《编译技术》课程设计文 档

学号：\_\_\_\_\_\_15061091\_\_\_\_\_\_

姓名：\_\_\_\_\_\_\_叶紫欣\_\_\_\_\_\_\_

2018年 1月 14 日

## 目 录

1. 需求说明 ----------------------------------------------------------------------------------------- 3
2. 文法说明 ------------------------------------------------------------------------------------- 3
3. 目标代码说明 ------------------------------------------------------------------------------- 4
4. 优化方案\* ------------------------------------------------------------------------------------ 6
5. 详细设计 ----------------------------------------------------------------------------------------- 6
6. 程序结构 ------------------------------------------------------------------------------------- 6
7. 类/方法/函数功能 **--------------------------------------------------------------------------** 6
8. 调用依赖关系 **-------------------------------------------------------------------------------** 6
9. 符号表管理方案 **----------------------------------------------------------------------------** 6
10. 存储分配方案 **-------------------------------------------------------------------------------** 7
11. 解释执行程序\* **------------------------------------------------------------------------------** 7
12. 四元式设计\* **--------------------------------------------------------------------------------- 7**
13. 目标代码生成方案\* **------------------------------------------------------------------------** 9
14. 优化方案\* **------------------------------------------------------------------------------------** 9
15. 出错处理 **-----------------------------------------------------------------------------------** 10
16. 操作说明
17. 运行环境 ------------------------------------------------------------------------------------ 11
18. 操作步骤 ------------------------------------------------------------------------------------ 11
19. 测试报告
20. 测试程序及测试结果 --------------------------------------------------------------------- 11
21. 测试结果分析 ------------------------------------------------------------------------------ 11
22. 总结感想 -----**-----------------------------------------------------------------------------------** 11

## 一．需求说明

### 1．文法说明

获取的文法为难度3的扩充C0文法：

＜加法运算符＞ ::= +｜-  
＜乘法运算符＞  ::= \*｜/  
＜关系运算符＞  ::=  <｜<=｜>｜>=｜!=｜==  
＜字母＞   ::= ＿｜a｜．．．｜z｜A｜．．．｜Z  
＜数字＞   ::= ０｜＜非零数字＞  
＜非零数字＞  ::= １｜．．．｜９  
＜字符＞    ::=  '＜加法运算符＞'｜'＜乘法运算符＞'｜'＜字母＞'｜'＜数字＞'  
＜字符串＞   ::=  "｛十进制编码为32,33,35-126的ASCII字符｝"  
＜程序＞    ::= ［＜常量说明＞］［＜变量说明＞］{＜有返回值函数定义＞|＜无返回值函数定义＞}＜主函数＞  
＜常量说明＞ ::=  const＜常量定义＞;{ const＜常量定义＞;}  
＜常量定义＞   ::=   int＜标识符＞＝＜整数＞{,＜标识符＞＝＜整数＞}  
                               | char＜标识符＞＝＜字符＞{,＜标识符＞＝＜字符＞}  
＜无符号整数＞  ::= ＜非零数字＞｛＜数字＞｝  
＜整数＞        ::= ［＋｜－］＜无符号整数＞｜０  
＜标识符＞    ::=  ＜字母＞｛＜字母＞｜＜数字＞｝  
＜声明头部＞   ::=  int＜标识符＞ |char＜标识符＞  
＜变量说明＞  ::= ＜变量定义＞;{＜变量定义＞;}  
＜变量定义＞  ::= ＜类型标识符＞(＜标识符＞|＜标识符＞‘[’＜无符号整数＞‘]’){,(＜标识符＞|＜标识符＞‘[’＜无符号整数＞‘]’ )}  
＜常量＞   ::=  ＜整数＞|＜字符＞  
＜类型标识符＞      ::=  int | char  
＜有返回值函数定义＞  ::=  ＜声明头部＞‘(’＜参数＞‘)’ ‘{’＜复合语句＞‘}’|＜声明头部＞‘{’＜复合语句＞‘}’  //第一种选择为有参数的情况，第二种选择为无参数的情况

＜无返回值函数定义＞  ::= void＜标识符＞(’＜参数＞‘)’‘{’＜复合语句＞‘}’| void＜标识符＞{’＜复合语句＞‘}’//第一种选择为有参数的情况，第二种选择为无参数的情况

