《编译技术》

课程设计文档

学号：\_\_\_\_\_16061011\_\_\_\_\_\_\_

姓名：\_\_\_\_\_\_牛宇航\_\_\_\_\_\_\_\_

2019 年 1 月 4 日

# 一．需求说明

## 1．文法说明

1. **获取的文法**

＜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＞｝| 0

＜整数＞ ::= ［＋｜－］＜无符号整数＞

＜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＞']' )} //＜无符号整数＞表示数组元素的个数，其值需大于0

＜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＞'('＜参数表＞')' '{'＜复合语句＞'}'

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

＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞

＜参数表＞ ::= ＜参数＞{,＜参数＞}| ＜空>

＜参数＞ ::= ＜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＞} //[+|-]只作用于第一个<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＞

＜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＞|＜空＞

＜有返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'

＜无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＞

＜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类型的变量或常量，用字符的ASCII码对应的整数参加运算

（2）标识符区分大小写字母

（3）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

（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

1. **文法改写**

＜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＞｝| 0

＜整数＞ ::= ［＋｜－］＜无符号整数＞

＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

＜声明头部＞ ::= ＜类型标识符＞＜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＞']' )} //＜无符号整数＞表示数组元素的个数，其值需大于0

＜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＞'('＜参数表＞')' '{'＜复合语句＞'}'

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

＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞

＜参数表＞ ::= ＜参数＞{,＜参数＞}| ＜空>

＜参数＞ ::= ＜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＞} //[+|-]只作用于第一个<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＞

＜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＞|＜空＞

＜有返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'

＜无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＞

＜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＞')']

改动总结：声明头部修改

1. **文法扩充说明**

printf后默认不换行。

## 2．目标代码说明

要生成的目标代码为32位MIPS汇编代码，MIPS指令集还包括了MARS支持的部分伪指令，具体包含的指令及含义如下：

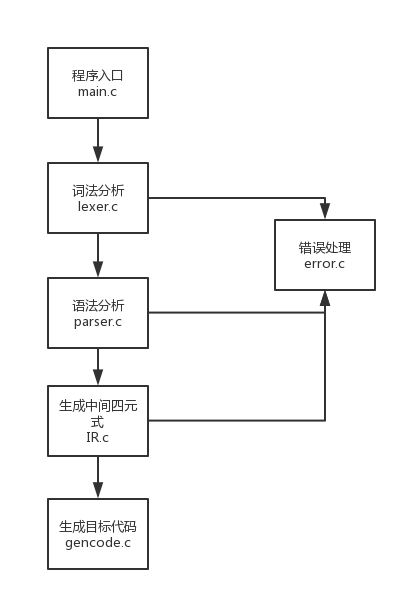
|  |  |  |  |
| --- | --- | --- | --- |
| 指令 | 助记符 | 示例 | 含义 |
| 加 | addu | addu $d, $s, $t | $d = $s + $t |
| 减 | subu | subu $d, $s, $t | $d = $s - $t |
| 乘 | mult | mult $s, $t | (Hi, Lo) = $s \* $t |
| 除 | div | div $s, $t | LO = $s / $t  HI = $s % $t |
| 取低位结果 | mflo | mflo $d | $d = LO |
| 取立即数 | li | li $s, 1 | $s = 1 |
| 赋值 | move | move $s, $t | $s = $t |
| 立即数加 | addi | addi $t, $s, immed | $t = $s + immed |
| 立即数减 | subi | subi $t, $s, immed | $t = $s - immed |
| 取字 | lw | lw $t, offset($s) | $t = MEM[$s + offset] |
| 存字 | sw | sw $t, offset($s) | MEM[$s + offset] = $t |
| 等于时跳转 | beq | beq $s, $t, label | if $s == $t  PC = label  else  PC = PC + 4 |
| 不等于时跳转 | bne | bne $s, $t, label | if $s != $t  PC = label  else  PC = PC + 4 |
| 大于0时跳转 | bgtz | bgt $s, label | if $s > 0  PC = label  else  PC = PC + 4 |
| 大于等于0时跳转 | bgez | bgez $s, label | if $s >= 0  PC = label  else  PC = PC + 4 |
| 小于0时跳转 | bltz | blzt $s, label | if $s < 0  PC = label  else  PC = PC + 4 |
| 小于等于0时跳转 | blez | blez $s, label | if $s <= 0  PC = label  else  PC = PC + 4 |
| 读取地址 | la | la $t0, array | set $t0 to address of array |
| 无条件跳转 | j | j target | PC = nPC  nPC = (PC & 0xf0000000) | (target << 2) |
| 跳转返回 | jr | jr $ra | PC = $ra |
| 跳转并链接 | jal | jal label | PC = label  $ra = PC + 4 |
| 系统调用 | syscall | syscall | 系统函数 |

