

**编译原理课程设计**

设计文档

北京航空航天大学

计算机学院

陈麒先

16061160

二○一八年十一月

**郑重声明**

**关于诚实守信公约：**

**设计文档中参考了文献或互联网资料的部分均有引用标注，另有与老师或同学研究后的成果引用将在特别致谢中说明，其余部分均为本人通过查阅课件、课堂笔记和教材后独立思考的结果。另，实验报告撰写仓促，如有错误，在所难免，欢迎批评指正！特此声明。**

**陈麒先**

**原创性声明**

**设计文档中未注释部分内容由作者原创。抄袭行为在任何情况下都是不能容忍的(COPY is strictly prohibited under any circumstances)！转载或引用须征得作者本人同意，并注明出处！勿谓言之不预！**

**陈麒先**



# 编译器设计文档

一、需求分析

### 1．文法说明

1.

＜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＞;}

＜变量说明＞ ::= ＜变量定义＞;{＜变量定义＞;}

＜有返回值函数定义＞ ::= ＜声明头部＞'('＜参数表＞')' '{'＜复合语句＞'}'

＜无返回值函数定义＞ ::= void＜标识符＞'('＜参数表＞')''{'＜复合语句＞'}'

＜主函数＞ ::= void main'('')''{'＜复合语句＞'}'

【范例】

const int a = 1; // 常量说明

int b; // 变量说明

int add(int x,int y){ //有返回值函数定义

return x + y;

}

void print(){ //无返回值函数定义

printf("helloworld");

}

void main(){ //主函数

int c;

b = 1;

c = add(a,b);

print();

}

【分析】

- 根据程序的文法可知，程序各个组成成分的顺序已经被限定好了，不能随便更改顺序，如“int a,b; const x=10;”此种顺序是不允许的。

- 每一个程序按顺序由常量说明部分，变量说明部分，有无返回值函数定义部分，以及主函数组成。其中常量，变量，过程及函数说明说明，复合语句之间有严格的先后顺序，不能打乱。而相连的有无返回值的函数说明部分则可以随意改变顺序。变量说明，常量说明，过程及函数说明对于一个分程序来说均为可有可无的部分，有无返回值的函数定义可以有零到多个。只有最后的主函数是必须存在的。

2.

＜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＞}

＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

＜字符＞ ::= '＜加法运算符＞'｜'＜乘法运算符＞'｜'＜字母＞'｜'＜数字＞'

＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝| 0

＜整数＞ ::= ［＋｜－］＜无符号整数＞

＜数字＞ ::= ０｜＜非零数字＞

＜非零数字＞ ::= １｜．．．｜９

＜字母＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

＜加法运算符＞ ::= +｜-

＜乘法运算符＞ ::= \*｜/

【范例】

const int \_aA = 0; //int常量

const char Ch = '+'; //char常量

【分析】

- 常量说明部分可以说明一至多个常量，每条定义必须有const开始，以分号结束，中间是常量定义。

- 常量定义可以是int也可以是char，对统一类型可定义多个常量，以逗号分隔，定义由等号连接，左边必须为标识符。

- 标识符必须以字母开头，有字母数字组成。

- 字母包含大小写和下划线，程序对大小写敏感。

- 整数的开头为可选的+-加上无符号整数。

- 无符号整数可以是0或者不允许前0的其他数字。

- 字符是能是+-\*/\_大小写字母和数字。

3.

＜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＞']' )}

＜类型标识符＞ ::= int | char

【范例】

int a; // 声明int变量

char b[100]; // 声明长100字符数组

【分析】

- 变量定义不支持在定义时直接赋值，如不允许‘int a = 1;’。

- 类型标识符包括2种，int和char。

- 变量定义包括变量和数组定义,数组定义中无符号整数值表示数组长度，值大于0。

- char类型的变量或常量，用字符的ASCII码对应的整数参加运算。

4.

