址: 0 其他: -1
变量名: x2 类型: int 偏移地址: 1 其他: -1
变量名: x3 类型: int 偏移地址: 2 其他: -1
变量名: a 类型: int 偏移地址: 3 其他: -1
变量名: b 类型: int 偏移地址: 4 其他: -1
变量名: w1 类型: int 偏移地址: 5 其他: -1
变量名: f 类型: function 偏移地址: 0 其他: 0
变量名: i 类型: int 偏移地址: 6 其他: -1
变量名: shuzu 类型: int[] 偏移地址: 7 其他: 20


函数表:

函数名: f
形参:
int c
int d
函数的局部变量:
变量名: c 类型: int 偏移地址: 0 其他: 0
变量名: d 类型: int 偏移地址: 1 其他: 0
变量名: e 类型: int 偏移地址: 2 其他: -1


活动记录: main
```

图 4　活动记录

```
DATAS SEGMENT
  x1 DB ?
  x2 DB ?
  x3 DB ?
  a DB ?
  b DB ?
  w1 DB ?
  i DB ?
  shuzu DB ?
DATAS ENDS
STACKS SEGMENT
  STK DB 20 DUP (0)
STACKS ENDS
CODES SEGMENT
  ASSUME CS:CODES,DS:DATAS,SS:STACKS
START:
  MOV AX,DATAS
  MOV DS,AX
  MOV SP,BP
  MOV AL,OFFSET 2D
  MOV [x2],AL
  MOV AL,OFFSET 3D
  MOV [x3],AL
  MOV AL,DS:[1]
  MOV [x1],AL
  MOV AL,DS:[2]
  MOV [x2],AL
  MOV AL,DS:[0]
  MOV [x3],AL
  MOV AL,DS:[2]
  ADD AL,DS:[1]
```

```
  MOV [x1],AL
  MOV AL,OFFSET 10D
  MOV [b],AL
  MOV AL,OFFSET 63D
  ADD AL,DS:[4]
  MOV [a],AL
  JMP JMPF0
f PROC NEAR
  MOV AL,SS:[BP]
  ADD AL,SS:[BP+1]
  MOV SS:[BP+2],AL
  MOV AL,SS:[BP+2]
  RET
f ENDP
JMPF0:
  MOV AL,OFFSET 10D
  MOV SS:[BP],AL
  MOV AL,OFFSET 15D
  MOV SS:[BP+1],AL
  CALL f
  MOV [a],AL
  MOV AL,OFFSET 1D
  MOV [w1],AL
  MOV CX,02H
WH1:
  INC CX
  MOV AL,DS:[5]
  MOV AH,OFFSET 5D
  CMP AL,AH
  JAE WE2
  MOV AL,DS:[4]
```

```
  ADD AL,OFFSET 2D
  MOV [w1],AL
  MOV [w1],AL
  MOV AL,DS:[5]
  MOV AH,OFFSET 3D
  CMP AL,AH
  JBE ES3
  MOV AL,DS:[3]
  ADD AL,OFFSET 200D
  MOV [a],AL
  JMP WE2
  JMP IE4
ES3:
  MOV AL,DS:[3]
  ADD AL,OFFSET 2D
  MOV [a],AL
IE4:
  LOOP WH1
WE2:
  MOV AL,OFFSET 0D
  MOV [i],AL
  MOV CX,02H
FOR5:
  INC CX
  MOV AL,DS:[6]
  MOV AH,OFFSET 6D
  CMP AL,AH
  JA FE6
  MOV AL,DS:[3]
  ADD AL,OFFSET 9D
  MOV [a],AL
```

```
  MOV AL,DS:[6]
  ADD AL,OFFSET 1D
  MOV [i],AL
  LOOP FOR5
FE6:
  MOV AL,OFFSET 20D
  MOV [shuzu+2],AL
  MOV AL,OFFSET 20D
  MOV [shuzu+3],AL
  MOV AL,DS:[9]
  ADD AL,DS:[10]
  MOV AH,4CH
  INT 21H
CODES ENDS
  END START
```

图 5　生成汇编代码

5. 结论

我们这次设计并实现了一种基于递归下降法用以将类 C 语言翻译为 8086 汇编语言的简单编译器。它拥有与用户交互的良好界面，所设计的文法支持整数类型及数组的定义和运算，函数的定义与调用，if、switch 分支语句，while、for 循环语句等功能，并允许各模块之间相互嵌套使用。在语义分析过程中填写了符号表，生成四元式，经过优化处理生成优化后的四元式，最终通过代码生成模块生成简易的可执行的汇编程序。总体来说是一个较为完整的编译器，达到了我们最初的目标。

6. 参考文献

1、陈火旺. *《程序设计语言编译原理》（第 3 版）*. 北京：国防工业出版社. 2000.