## 3. 优化方案\*

计划使用DAG图消除基本块内部的公共子表达式，常量替换等优化，实际由于时间原因，没有实现，只是用了$t0-$t3四个临时寄存器。

# 二．详细设计

## 1．程序结构



## 2．类/方法/函数功能

1. **词法分析**

|  |  |
| --- | --- |
| 函数名 | 功能 |
| void getch() | 从源代码中读一个字符，跳过\n和\t |
| void push\_in\_symList() | 将读出的sym填入符号表symlist中 |
| int isinkey() | 判断是否为保留字 |
| void getsym() | 得到下一个sym |

1. **语法分析**

|  |  |
| --- | --- |
| 函数 | 功能 |
| void init() | 初始化符号表 |
| void updateFuncVarNum() | 填写完符号表后，更新函数大小 |
| void nameATempVarByIndex() | 根据给定的下标命名临时变量 |
| void nameATempVar() | 命名新的临时变量 |
| void nameALabel() | 命名新的label |
| void nameAString() | 命名新的字符串 |
| void enter() | 将符号存入符号表 |
| void updateParaNumber() | 函数定义中读入所有参数后更新函数参数个数 |
| struct node findIdentInSymTable() | 四元式生成目标代码时，在当前层和全局变量区查找并返回当前变量 |
| struct node findIdentInLastSymTable() | 生成四元式时，在当前层（即符号表中最后一层）和全局变量区查找并返回当前变量 |
| struct node findLabel() | 在生成目标代码时，根据四元式label找到对应函数名 |
| struct node findFunction() | 查找函数 |
| struct node findPara() | 查找参数 |
| struct node findFunctionByLayer() | 根据层数查找函数 |
| int isInteger() | 判断是否为整数 |
| void program() | 处理程序 |
| void condecl() | 处理常量说明 |
| void constdef() | 处理常量定义 |
| void vardecl() | 处理变量说明 |
| void vardef() | 处理变量定义 |
| void headdef() | 处理声明头部 |
| void paratable() | 处理函数声明时参数表 |
| void para() | 处理参数表中的参数 |
| void funcdef() | 处理有/无返回值函数定义 |
| void funccall() | 处理有/无返回值函数调用 |
| void valueparatable() | 处理值参数表 |
| void mainprog() | 处理主函数 |
| void complexsentence() | 处理复合语句 |
| void sentenceslist() | 处理语句列 |
| void sentence() | 处理语句 |
| void expression() | 处理表达式 |
| void term() | 处理项 |
| void factor() | 处理因子 |
| void assignsentence() | 处理赋值语句 |
| void ifsentence() | 处理if条件语句 |
| void whilesentence() | 处理while循环语句 |
| void switchsentence() | 处理switch情况语句 |
| void casetable() | 处理情况表 |
| void casesentence() | 处理情况子语句 |
| void defau() | 处理缺省 |
| void printfsentence() | 处理printf写语句 |
| void scanfsentence() | 处理scanf读语句 |
| void returnsentence() | 处理return返回语句 |
| void conditionsentence() | 处理条件语句 |