＜有返回值函数定义＞ ::= ＜声明头部＞'('＜参数表＞')' '{'＜复合语句＞'}'

＜声明头部＞ ::= int＜标识符＞ |char＜标识符＞

＜参数表＞ ::= ＜参数＞{,＜参数＞}| ＜空>

＜参数＞ ::= ＜类型标识符＞＜标识符＞

＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞

【范例】

int add\_1(int x,int y){ //声明头部 参数表 复合语句

const int a = 1; // 常量说明

int result; // 变量说明

result = x + y + a; // 语句列

return result;

}

【分析】

- 有返回值函数定义有声明头部（参数表）{复合语句}组成，顺序不得更改，三者不可缺省，（）{}为必有字符。

- 声明头部由int,char+标识符组成，表示声明的函数返回值类型，二者只能选择其一进行声明。

- 参数表由零至多个参数，传入参数可以为空。各个参数之间由逗号分隔。

- 参数为一个类型标识符加一个标识符组成，不允许多个标识符的情况出现在参数中，如不允许‘int a,b’。

- 复合语句由常量、变量说明和语句列组成，顺序不可调换，常量、变量为可选项，必须有语句列。

- 有返回值的函数定义文法上允许无返回值，但是不符合语义需要报错。

5.

＜无返回值函数定义＞ ::= void＜标识符＞'('＜参数表＞')''{'＜复合语句＞'}'

【范例】

void hello(){ //标识符 参数表 复合语句

printf("hello world!");

}

【分析】

- 无返回值即void 标识符为函数名，其余部分均与有返回值函数定义相同。

- 此处范例给出了参数表空的情况。

- return();是不合法的。

6.

＜主函数＞ ::= void main'('')''{'＜复合语句＞'}'

【范例】

void main(){

printf("hello world!");

}

【分析】

- 主函数中的各个成分是确定的void main(){ 中间为复合语句块 最后以}结尾。

- 无返回值的函数可以有return语句，有返回值可以没有return语句。

7.

＜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＞

【范例】

① a + b

② a[i] \* b + (c[j + k] + d \* e)

③ a \* 4 - 'a'/add(x,y)

④ - a[i \* 4 + base] + c/(d \* e[j])

【分析】

- 表达式有一至多个项组成，每个项之间有加法运算符链接，[+|-]只作用于第一个项，即第一项之前也可有加法算符。

- 项由一至多个因子组成，因子之间由乘法运算符连接。

- 因子可以是一个标识符，即常量或者变量；或是＜标识符＞'['＜表达式＞']'，即一个数组，数组下标为【】中的内容，也可以通过一个表达式进行计算；或是'('＜表达式＞')'由（）包含的部分整体为一个表达式的求值语句；或是一个带返回值的函数调用，通过函数获取数值；或是整数、字符。综上所述，因子应该得到一个数值带回到项中进行计算。

8.

＜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＞;

【范例】

有关语句的范例将通过每个语句语法成分分别给出，敬请期待。

【分析】

- 语句的定义比较易于理解，即是各种不同的语句类型的集合。其中要注意有分号结尾的情况，除了条件、循环、情况语句和'{'＜语句列＞'}'之外，都有分号结尾。注意空语句直接只有一个分号。

9.

＜条件语句＞ ::= if '('＜条件＞')'＜语句＞

＜条件＞ ::= ＜表达式＞＜关系运算符＞＜表达式＞｜＜表达式＞

＜关系运算符＞ ::= <｜<=｜>｜>=｜!=｜==

【范例】

if(a == b){

printf("hello world");

}

【分析】

- 条件语句由关键字if引导，根据（）中的条件进行跳转，若条件为真则执行语句否则不执行。

- 条件分两种，一种是由有关系运算符连接的两个表达式形成的判断条件，另一种是单表达式形成的判断条件。由前文分析可知，表达式会返回数值，表达式为0条件为假，否则为真；而由关系运算符连接的两个数值运算结果将返回0、1，其中0为假，1为真。

- 关系运算符为数值比较运算，返回真值。

10.

＜循环语句＞ ::= while '('＜条件＞')'＜语句＞

【范例】

while(i < n){

printf(i);

i ++;

}

【分析】

- 关键字while ，条件为上文分析过的，如果条件值为真，则执行语句，直到不满足条件退出循环。

11.

＜语句列＞ ::= ｛＜语句＞｝

【范例】

a = 1 + 1;

b[9] = a;

c = b[9] \* 6;

【分析】

- 语句列即一组语句的序列集合，由零到多个语句构成。

12.

＜有返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'

＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}｜＜空＞

【范例】

add(x,y)

【分析】

- 注意有返回值函数调用语句在语句定义中，其应有的分号在语句定义中体现。

- 注意各个成分不得缺省和调换顺序，标识符为调用的函数名。

- 值参数表为传入的实参列表，必须与函数定义的形参相对应，每个相应位置都应该出现数值，可通过表达式求得。

13.

