## 详细设计

### 1．程序结构

【从总体上描述程序的结构，文字或图示均可】

读输入文件→语法分析（词法分析、登录符号表、跳过错误代码并报错、生成中间代码（四元式））→优化中间代码→生成目标代码（MIPS 汇编）并输出到文件

### 2．类/方法/函数功能

【描述各类/方法或函数的功能，以及关键算法】

词法分析函数：读下一个字符，判断其类型（数字、标识符、保留字、运算符、标点符号、括号、空白符）。其中，如果读到数字还要通过一个计算函数来计算数值。

语法分析函数：将读到的字符按照所给文法进行分析，跳过错误代码并报错，并生成中间代码；包括：跳读函数，处理说明（常量、变量、函数）的函数，处理文法中各成分（包括字符串、参数表、表达式、项、因子、赋值语句、条件语句、循环语句、情况语句、读语句、写语句和返回语句）的函数。

中间代码（四元式）生成函数：根据语法分析函数中所传参数生成中间代码。

目标代码（MIPS汇编）生成函数：根据中间代码生成目标代码。

优化处理函数：优化中间代码。

错误处理函数：根据产生错误打印错误信息。

登录符号表函数：把各变量、常量和函数的名字及具体信息按照数据结构登录到符号表中；数组登录到数组表中，函数登录到函数表中。

Main函数。

### 3．调用依赖关系

【说明各类之间的关系，方法/函数之间的调用关系】

语法分析函数要调用其内部的函数，以及词法分析函数、中间代码生成函数和错误处理函数；

语法分析函数调用处理说明函数，处理说明函数调用登录符号表函数；

Main函数调用语法分析函数、优化处理函数和目标代码生成函数。

### 4．符号表管理方案

【说明符号表的数据结构、管理算法】

符号表包括：全局符号表、函数表、数组表

符号表的数据结构：

全局符号表：符号名，种类（常量、变量、函数），类型（整型、实型、字符型、数组），值，地址，参数个数

函数表：记录函数内部的变量在全局符号表里的范围

数组表：元素类型，数组上下界，数组大小

管理算法：读入标识符时，遍历符号表查看是否已登录在符号表中，如果没有则登录该标识符

### 5．存储分配方案

【说明运行时的存储组织及管理方案，运行栈结构】

采用动态存储分配。运行栈要对不同类型的数据进行存取和操作。

运行栈结构：

栈顶

全局变量区

Main函数变量区

参数区（函数的运算结果，返回地址，局部变量区，在符号表中登记项的位置）

### 6. 解释执行程序\*

【说明解释执行程序的数据结构，关键算法，及解释执行过程】

不需要解释执行程序。

### 7. 四元式设计\*

【对采用的四元式进行详细说明】

原则上按照中缀表达式格式输出中间代码，即，形如x = y op z，其中x为结果，y为左操作数，z为右操作数，op为操作符。

* 1. 函数声明

源码形如：

int foo( int a, int b, int c, int d)

中间代码：

int foo()

para int a

para int b

para int c

para int d

* 1. 函数调用

源码形如：

i = tar(x,y)

中间代码：

push x

push y

call tar

i = RET

* 1. 函数返回

源码形如：

return (x)

中间代码：

ret x

* 1. 变量声明

源码形如：

int i, j;

中间代码（符号表信息输出，程序中可不生成真正的中间代码）：

var int i

var int j

* 1. 常数声明

源码形如：

const int c = 10

中间代码（符号表信息输出，程序中可不生成真正的中间代码）：

const int c = 10

* 1. 表达式

源码形如：

x = a \* (b + c)

中间代码（可优化）：

t1 = b + c

t2 = a \* t1

x = t2

* 1. 条件判断

源码形如：

x == y

中间代码：

x == y

* 1. 条件或无条件跳转

中间代码：

GOTO LABEL1 //无条件跳转到LABEL1

BNZ LABEL1 //满足条件跳转到LABEL1

BZ LABEL1 //不满足条件跳转到LABEL1

* 1. 带标号语句

源码形如：

Label\_1: x = a + b

中间代码：

Label\_1 :

x = a + b

* 1. 数组赋值或取值

源码形如：

a[i] = b \* c[j]

中间代码：

t1 = c[j]

t2 = b \* t1

a[i] = t2

### 8. 目标代码生成方案\*

【说明代码生成有关的数据结构、关键算法】

目标代码选择32位MIPS指令集。目标代码生成程序的输入为中间代码生成的DAG图，利用DAG图节点生成的顺序将各个节点依次翻译为MIPS汇编代码。

### 9. 优化方案\*

【说明代码优化有关的数据结构、关键算法】

1. DAG消除局部公共子表达式
2. 常数合并（窥孔优化）
3. 消除死代码
4. 删除冗余跳转代码
5. 引用计数和FIFO的寄存器分配策略

### 10. 出错处理

【说明出错处理方案、错误信息及含义】

|  |  |  |
| --- | --- | --- |
| 序号 | 错误信息 | 含义 |
| 0 | FILE NOT EXIST | 文件不存在 |
| 1 | ILLEGAL CHAR | 非法字符 |
| 2 | LACK SINGLE QUOTES | 缺少“’” |
| 3 | LACK DOUBLE QUOTES | 缺少“”” |
| 4 | LACK LEFT PARENT | 缺少“(” |
| 5 | LACK RIGHT PARENT | 缺少“)” |
| 6 | LACK LEFT BRACKET | 缺少“[” |
| 7 | LACK RIGHT BRACKET | 缺少“]” |
| 8 | LACK LEFT BRACE | 缺少“{” |
| 9 | LACK RIGHT BRACE | 缺少“:” |
| 10 | LACK SEMICOLON | 缺少“;” |
| 11 | LACK RETURN VALUE | 缺少返回值 |
| 12 | UNDEFINED IDENT | 该标识符未定义 |
| 13 | DUPLICATE DEFINE | 标识符重复定义 |
| 14 | UNDEFINED FUNCTION | 调用函数未定义 |
| 15 | WRONG PARA NUMBER | 函数参数个数错误 |
| 16 | WRONG PARA TYPE | 函数参数类型错误 |
| 17 | STACK OVERFLOW | 运行栈溢出 |
| 18 | ARRAY OVERFLOW | 数组溢出 |