1. **错误处理**

|  |  |
| --- | --- |
| 函数 | 功能 |
| void error() | 根据错误编号输出错误 |

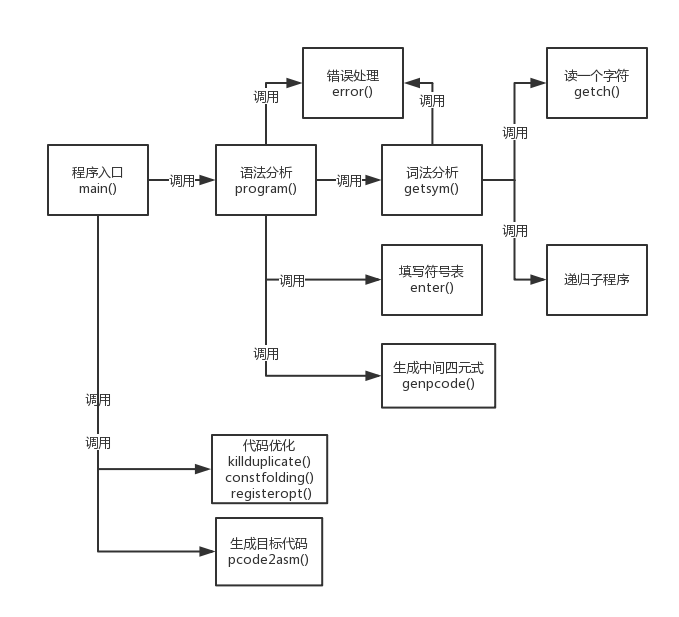
1. **生成中间四元式**

|  |  |
| --- | --- |
| 函数 | 功能 |
| void insertIntoIRlist() | 将中间四元式加入四元式表中 |

1. **生成目标代码**

|  |  |  |
| --- | --- | --- |
| 函数 | 功能 | 关键算法 |
| void gendata() | 生成.data段目标代码 |  |
| void gentext() | 生成.text段目标代码 |  |

## 3．调用依赖关系



## 4．符号表管理方案

采用动态栈式符号表，在分程序表记录该分程序首个变量下标，每一层的符号表具体内容如下：

struct node {

char name[100]; // 标识符名称

int kind; // 标识符类型，1:constant,2:variable,3:function,4:para,5:tempVar,6:STRING

int type; // 函数的返回值类型或其他标识符的小类型，1:void,2:int,3:char,4:int arrays,5:char arrays

int value; // 常量的值或ascii码，立即数对应临时变量的值，其他类型此处为0

int number; // 函数参数个数或数组元素个数，其他类型此处为0

int addr; // 内存中标识符相对于所在AR首地址的偏移量，数组为数组首地址的偏移量

char STRING[100]; // 字符串的内容

char label[100]; // 函数的label

int sum; // 一个函数中变量、常量、临时变量、参数的内存大小总和+4(包括4个ra的位置和4个fp位置)

int level; // 该node所在分程序下标};

struct symtable{

struct node table[MAX\_SYMTABLE\_SIZE]; // 程序所有标识符信息数组

int top; // 符号表栈顶的位置

int subprogramNumber; // 分程序总数

int subprogramTable[MAX\_LAYER]; // 分程序索引};

## 5．存储分配方案

使用$t0-$t3四个临时寄存器保存计算中固定生成的变量，其余局部变量和临时变量保存在内存中，需要保存的字符串和全局变量变量存储在.data区中。

运行栈对课本中的经典运行栈进行了大幅更改，具体如下：

低地址

高地址

|  |
| --- |
| 栈顶 |
| 参数区 |
| $ra |
| $fp |
| 局部数据区 |

寻址时由高地址向低地址寻址，$sp指向下一个要填写的位置，$fp指向当前函数头位置。

在函数调用时，向$sp处填入值参数，$sp自减4，所有值参数填写完，继续保存父函数的$ra和父函数的$fp。$sp自增至填函数前的位置，令$fp移到$sp位置，jal指令更新$ra，开始运行子函数。