＜无返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'

【范例】

print()

【分析】

- 比有返回值的函数调用简单，标识符为函数名，由（）包含参数列表，这里的范例给出值参数表为空的情况。

- 同样分号是定义在语句中的。

14.

＜赋值语句＞ ::= ＜标识符＞＝＜表达式＞|＜标识符＞'['＜表达式＞']'=＜表达式＞

【范例】

① a = 1

② b[0] = 1

③ c[i \* 4 + base] = 1

【分析】

- 赋值可以为变量或数组中的某一个元素赋值，有=连接。

- 所有形如＜标识符＞'['＜表达式＞']'均表示一个数组，表达式中计算数组索引值，数组的下标从0开始。

- 赋值语句等号右边不能是比较表达式。

15.

＜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＞')']

【范例】

① scanf(a)

② printf("hello world",a + b)

③ return （a + b）;

【分析】

- 读写语句关键词和（，）不可缺少，按顺序排列。标识符为变量名或数组名。

- 写语句分三种，可以写字符串也可写表达式，也可同时写。字符输出char，其余情况输出整数。不能printf(),字符串中不允许双引号（34号字符）。

- 返回语句由关键字return起始，后边的返回值为可选项。对于无返回值的函数定义直接return即可，若有返回值则必须在表达式外加上（）。

- 写语句中，字符串原样输出，单个char类型的变量或常量输出字符，其他表达式按整型输出，写语句自带换行。

- 读语句和变量类型有关，char读入字符，int读整数或字符的ascii码。

16.

＜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＞|＜空＞

【范例】

switch(state){

case 0:{ // 情况表

a = 0;

b = 1;

}

case 1 : a = 1;