2、美 Alfred V. Aho Ravi Sethi Jeffrey D. Ullman 著. 李建中，姜守旭译. *《编译原理》*. 北京：机械工业出版社. 2003.

3、美 Kenneth C.Louden 著. 冯博琴等译. *《编译原理及实践》*. 北京：机械工业出版社. 2002.

4、金成植著. *《编译程序构造原理和实现技术》*. 北京：高等教育出版社. 2002.

7. 收获、体会和建议

组长 马莹莹：

为期两周的编译原理课设结束了，我有收获，也有遗憾。

超过 5000 行的代码量，是迄今为止量最多难度也最大的课程设计了。

在一个月前，写实验的时候，我还对编译原理近乎一窍不通。

作为一个不擅长理论学习而更喜欢通过实践学习的人，长达半年的编译原理课程学习并没有使我对这门课产生足够的了解。虽然编译原理课程组的老师们是我大学专业课里遇到的最好的、最负责的、最有魅力的老师，但听课学习对于我来说仍然效率欠佳。不过，在经历了课程设计以后，我觉得编译原理应该是我大学里学的最明白的专业课了。

不得不提，实验对课程设计的帮助非常非常大。如果我没有在实验中把前端的词法分析器和四种语法分析器都尝试了一遍，课设的难度将会是现在的数倍，那种情况下我会很难做到现在这样的完成度。实验阶段编译器前端部分的自学效果极佳，主要感谢编译原理课程组老师们精心制作的 ppt，非常的清晰易懂。在过去我常常在网上找 CSDN 或者别的地方的博客来学习某些专业知识，而在编译器前端的学习中，ppt 是我能找到的最易懂的材料，比网上的博客都好得多。

ppt 中唯一不太好懂的是符号表那章。虽然最主要的原因是符号表本身就是最难最琐碎的部分，但 Pascal 举例同样的让我们不那么容易看懂的理由之一。虽然 Pascal 语法简单，但它仍然是一门我们不熟悉的语言，难以让我们将 ppt 上的内容与过去已知的知识产生联系。

如果说课设里让我选出一个最难的点，那么我一定会说是符号表的理解。写符号表花了我三天多的时间，没有任何别的部分能卡住我这么久。而这三天多，

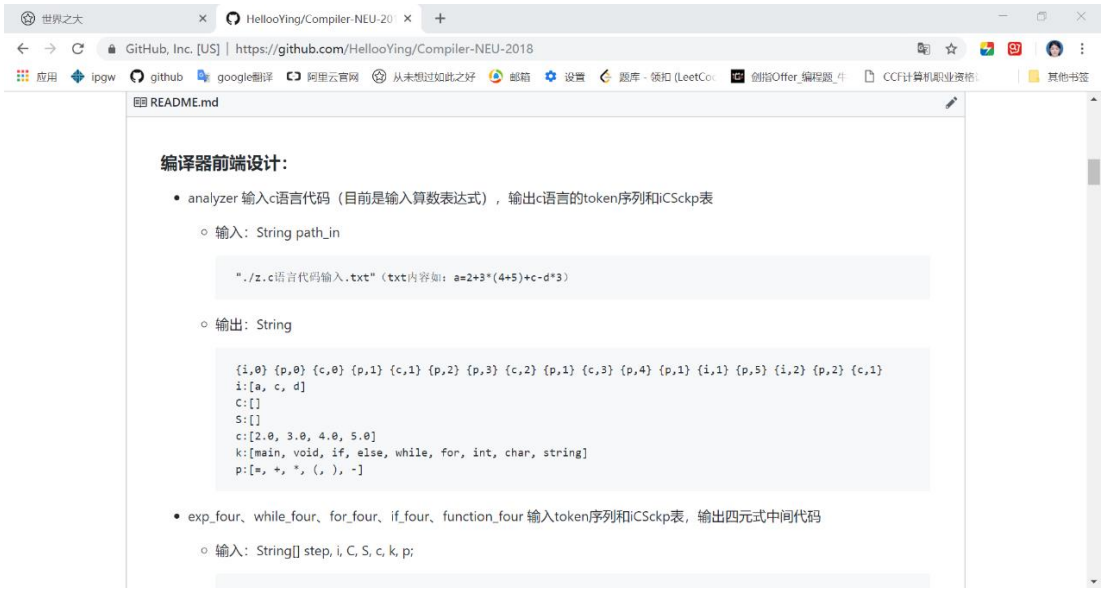