在结束函数调用时，jr指令回到ra中的指令，将本指向子函数头部的$fp的值赋给$sp，作为父函数尾部，从$sp向下读出刚才保存的$ra和$fp，继续运行父函数。

## 6. 解释执行程序\*

无解释执行程序。

## 7. 四元式设计\*

【对采用的四元式进行详细说明】

|  |  |  |
| --- | --- | --- |
| 操作 | 四元式 | 解释 |
| int a; | varop,int,0,a | 声明int型普通变量 |
| char a; | varop,char,0,a | 声明char型普通变量 |
| int a[5]; | arrop,int,10,a | 声明int型数组变量 |
| char a[5]; | arrop,char,5,a | 声明char型数组变量 |
| const int a=5; | conop,int,5,a | 声明int型常量 |
| const char a=’c’; | conop,char,99,a | 声明char型常量 |
| a | getop,a,,res | 拿出普通变量的值 |
| ‘a’ | getiop,97,,res | 拿出字符或数字的值 |
| b[i] | getaop,b,i,res | 拿出数组变量的值 |
| a = b; | stoop,b,,a | 赋值给普通变量 |
| a[i] = b | stoop,b,i,a | 赋值给数组变量 |
| f() | callop,,,f | 无参数函数调用 |
| f(a,...) | paraop,,,a  callop,,,f | 有参数函数调用 |
| If (a<b) | bgezop,a,b,label | 若条件语句后表达式为0，跳走 |
| If(a<=b) | bgtzop,a,b,label | 若条件语句后表达式为0，跳走 |
| If(a>=b) | bltzop,a,b,label | 若条件语句后表达式为0，跳走 |
| If(a>b) | blezop,a,b,label | 若条件语句后表达式为0，跳走 |
| If(a==b) | Bneop,a,b,label | 若条件语句后表达式为0，跳走 |
| If(a!=b) | Beqop,a,b,label | 若条件语句后表达式为0，跳走 |
| set label; | setop,,,label | 设置label |
| return; | retop,,, | 返回语句无返回值 |
| return a; | retop,,,a | 返回语句有返回值 |
| 得到返回值 | getrop,a,, | 得到返回值 |
| scanf(a); | scaop,,,a | 读变量a |
| printf(a); | priop,,,a | 写变量a |
| printf(“a”); | priop,a,, | 写字符串 |
| printf(“a”,b); | priop,a,,b | 写字符串和变量b |
| a + b | addop,a,b,res | 表达式a+b |
| a – b | subop,a,b,res | 表达式a-b |
| a \* b | multop,a,b,res | 表达式a\*b |
| a / b | divop,a,b,res | 表达式a/b |
| 无条件跳转 | jop,,,label | 无条件跳转 |
| 跳转并连接 | jrop,,, | 跳转到ra |
| 接受函数f返回值 | getrop,res,,f | 接受函数f返回值 |
| 函数声明int fun(int a, char b) | setop,,,label  fstaop,,,f  ……  jrop,,, | 函数声明 |
| 程序结束 | endop,,, | 程序结束 |