default: a = 10; // 缺省

}

【分析】

- 情况语句由关键字switch(){}引导，（）中的表达式为待判断的数量，由{}包含情况表case和缺省情况default。

- 情况表有一至多个情况子语句构成。

- 每个情况子语句由case引导，后面的常量为判断条件，若与表达式的值相等则执行冒号后面的语句。范例中给出了两种可能的语句情况。

- 缺省有default:引导，当前面没有一个case生效的时候执行default后面的语句，缺省情况可以为空。

- <空>是一个特殊的符号，他是一个终结符号ε。

- 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。

### 2．目标代码说明

生成的目标代码为MIPS汇编代码，采用MIPS-C指令集中的基本指令与扩展指令。

【R型指令】

add 加法运算

sub 减法运算

mul 乘法运算

div 除法运算

【I型指令】

addi 加立即数运算

subi 减立即数运算

lw 加载字

sw 存入字

【b类跳转指令】

beq 相等跳转

bne 不等于跳转

bge 大于等于跳转

bgt 大于跳转

ble 小于等于跳转

blt 小于跳转

【j类跳转指令】

j 无条件跳转

jr 跳转到寄存器中存储的地址

jal 跳转并链接，返回地址存储在$ra中，用于函数调用

【扩展指令】

li 加载立即数

move 寄存器赋值

syscall 系统调用

la 加载地址

### 3. 优化方案\*

【方案1】 基本块内优化

- 基本块DAG图

- 消除公共子表达式

- 从DAG图重新导出中间代码

- 窥孔优化

- 常数合并及传播

【方案2】全局优化

- 数据流分析

- 活跃变量分析

- 使用链、网和冲突图

- 消除全局公共子表达式

- 复制传播

- 删除死代码

【方案3】循环优化

- 循环交换

- 循环展开

- 代码外提和循环强度削弱

【方案4】其他方案

- 敬请期待

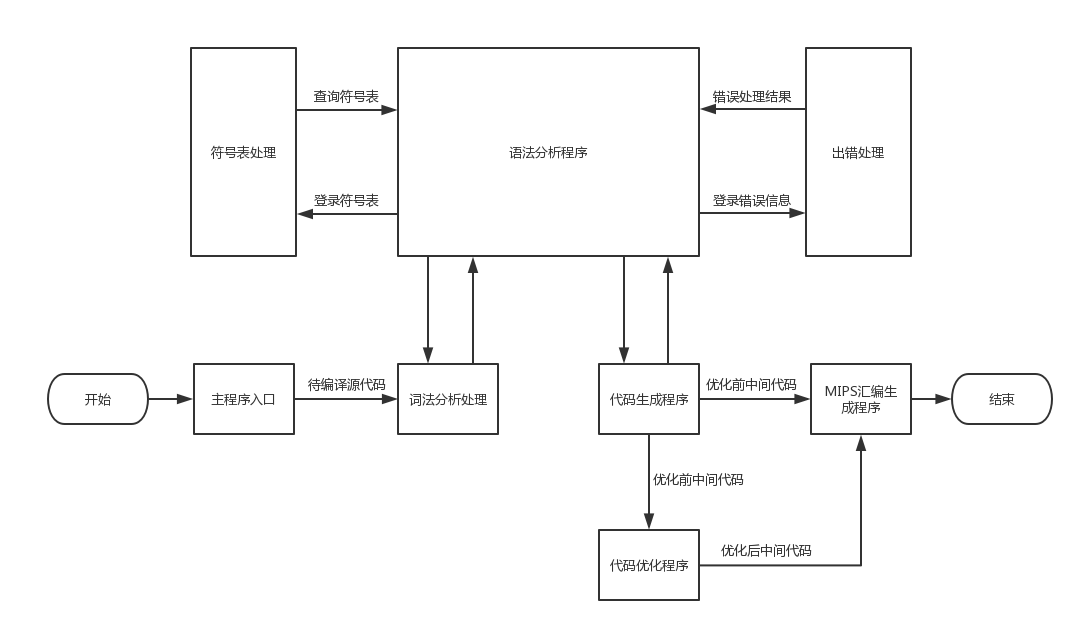
## 二．详细设计

### 1．程序结构

【概述】

程序将仿照Pascal-S程序进行编写。按照编译流程的五个阶段分阶段执行。

【结构框图】



### 2．类/方法/函数功能

编译器采用纯C语言打造。故不体现与类有关内容。以下对各个文件中的核心函数进行解读分析。

(1) 参数定义

【包括文件】

parameters.h ; parameters.c

【功能概述】

用于预定义编译器中所采用的数据结构和变量。

设置自定义类型、常量、全局变量等参数。

【函数定义】

(2) 编译初启

【包括文件】

initial.h ; initial.c

【功能概述】

用于编译阶段的初启动。

【函数定义】

init(); 用于外部调用。

initvar(); 为索引值、变量赋初值。

setup(); 建立保留字表KeyTab，建立特殊字符表SpsTab，建立中间代码指令表MidOpTab，建立目标代码指令表MipsOpTab.

fileSetUp(); 打开各种文件，读入源代码文件和输出中间结果、输出目标结果文件。

(3) 词法分析

【包括文件】

lexical.h ; lexical.c

【功能概述】

用于从源代码中获取下一符号symbol。

【函数定义】

getch(); 用以从源代码中读取下一字符。在getsym()中被调用。

retract(); 指针回退函数。由于采取预读字符的设计，所以读到的某些分支情况需要指针回退。

getsym(); 从源代码通过反复调用上述两个函数读入下一个单词，将单词类型存储至全局变量sy，如果读到为标识符将名称存入全局变量id，如果读到数字则计算其数值，并将其计算出来将结果存储至全局变量inum，如果读到字符串则将其存储入字符串表，如果读到比较运算符等符号则只设置sy为相应类型，其余非法字符进行报错处理。