主要就花在理解“符号表到底是干嘛的”、“它为何而存在”上，其次是设计符号表的类结构，真正用来编程实现它的时间反而很短。

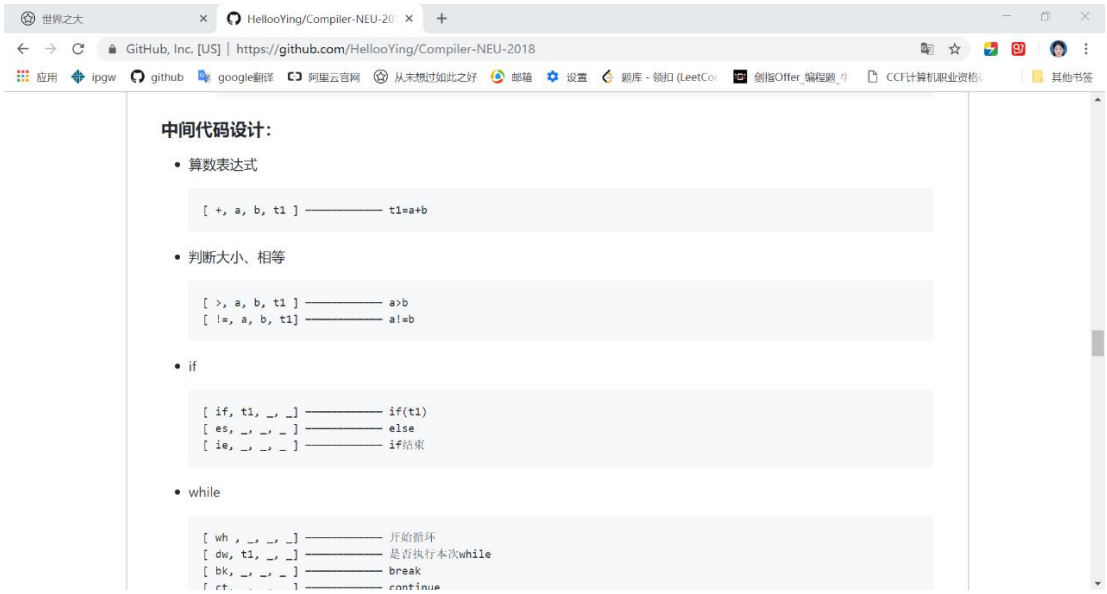
在课设的最开始，我花了几天时间实现了基本块与算数表达式的从词法扫描器到目标代码生成。因为 ppt 做的太好了，几乎没遇到什么坎。我以为接下来就是帮助队友一起把循环结构和分支结构加到现有的框架里来，却没想到这时给自己留下了一个坑——ppt 里的汇编是伪代码，而非我们想要的可执行 8086 汇编。在发现这一点后，我花了 2 天左右去复习汇编语言，去调通我们需要的语句。这里要感谢一下我的同学们，8086 汇编的资料在网上不太好找，汇编课本上的内容也不太够用，尤其是对数据段和堆栈段的读写操作，我找了好久也试了挺多次都没找到正确操作，最后问了同样写 8086 的朋友才解决。为了记录我们应该生成的汇编代码是什么样的，我把目标代码说明详细的写在了文档里：

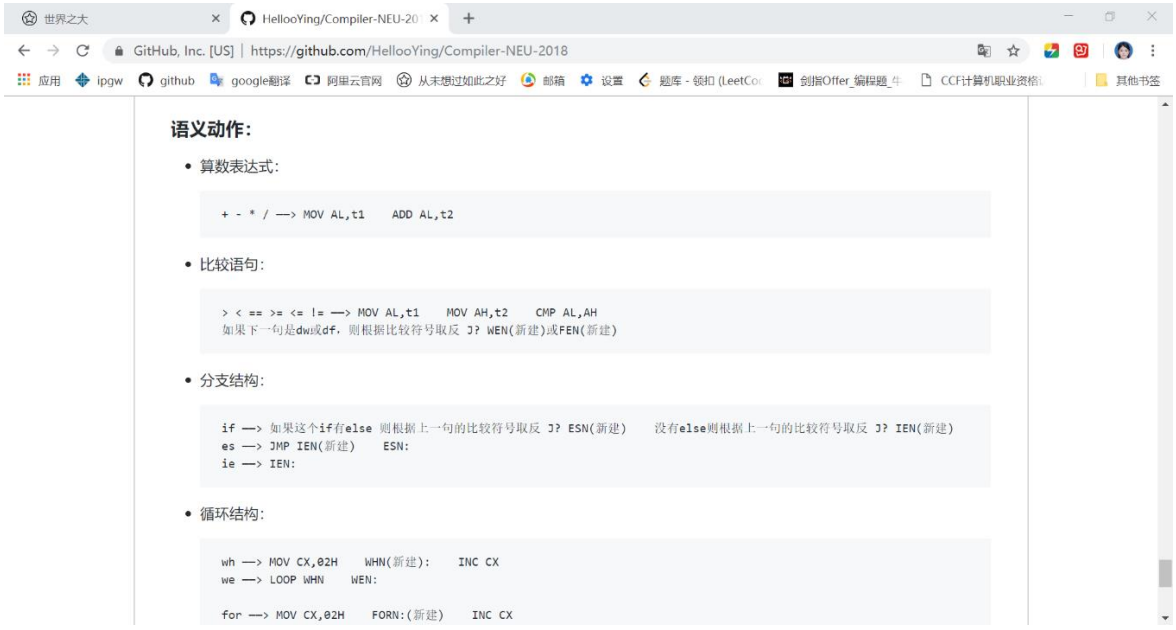
汇编语言不熟练的困扰其实是与对符号表的不解并行。我看 ppt、看书、跟同学讨论，才明白符号表在我们简单 c 语言编译器中最重要的意义是与变量声明和变量存储的偏移地址相关的。当然，重定义判别等别的功能也很重要。于是在课设中某一天的夜晚，我着手去实现符号表的构建。我大概有一个不太好的编程习惯，那就是常常使用 List 或数组来存放一切东西而不是写一个类或者结构体。但那天晚上，符号表的内容太复杂了，用 List 和数组实在是不够用了，我才终于想到去构建一个类来实现。或许在这里我又弥补了一些之前欠下的基础：更深刻的理解了何为面向对象，何为抽象化具体化。因此我在课设中最得意的地方也

是设计并实现了符号表结构和符号表在各处如何使用。符号表这部分，老师讲的也比较少，不能照着做，只能一步步摸索。但是我自己设计、实现、发现不合理而重构等等一套下来，最后留下的自己设计的符号表竟与听说正确的符号表该是怎样的没有太大差别。合理的设计大概是相似的，这使我受益匪浅。