## 8. 目标代码生成方案\*

根据四元式和符号表生成目标代码，目标代码为x86 mips汇编代码，后缀名为.asm。局部变量的生成方案如下，全局变量仅有取值存值位置的区别：

|  |  |
| --- | --- |
| conop,int,5,a | Li $t0, 5  sw $t0, -offset($fp) |
| getop,a,,res | lw $t0, -offset1($fp)  sw $t0, -offset2($fp) |
| getiop,97,,res | li $t0,97  sw $t0, -offset($fp) |
| getaop,b,i,res | la $t0, res  lw $t1, -offset1($fp)  li $t2, 4  mult $t1, $t2  mflo $t1  addu $t0, $t0, $t1  lw $t0, ($t0)  sw $t0, -offset2($fp) |
| stoop,b,,a | lw $t0, -offset1($fp)  sw $t0, -offset2($fp) |
| callop,,,f | sw $ra, ($sp)  sw $fp, -4($sp)  addi $sp, $sp, %d  move $fp, $sp  addi $sp, $fp, -%d  jal f  move $sp, $fp  lw $fp, -offset($sp)  lw $ra, -offset($sp) |
| paraop,,,a | lw $t0, -offset1($fp)  sw $t0, ($sp)  addi $sp, $sp, -4 |
| bgezop,a,b,label | lw $t0, -offset1($fp)  lw $t1, -offset($fp)  subu $t0, $t0, $t1  bgez $t0, label |
| bgtzop,a,b,label | lw $t0, -offset1($fp)  lw $t1, -offset($fp)  subu $t0, $t0, $t1  bgtz $t0, label |
| bltzop,a,b,label | lw $t0, -offset1($fp)  lw $t1, -offset($fp)  subu $t0, $t0, $t1  bltz $t0, label |
| blezop,a,b,label | lw $t0, -offset1($fp)  lw $t1, -offset($fp)  subu $t0, $t0, $t1  blez $t0, label |
| bneop,a,b,label | lw $t0, -offset1($fp)  lw $t1, -offset($fp)  bne $t0, $t1, label |
| beqop,a,b,label | lw $t0, -offset1($fp)  lw $t1, -offset($fp)  beq $t0, $t1, label |
| setop,,,label | $label\_0: |
| retop,,, | jr $ra |
| retop,,,a | lw $v0, -offset($fp)  jr $ra |
| scaop,,,a | li $v0, 5  syscall  sw $v0, -offset($fp) |
| priop,,,a | lw $a0, -offset($fp)  li $v0, 1  syscall |
| priop,a,, | la $a0, a  li $v0, 4  syscall |
| getrop,a,,, | sw $v0, -offset($fp) |
| addop,a,b,res | lw $t0, -offset($fp)  lw $t1, -offset($fp)  addu $t0, $t0, $t1  sw $t0, -offset($fp) |
| subop,a,b,res | lw $t0, -offset($fp)  lw $t1, -offset($fp)  subu $t0, $t0, $t1  sw $t0, -offset($fp) |
| multop,a,b,res | lw $t0, -offset($fp)  lw $t1, -offset($fp)  mult $t0, $t1  mflo $t0  sw $t0, -offset($fp) |
| divop,a,b,res | lw $t0, -offset($fp)  lw $t1, -offset($fp)  div $t0, $t1  mflo $t0  sw $t0, -offset($fp) |
| jop,,,label | j label |
| jrop | jr $ra |
| endop,,, | li $v0, 10  syscall |