(4) 语法分析&语义分析

【包括文件】

block.h ; block.c ; enter.h ; enter.c

【功能概述】

用于对待编译源代码的语法和语义进行分析。分析其中的语法成分，以及每一个语句所属的语句类型。将信息登录至各个表项中，并生成四元式中间代码。

【函数定义】

- block.c

block(); 供外部调用的函数，启动语法语义分析程序。

constDeclare(); 处理常量定义。

varDeclare(); 处理变量定义

voidFuncDeclare(); 处理无返回值函数定义。

returnFuncDeclare(); 处理有返回值函数定义。

compoundStatement(); 处理复合语句。

paraDeclare(); 处理参数声明定义。

mainFunc(); 主函数处理。

expression(); 表达式处理。

item(); 项处理。

factor(); 因子处理。

statement(); 语句处理。

assignStatement(); 赋值语句处理。

ifStatement(); 条件语句处理。

condition(); 条件处理：记录跳转相关信息。

whileStatement(); while循环语句处理。

switchStatement(); switch分支语句处理。

caseTable(); 情况表处理。

callReturnFunc(); 有返回值函数调用。

callVoidFunc(); 无返回值函数调用。

scanfStatement(); 读语句处理。

printfStatement(); 写语句处理。

returnStatement(); 返回语句处理。

- enter.c

enter(); 登录到符号表tab信息

enterVar(); 登录变量信息

enterArray(); 登录数组信息到atab

enterFunc(); 登录函数信息

(5) 错误处理

【包括文件】

error.h ; error.c

【功能概述】

用于处理源代码在编译过程中遇到的错误。输出报错信息。

【函数定义】

error(); 错误。根据错误类型输出报错信息和出错位置。

fatal(); 致命错误。如超出表的存储。输出致命错误类型，强制终止编译过程。

(6) 编译优化

【包括文件】

optimize.h ; optimize.c

【功能概述】

用于对已生成的中间代码进行优化。

注：由于此部分内容胡春明老师班课程进度未覆盖至本章节，故再此仅给出个人理解的实现流程。疏漏和不当之处在所难免，将于后续编译器的完善过程中修正。

【函数定义】

optimize(); 提供给外部调用，用于完成代码优化操作。

(7) 生成目标代码

【包括文件】

mid2Mips.h ; mid2Mips.c

【功能概述】

将中间四元式代码转化成MIPS汇编码。

【函数定义】

strLabel(); 为字符串常量在.data域生成label

pushReg(); 寄存器值压入运行栈

clearReg(); 清空寄存器

allocReg(); 分配寄存器

id2Reg(); 根据分配的寄存器编号与MIPS实际寄存器之间转化

varCount(); 变量计数器

calculateIns(); 处理+ - \* / 运算指令

assignIns(); 处理赋值指令

jumpIns(); 处理无条件跳转指令

conditionJumpIns(); 处理条件跳转指令

functionIns(); 处理函数并生成相关指令

inputIns(); 处理输入生成相应指令序列

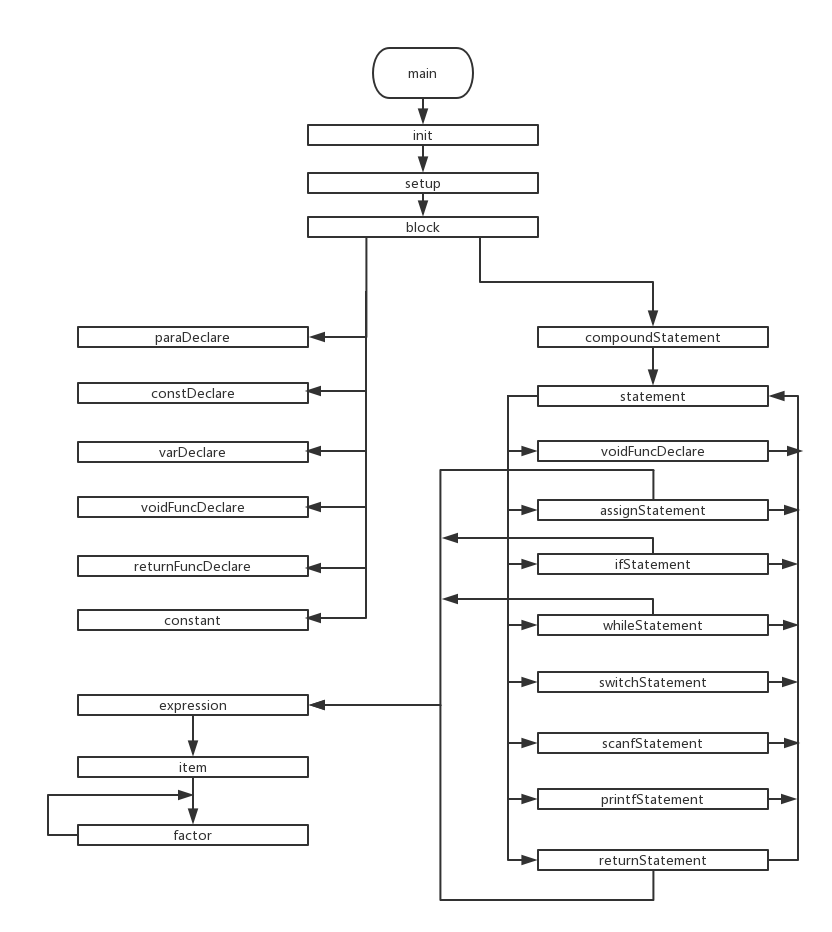
outputIns(); 处理输出生成相应指令序列

parasIns(); 处理参数传递生成相应指令序列

callIns(); 处理函数调用相应指令序列

### 3．调用依赖关系

【调用依赖关系图】



### 4．符号表管理方案

【概述】

符号表均采用struct结构体进行存储。

【具体实现】

1. tab 符号表

【索引变量】tx

struct tab{

name; 标识符名

link; 指向统一子程序中上一个标识符在tab中的位置。

obj; 标识符种类：const,var,typel,func

typ; 标识符类型：int,char,notype

ref; 当标识符为数组类型名、数组变量名时，ref指向atab中该数组的登录位置；当标识符为函数时，ref指向该函数在btab中的位置。