＜复合语句＞   ::=  ［＜常量说明＞］［＜变量说明＞］＜语句列＞  
＜参数＞    ::= ＜参数表＞  
＜参数表＞    ::=  ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}  
＜主函数＞    ::= void main‘(’‘)’‘{’＜复合语句＞‘}’  
＜表达式＞    ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞}  
＜项＞     ::= ＜因子＞{＜乘法运算符＞＜因子＞}  
＜因子＞    ::= ＜标识符＞｜＜标识符＞‘[’＜表达式＞‘]’|‘(’＜表达式＞‘)’｜＜整数＞|＜字符＞｜＜有返回值函数调用语句＞           
＜语句＞    ::= ＜条件语句＞｜＜循环语句＞| ‘{’＜语句列＞‘}’｜＜有返回值函数调用语句＞;   
                           |＜无返回值函数调用语句＞;｜＜赋值语句＞;｜＜读语句＞;｜＜写语句＞;｜＜空＞;|＜情况语句＞｜＜返回语句＞;  
＜赋值语句＞   ::=  ＜标识符＞＝＜表达式＞|＜标识符＞‘[’＜表达式＞‘]’=＜表达式＞  
＜条件语句＞::= if ‘(’＜条件＞‘)’＜语句＞else＜语句＞  
＜条件＞    ::=  ＜表达式＞＜关系运算符＞＜表达式＞｜＜表达式＞ //表达式为0条件为假，否则为真  
＜循环语句＞   ::=  while ‘(’＜条件＞‘)’＜语句＞  
＜情况语句＞  ::=  switch ‘(’＜表达式＞‘)’ ‘{’＜情况表＞[＜缺省＞] ‘}’  
＜情况表＞   ::=  ＜情况子语句＞{＜情况子语句＞}  
＜情况子语句＞  ::=  case＜常量＞：＜语句＞  
＜缺省＞   ::=  default : ＜语句＞  
＜有返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’|<标识符> //第一种选择为有参数的情况，第二种选择为无参数的情况  
＜无返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’|<标识符> //第一种选择为有参数的情况，第二种选择为无参数的情况  
＜值参数表＞   ::= ＜表达式＞{,＜表达式＞}  
＜语句列＞   ::= ｛＜语句＞｝  
＜读语句＞    ::=  scanf ‘(’＜标识符＞{,＜标识符＞}‘)’  
＜写语句＞    ::= printf ‘(’ ＜字符串＞,＜表达式＞ ‘)’| printf ‘(’＜字符串＞ ‘)’| printf ‘(’＜表达式＞‘)’  
＜返回语句＞   ::=  return[‘(’＜表达式＞‘)’]