我完成了最开始想完成的所有：完整的编译器前后端、目标代码可以在 masm 中运行而不出错、支持数组、支持函数、循环结构与分支结构都各写了两个语句（for 和 while、if 和 switch）、简洁明了的可视化界面、完善的文档说明，还有尽量让两个队友从前端到后端全面接触一遍来促进他们对这门课程的理解。







图：简洁有效的文档说明

但最后却发生了让我感到有些遗憾的事： 我没有想到结构体会成为验收标准中的一项，这或许也是我长期忽略使用类与结构体造成的。而且课设进行时我们也没被告知结构体应该完成。如果早知道，我肯定不会放着它不管。还有一个遗憾是，我们没有想到应该在写代码的同时写报告。最后只有 2、3 天时间写报告，还穿插着各种别的课的实验，我负责的模块较多，时间上不足以让我对每个模块写出详细的报告了，甚至流程图也找队友帮着画的。虽然知道老师也是想让我们有更多时间复习，但还是希望时间再充裕一些，我们也能把报告写的更完善一些。

虽然有着遗憾，我仍然觉得收获许多。我在过去的时间里一直使用 python 语言做项目，但明年春招却想找 Java 的实习，编译课设为我解了“没有用 Java 做过项目”的燃眉之急。5000 行的项目说大不大，说小不小，又偏重基础，是一个适合放到简历上的好项目。除此之外，这次课设也使我感受到了我长期以来刷算法题的效果：写起工程确实比过去更加流畅通顺、bug 也更少了。但是想特殊情况、想边界的能力确实仍需提高，虽然我已经能想到大部分意外情况并规避，但仍有时候出现意料之外的 bug，需要花较长时间去定位测试解决。再除此之外，作为一个有着“毕业前码十万行代码”志向的大学生，编译课设贡献的 5000 行也让我离目标更近了一步。

组员 赵祉怡：

这次的课程设计我主要负责文法、循环结构的中间代码四元式及目标代码的生成和后期的测试工作。

对于此次任务的分工我觉得我组较为合理，我组并没有如许多组般按照编译程序的五大步骤拆分成任务，而是对希望实现的类 C 源代码的各个功能模块进行拆分。 这样做的好处是可以使每位成员都对整个编译程序过程中的前后端均有学习、设计和实现，大大提高我们对编译器的整体理解和认知。