lev; 标识符所在层次。

adr; 对于变量、形参，填入该变量、形参在运行栈中分配存储单元的相对地址；对于函数名，填入中间代码入口地址；对于整数和字符常量名，填入其具体的值。

}

(2) btab子程序表

【索引变量】bx

struct btab{

last; 子程序中说明的当前最后一个标识符在tab中的位置

lastpar; 指向函数中最后一个参数在tab中的位置

psize; 参数占据的存储单元数

vsize; 局部变量占据的存储单元数

}

(3) display子程序索引表

文法中不允许子程序的嵌套定义，故层次数（从0开始计数）只有0,1两种取值。

int display[2]; 该一维数组索引变量即为层次数。更具其值从btab中查找相应项的内容。

(4) atab数组信息向量表

【索引变量】ax

struct atab{

eltype; 数组元素类型 int/char

high; 数组上界（数组从下标0开始计数）

elsize; 元素大小

size; 数组大小

}

(5) stab字符串常量表

【索引变量】sx

char stab[SMAX][STRMAX];

文法限定仅当输出语句时，才可以使用字符串。词法分析中读到字符串，就将其登录至stab,并记录起始位置和长度。

### 5．存储分配方案

(1) 运行时的存储组织方案

【概述】

采用动态存储分配的方式。

【具体实现】

为每个程序模块申请一片数据区域（运行栈），并保留该空间直至执行完整个模块为止。

此外应维护一个数据栈。对于局部变量(常量)、形参、全局变量(常量)利用tab表查找找到其在数据栈中的相对当前数据区的基地址偏移；对于临时变量建立临时变量数据表进行存储，根据临时变量的名字查找其所在的数据栈地址偏移。

1. 运行栈的结构

【图示】



【说明】

假设运行栈向上生长。

内容(沿运行栈生长方向)依次是：

全局数据区，用以存放最外层的全局数据（常量、变量）；

ret\_addr存储返回地址，返回地址应为当前指令的下一条指令的地址(pc + 4)；

prev\_abp指向前一个活动记录(AR)的基地址；

存放函数调用的返回值；

存放局部变量和参数；

存放临时变量等信息等。

### 6. 解释执行程序\*

本难度的编译器不要求实现解释执行程序。

### 7. 四元式设计\*

(1) 四元式结构体定义

struct midCode{

int op; 操作指令

char z; 结果

char x; 第一个操作数

char y; 第二个操作数

}

(2) 采用的四元式指令表示法

【概述】

四元式指令采用类MIPS汇编指令。具体指令表示方法如下。当然对于部分的跳转指令，或是其他功能的指令，不是严格的4个操作数，这里采用类4元式表示形式。

【表示方式说明】

1） 运算指令

+ z, x, y 加法指令 add/addi

- z, x, y 减法指令 sub

\* z, x, y 乘法指令 mul

/ z, x, y 除法指令 div

= z, (x), y 赋值指令 mov

注：赋值指令的三种形式分别为z = y 、 z = y[x] 、 z[x] = y

2） 跳转指令

PC = z 以下设置z为label值，从而指向一个具体代码地址。

beq z, x, y x == y, goto z

bne z, x, y x!=y, goto z

bge z, x, y x>=y, goto z

bgt z, x, y x>y, goto z

ble z, x, y x<=y, goto z

blt z, x, y x<y, goto z

ret y 函数返回 y 值

j z 无条件跳转到label/PC值

jal z 跳转并链接, 紧跟call指令使用，用于处理函数调用

3）其他指令

setlab y 设置标签名字为y

func x , y 函数声明，y为函数名, x为类型 int, char, void

para x , y 函数定义时的形参，x为类型，y为形参名

in y 调用scanf，将值放到y

out x , y 输出字符串x, 表达式的值y

passpara y 传参y

call y 函数调用操作, y为函数名称

### 8. 目标代码生成方案\*

(1) 代码生成相关的重要数据结构

- 中间代码存储结构

struct midCode{ 单条中间代码

int op; 操作符

char z; 中间结果

char x; 操作数（寄存器、标签）

char y;