附加说明：

（1）char类型的表达式，用字符的ASCII码对应的整数参加运算，在写语句中输出字符

（2）标识符不区分大小写字母

（3）写语句中的字符串原样输出

（4）情况语句中，switch后面的表达式和case后面的常量只允许出现int和char类型；每个情况子语句执行完毕后，不继续执行后面的情况子语句

（5）数组的下标从0开始

### 2．目标代码说明

目标代码选择32位MIPS指令集：

### 指令 示例 语义

### 系统调用 syscall

### 取立即数 li $s1, 100 $s1 = 100

### 加 add $s3,$s1,$s2 $s3=$s1+$s2

### 立即数加 addi $s3,$s1,100 $s3=$s1+100

### 减 sub $s3,$s1,$s2 $s3=$s1-$s2

### 立即数减 subi $s3,$s1,100 $s3=$s1-100

### 乘 mul $s3,$s1,$s2 $s3=$s1\*$s2

### 除 div $s3,$s1,$s2 $s3=$s1/$s2

### 取字 lw $s1,100($s2) $s1=memory[$s2+100]

### 存字 sw $s1,100($s2) memory[$s2+100]=$s1

### 相等时跳转 beq $s1,$s2,label If($s1==$s2) goto label

### 不等时跳转 bne $s1,$s2,label If($s1!=$s2) goto label

### >=0时跳转 bgez $s1,label If($s1>=$s2) goto label

### >0时跳转 bgtz $s1,label If($s1>$s2) goto label

### <=0时跳转 blez $s1,label If($s1<=$s2) goto label

### <0时跳转 bltz $s1,label If($s1<$s2) goto label

### 无条件跳转 j 2000 Goto 8000

### 跳转至寄存器 jr $ra Goto $ra

### 跳转并链接 jal 2500 $ra=PC+4; Goto 10000;

### 赋值 or $s1,$0,$s2 $s1=$s2

### 立即数赋值 ori $s1,$0,123 $s1=123

### 3. 优化方案\*

需要完成的优化方案包括：

## 基本块内部的公共子表达式删除（DAG图）

## 全局寄存器分配（引用计数或着色算法）

## (3) 数据流分析（通过活跃变量分析，或利用定义-使用链建网等方法建立冲突图

## (4) 代码生成时合理利用临时寄存器（临时寄存器池），并能生成较高质量的目标代码

## 二．详细设计

### 1．程序结构

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

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

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

符号表的数据结构：

符号名，种类（常量/变量（数组）/函数），类型（整型/字符型），值，地址，长度（参数个数/数组长度），所在基本块号。

管理算法：

1. 声明标识符时，遍历符号表查看是否已登录在符号表中，如果未登录，则登录该标识符；否则报错：重复声明。
2. 使用标识符时，遍历符号表查看是否已登录在符号表中，如果未登录则报错：标识符未声明。

### 5．存储分配方案

编译器采用了动态存储分配，其运行栈全部放在内存中，从0x00000000开始，地址逐渐增加。

管理方案：运行栈主要需要用到两个指针，fp和sp：fp只有每次调用函数的时候才移动，fp左边的四个位置，地址从低到高分别为：函数返回地址，函数返回值，上一个sp地址和上一个fp地址；fp右边的位置，地址从低到高分别存放函数的临时变量、参数和局部常变量。而sp则根据程序的运行来移动，始终指向当前函数的最后一个局部常量或变量的地址+4。

运行栈结构：

栈顶（0x00000000） 低地址

全局变量区

Main函数变量区

每次调用函数时：

临时变量

函数返回值

返回地址

参数

局部常变量 高地址

### 6. 解释执行程序\*

不需要解释执行程序。

### 7. 四元式设计\*

* 1. 函数声明

源码形如：

int foo1(int a, int b){}

int foo2{}

int foo3(char c, char d){}

void main{}

中间代码：

func int foo1

para int a

para int b

func int foo2

func void foo3

para char c

para char d

func void main

* 1. 函数调用

源码形如：

i = tar1(x,y)//有返回值

tar2(x,y) //无返回值

中间代码：

push x

push y

call tar1

assign ret i

call tar2

* 1. 函数返回

源码形如：

return (x)

return

中间代码：

ret x

ret

* 1. 变量声明

源码形如：

int i, j;

int a1[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）：

var int i

var int j

var int a1 10

* 1. 常数声明

源码形如：

const int c = 10

const char plus=’+’

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

const char plus +

* 1. 表达式

源码形如：

x = a \* (b + c)

中间代码（可优化）：

add b c t1

mult a t1 t2

assign t2 x

* 1. 条件判断

源码形如：

x == y

x<y

中间代码：

== x y

< 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：

Label1: x = a + b

中间代码：

Label1 :

add a b t1

assign t1 x

* 1. 数组赋值或取值

源码形如：

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

中间代码：

arrget c j t1

mult b t1 t2

arrass t2 i a

### 8. 目标代码生成方案\*

### 本编译器的四元式序列以结构数组intercode[ ]的形式储存，在生成目标代码时，用于四元式转汇编的inter2assem()函数遍历这个结构数组，首先将所有字符串存到.kdata区域，然后分别分析每一种四元式，并将生成的汇编代码直接print到assem.txt中。

### 9. 优化方案\*

由于时间和能力有限，只完成了DAG消除局部公共子表达式的优化，主要用到了以下三个函数：

partition()函数：遍历四元式序列，划分基本块，以既不是赋值语句又不是四则运算与局的语句为划分点。对于每一个基本块，将所有变量填入节点表nodelist[ ]，并生成DAG图，用一个结构数组midnode[ ]表示。非基本块部分直接复制到优化后的四则运算序列中。

optimize()函数：根据上述节点表和DAG图，用启发式算法将所有DAG图中的中间节点加入nodequeue[ ]节点序列，逆序输出该节点序列及其子节点，从而生成优化后的中间代码（四元式序列）。