对于循环语句四元式生成的实现，当自己的代码运行时并为出现太大问题，但在之后与其它组员的代码进行合并之后就暴露出了许多问题，尤其是在和分支结构及循环结构自身嵌套使用及加入跳出指令之后在生成的 8086 汇编代码执行时会出现死循环的问题。例如 for1 表达式我一开始放在循环内部，而这样的话若是 for1 表达式为初始化赋值操作，那么每次循环后其值都会被重新赋为初值，而循环内部对变量的操作将毫无意义，从而造成死循环。诸如此类的错误还有很多，对此需要的是细心和耐心，仔细查看生成的汇编代码并对其分步调试 debug，观察每一步寄存器中值的变化和下一条指令的变化情况，从中找出编译程序逻辑上的错误然后对自己的程序进行修改。

本次实验使我对编译原理的理论知识有了实践的机会，加深了我对编译器各模块实现的理解和掌握程度，相信我会在期末的编译考试中取得好成绩。另外这次实验使用本学期新学的 Java 语言编写，也促进了我对 Java 的学习，对我之后的 Java 大作业也很有帮助。

组员 张馨月：

由于这是第一次用 java 语言写程序，对语言的类库和特性掌握能力并不是很好，写程序是十分的吃力，让我感觉真的是无从下手。于是我先看了看组长写好的词法分析等个别程序，慢慢理解熟悉语言的应用，整理好自己的思路，初步设计好分支结构的递归下降子程序，准备工作花费了不少时间，但是是很有意义的。最先写的是 if 语句，由于一直考虑的是 if-else，所以就忘记考虑没有 else 的情况；还有 if 语句的条件如果是复杂的算数表达式，则调用算数表达式的方

法生成四元式，当时也是没有多想，就直接写了，所以就忘记考虑如果是简单的比较，即没有算数表达式时，就不能调用这个方法，这些问题是在最后测试的时候发现的，发现这些问题时还是很郁闷的，毕竟要是再仔细一点就会想到的，不过还好问题不是很严重，在原有的基础上稍微修改加一些条件就能满足了。

吸取了 if 语句的教训，在设计 switch 语句的程序时，就比较细心，考虑了很多方面,比如 default 和 case 顺序是不固定,case 后面可能没有 break 等等。通　过比较仔细的分析，改完 bug 后测试的时候基本没有发现问题

通过这次课设，加深了我对编译原理这门课的理解，同时也让我对 java 有了更深的了解，通过不断的改 bug，修改程序，最终完成该功能，让我感到很开心，这几天大家一起改进程序，一起分析不足，收获很多。