int offset; 偏移量或立即数

}

struct midCodeTable{ 中间代码表

midCode midCodes[MIDCODEMAX]; 中间代码表

int mcx; mid code index 索引变量

}

- 寄存器状态存储结构

struct reg{ 单个寄存器

char name; 寄存器名称

int kind; 寄存器存储数据的种类（常量，局部变量，全局变量）

int offAddr; 偏移地址

int isEmpty; 寄存器可用标志

}

struct reg\_use{ 用于存储的寄存器集

reg regs[REGSMAX]; 使用的寄存器集

int rx; reg index 当前空闲的寄存器下标

int full; 判断是否已满

}

(2) 实现代码生成的关键算法

1、操作数处理算法

建立tmap,smap两个表，用以存放临时变量和局部变量。

对于z,x,y三个操作数，根据其类型的不同执行不同的操作：

- 若为数字，则直接获取其数值并返回。

- 若为临时变量，则从tmap中寻找其位置：在寄存器中存储，则获取寄存器编号，若在数据存储区存储，则使用lw将其加载到一个未使用的寄存器中，并获取此寄存器的编号。

2、寄存器清空算法

根据寄存器的功能，我们使用$8-$25寄存器。

共设计4种清空寄存器策略：

- 清空不写入内存，用于函数声明阶段。

- 清空并将运行信息写入存储，用以保存现场，为函数调用做准备。

- 清空只将全局变量写入存储，用以保存函数执行结果，函数返回。

- 清空临时变量寄存器，保留全局寄存器，用于函数内部分层次使用。

3、标签处理方案

首先应决定标签种类，根据不同标签需要实现的功能生成不同标签。

- 循环、分支控制标签：先根据2中第4条策略清空寄存器，设置标签。

- 主函数入口标签：设置$fp(函数基址)和$sp(栈顶)的值，分配全局寄存器。

- 函数起始标签：根据函数名生成标签，记录pc值。

- 常量定义标签：.data域中生成一些需要打印输出的字符串标签和数组标签。

4、函数处理方案

- 函数声明处理：根据2中1策略清空寄存器，清空stab，设置display子程序索引表。

- 函数调用处理：保存下一条指令值至$ra，设置prevabp，保存函数标签至栈。

- 函数返回处理：处理返回值，从栈中恢复调用时刻状态，pc返回$ra的位置。

### 9. 优化方案\*

【特别说明】

注：由于此部分内容胡春明老师班课程进度未覆盖至本章节，故再此仅给出个人结合课本相关内容，理解的实现流程。疏漏和不当之处在所难免，将于后续编译器的完善过程中修正。以下部分内容参考了他人的成果。

(1) 删除局部公共子表达式

1）采用的数据结构 –- 二叉树

struct tree\_node {

// 叶子节点对应的属性

char name; // 叶子节点所对应的变量名

// 中间节点对应属性

int op\_id; // +,-,\*,/,= 的操作符对应的midop编号, -1 代表其为叶子节点

struct tree\_node\* lchild; //左孩子node编号

struct tree\_node\* rchild; //右孩子node编号

link var\_names; //该节点所对应的变量名称链表

// 公共部分

int is\_mid\_node; //是否为中间节点

list parents; //父节点, 对两个孩子节点的父节点进行更新

};

2）具体实现流程

首先根据函数的开始标签地址和结束标签地址划分函数块，然后再函数块内找到连续的表达式(加减乘除和赋值语句)进行处理，先根据表达式生成DAG图，再使用启发式算法进行导出。其中对于变量名的更新和局部变量重定义要进行注意：

a.变量名的更新：由于公共子表达式的删除会使删除程序中一些冗余的临时变量，而为了防止表达式块后的代码对删除掉的变量进行引用导致出错，以函数块为单位建立变量名的Old\_New\_Map对照表，然后在一个函数块处理完成之后再重新从头扫描更新变量名。

b.局部变量重定义：先使用Fn对重定义的变量进行节点的变量名替换，并将其对应关系加入F\_S\_Map中，之后再从T\_Var中取出临时变量名对 Old\_New\_Map进入信息插入(Fn，Tn)。其中如果中途计算得到的F类变量名数量大于T\_var的数量，则不对该代码块进行优化，因为新增的重定义赋值语句个数(a0 = a这种)已经大于被删除的语句了。