leftest()函数：启发式算法递归访问最左子节点的递归函数。

### 10. 出错处理

|  |  |  |
| --- | --- | --- |
| 序号 | 错误信息 | 含义 |
| 0 | FILE NOT EXIST | 文件不存在 |
| 1 | ILLEGAL SYM | 非法字符 |
| 2 | UNDEFINED IDENT | 该标识符未定义 |
| 3 | DUPLICATE DEFINE | 标识符重复定义 |
| 4 | WRONG ASSIGN TYPE | 赋值类型错误 |
| 5 | WRONG PARA NUMBER | 函数参数个数错误 |
| 6 | WRONG PARA TYPE | 函数参数类型错误 |
| 7 | LACK SINGLE QUOTES | 缺少“’” |
| 8 | LACK DOUBLE QUOTES | 缺少“”” |
| 9 | LACK LEFT PARENT | 缺少“(” |
| 10 | LACK RIGHT PARENT | 缺少“)” |
| 11 | LACK LEFT BRACKET | 缺少“[” |
| 12 | LACK RIGHT BRACKET | 缺少“]” |
| 13 | LACK LEFT BRACE | 缺少“{” |
| 14 | LACK RIGHT BRACE | 缺少“}” |
| 15 | LACK SEMICOLON | 缺少“;” |
| 16 | LACK EQUALMARK | 缺少“=” |
| 17 | LACK COMMA | 缺少“,” |
| 18 | LACK RETURN VALUE | 缺少返回值 |
| 19 | LACK COLON | 缺少“:” |
| 20 | CANNOT DIVIDE ZERO | 不能除零 |
| 21 | LACK PARAMETERS | 函数定义有括号无参数 |
| 22 | GLOBAL DEFINE  AFTER FUNC | 全局变量常量说明在函数定义后 |
| 23 | WRONG ARRAY LENGTH | 数组长度小于零或数字前有符号 |
| 24 | VOID HAS RETURN VAL | void函数有返回值 |
| 25 | VOIDFUNCUSE  IN FACTOR | 在因子中调用了无返回值函数 |
| 26 | LACK MAIN | 缺少main函数 |
| 27 | MAIN HAS RETURN VAL | main函数有返回值 |
| 28 | MAIN NOT LAST | Main函数后有其他函数 |
| 29 | WRONG RETURN TYPE | 返回值类型错误 |
| 30 | CONST NOT ASSIGN | 常数未赋值 |
| 31 | CONST ASSIGN TWICE | 常数二次赋值 |
| 32 | SIGN BEFORE ZERO | 零前有正负号 |
| 33 | WRONG CASE TYPE | case后常量与switch后表达式类型不同 |
| 34 | LACK CASE | switch后无case |
| 35 | LACK ELSE | if后无else |
| 36 | CONSTDEF AFTER VAR | 常量说明在变量说明后 |

## 三．操作说明

### 1．运行环境

使用CodeBlocks13.12搭建环境。

### 2．操作步骤

将工程文件导入CodeBlocks13.12，单击Build and run按钮，控制台会输出提示信息”Please input the test file”，输入测试文件所在的绝对路径，则程序开始进行编译，在工程文件夹中共生成六个文件：语法分析结果15061091\_result.txt，错误信息error.txt，中间代码inter.txt，目标代码assem.txt，优化后的中间代码newinter.txt，优化后的目标代码newassem.txt。将目标代码文件复制到mars中运行即可得到测试结果。

## 四．测试报告

### 1．测试程序及测试结果

见测试程序文件夹。

### 2．测试结果分析

测试程序能够覆盖全部语法成分。

## 五．总结感想

这学期编译的课程设计总体来说是一个前松后紧的过程，尽管学长和老师多次提醒我们要提前完成进度，但我还是败给了拖延症。在文法解读、词法分析和语法分析阶段都很简单，但是到了中间代码、目标代码的生成和优化环节，工作量突然就变得非常大了。而且还会遇到很多在之前的环节中产生但是没有发现的bug，改掉这些bug又会耽误很多时间。因此，我认为编译课设不仅让我学会工程化地思考问题，更让我意识到合理安排时间的重要性。

在重新完善这份设计文档的时候，我看到了第一次设计我的编译器时存在的很多问题，甚至看到了自己写的“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。”这种完全不对的话。想想自己写词法和语法分析的时候还对编译器究竟是什么、是如何工作的毫无头绪，但是现在对于它的每个部分都已经很清楚了。这就更能证明通过课程设计，我才对编译器有了一个全面透彻的认识，将课上学到的理论知识与实践结合了起来，而不只是会算几个题而已。

其实还有很多想说的话，但是因为我愚蠢地记错了提交最终结果的时间，所以这份总结感想只花了二十分钟草草写成，倍感愧疚。