(2) 全局寄存器分配及临时寄存器调度策略

1）利用引用计数

由于我们为局部变量分配的寄存器共有10个，其余的均为临时变量寄存器。

在main函数和其他函数的声明标签处对该函数进行全局寄存器分配。首先扫描函数体内出现的局部变量，将其次数信息加入到引用计数表中。若出现的局部变量个数小于等于10个，则每个局部变量分配一个寄存器；若局部变量的个数大于10个，则对引用计数表中的计数信息进行从大到小的排序，为前十个分配全局寄存器，剩下的局部变量在用到时使用临时寄存器。

2）FIFO策略分配寄存器

除全局寄存器以外，剩下的都是临时寄存器，使用FIFO进行调度，在层次变换的时候对临时寄存器池进行清空操作(使用寄存器清空的第4种情况)。

### 10. 出错处理

【处理方案】

对于在编译阶段出现的错误，需要划分为两种不同的情况进行处理。

其一，对于编译中发现的不符合语法语义的错误、非法字符错误等不会导致编译过程崩溃的错误类型，将采取报错并在一定限度内容错的机制。编译过程继续执行，并继续对源代码进行分析。

其二，对于编译中出现的严重错误，比如导致表存储空间不足，或数据溢出的情况。这种“致命”错误，将会导致编译过程崩溃。为了保护程序的安全，需要强制终止编译过程。

【报错信息】

**1.error 类型的错误信息:**

errorMsg[52] = {

"源代码不完整", //0

"整数超出范围", //1

"字符不能为空", //2

"字符结尾应是\'", //3

"识别到字符串中的非法字符", //4

"字符串不以双引号结尾", //5

"!后面未连接=", //6

"识别到程序中的非法字符", //7

"识别到字符中的非法字符", //8

"非零整数检测到前导0", //9

"main声明错误", //10

"标识符缺失错误", //11

"函数返回值与函数声明不匹配", //12

"缺少\'(\'", //13

"缺少\')\'", //14

"缺少\'[\'", //15

"缺少\']\'", //16

"缺少\'{\'", //17

"缺少\'}\'", //18

"缺少\';\'", //19

"缺少\'=\'", //20

"if、while后缺少条件", //21

"case的内容不为常量", //22

"标识符未定义", //23

"标识符重复定义", //24

"数组越界", //25

"类型声明应为int、char", //26

"常量的赋值与声明的类型不符", //27

"变量声明格式错误", //28

"const声明顺序错误", //29

"数组声明格式错误", //30

"函数声明格式出错", //31

"常量声明、变量声明或者语句头符号错误",//32

"表达式格式错误", //33

"未检测到有返回值函数调用", //34

"缺少关键词while", //35

"形参个数与实参个数不匹配", //36

"检测到对常量赋值错误", //37

"赋值语句格式错误", //38

"scanf语句格式错误", //39

"printf语句格式错误", //40

"return语句格式错误", //41

"switch语句错误", //42

"case语句格式错误", //43

"除0错误", //44

"形参与实参类型不匹配", //45

"不能对char赋值int类型", //46

"case 常量与switch中表达式类型不匹配"， //47

"函数定义错误", //48

"数组越界或索引值表达式非法", //49

"条件格式错误" //50

};

**2.fatal:**

char fatalmsg[7] = {

"词法分析阶段字符串表溢出", //0

"符号表tab溢出", //1

"子程序表btab溢出", //2

"数组表atab溢出", //3

"中间代码表溢出", //4

"mips代码表溢出" //5

};

## 三．操作说明

### 1．运行环境

【说明搭建运行环境的步骤】

### 2．操作步骤

【详细说明操作步骤】

## 四．测试报告

### 1．测试程序及测试结果

【给出提供的测试程序以及每个程序的测试结果，至少5个正确程序，5个错误程序，无需截屏】

### 2．测试结果分析

【说明上述测试程序对语法成分的覆盖情况】

## 五．总结感想

【说明在完成课程设计中的收获、认识和感想】

**陈麒先**

2018/11/16