## 9. 优化方案\*

因为时间问题，仅开始了DAG图消除公共子表达式，但没有完成预期优化，存在bug，因此不表。

## 10. 出错处理

|  |  |  |
| --- | --- | --- |
| 错误类型 | 错误编号 | 错误说明 |
| UNDEF\_ID | 1 | 未声明标识符 |
| MULTI\_DEF | 2 | 重定义标识符 |
| PROGRAM\_ERROR | 3 | 程序中主函数后仍有代码 |
| STR\_EMPTY | 4 | 空字符串 |
| STR\_CONTENT\_ERROR | 5 | 字符串内容错误 |
| STR\_MISSINT\_RPARENT | 6 | 字符串在该行没有右双引号 |
| ZERO\_HEAD\_NUM | 7 | 数字开头有前导零 |
| CHAR\_MISS\_QUOTE | 8 | 字符类型中字符后不紧跟单引号 |
| CHAR\_CONTENT\_ERROR | 9 | 左单引号后符号不合法 |
| NEQUAL\_MISS | 10 | 感叹号后不是等号 |
| MISSING\_SEMICOLON | 11 | 丢失分号 |
| MISSING\_COLON | 12 | 丢失冒号 |
| MISSING\_RPARENT | 13 | 丢失右括号 |
| MISSING\_RBRACK | 14 | 丢失右中括号 |
| MISSING\_RBRACE | 15 | 丢失右大括号 |
| MISSING\_IDENT | 16 | 丢失标识符 |
| ERROR\_PARA\_NUM | 17 | 参数数目错误 |
| MISSING\_LPARENT | 18 | 丢失左小括号 |
| MISSING\_LBRACE | 19 | 丢失左大括号 |
| MISSING\_INTEGER | 20 | 丢失整数 |
| MISSING\_NUMBER | 21 | 丢失无符号整数 |
| MISSING\_IS | 22 | 丢失一个等号（赋值符号） |
| MISSING\_CHAR | 23 | 丢失字符类型（‘a’） |
| RETURN\_ERROR | 24 | 返回值类型错误或缺少返回值 |
| MISSING\_MAIN | 25 | 丢失main函数 |
| MISSING\_CASE\_KEYWORD | 26 | 丢失case关键字 |
| WRONG\_ASSIGN\_TYPE | 27 | 赋值类型错误 |
| CASE\_CONTENT\_WRONG | 28 | case内容错误 |
| FACTOR\_CONTENT\_WRONG | 29 | 因子内容错误 |
| SENTENCE\_CONTENT\_WRONG | 30 | 语句内容错误 |
| OUT\_OF\_ARRAY | 31 | 数组越界 |
| WRONG\_TYPE | 32 | 声明类型错误 |
| FILE\_NOT\_EXIST | 33 | 源文件不存在 |
| ILLEGAL\_WORD | 34 | 非法字符（如汉字） |
| FUNC\_NO\_DEFINED | 35 | 函数未定义 |
| FUNC\_NO\_RET | 36 | 无返回值函数当做表达式使用 |
| ERROR\_PARA\_TYPE | 37 | 参数类型错误 |
| ASSIG2CONST | 38 | 给常量赋值 |
| CONDITION\_TYPE\_ERROR | 39 | 条件中类型不匹配或类型不为整型 |
| ARRAY\_INDEX\_NOT\_INT | 40 | 数组下标不为int |
| SWITCH\_TYPE\_ERROR | 41 | Switch中类型不匹配 |

错误处理：

发现错误后将错误类型传给error，进行错误输出并进行相应的跳读。

# 三．操作说明

## 1．运行环境

Windows系统VS2017运行编译器。

MIPS代码使用MARS虚拟机运行。

## 2．操作步骤

打开项目内的.sln文件，运行程序后在控制台输入源代码路径。MIPS指令在MARS中运行。

# 四．测试报告

## 1．测试程序及测试结果

test1：求阶乘

输入：5

输出：120

输入：1

输出：1

test2：测试switch和while

输入：3

输出:num=1num=1num=1

test3：测试关系运算符

输入：98

输出：<===>=

输入：97：

输出：<<=!=

输入：99

输出>>=!=

test4：测试printf

输出：

!#$%&'()\*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{|}~\nerror

test5：测试数组

输出：1224455

error1：遍历数组错误

output：16个错

error2：switch错误

output：4个错误

error3：缺少右括号

output：1个错误

error4：缺少分号

output：2个错误（分号后不是标识符时退出复合语句，需要右括号）

error5：无main函数

output：1个错误

## 2．测试结果分析

test1: 编译时程序没有报错，说明常量声明、变量声明、函数声明、主函数声明是正确的。同时也可以说明主函数中的scanf和return被正确调用。程序求0阶乘部分正确。if部分正确，printf部分正确。函数声明、调用部分正确。

test2：运行test\_while\_and\_switch()函数正确，说明while部分正确。switch后为整数情况正确调用。程序未输出error，while至此完整覆盖。

test3：运行test\_relation()函数正确，关系运算符部分正确。

test4：switch进入default，switch完整覆盖。printf完整覆盖。

test5：成功遍历所有赋值和数组情况。

# 五．总结感想

我自己给自己的编译课设打分可能为负了……心心念念的优化因为自己临近考期，疲于复习，没有完成。最后只能完成基本功能和错误处理，心里是很失望的。

编译课设开始时间比较晚，但一周两个DDL确实很紧凑，在优化之前的工作中，生成中间代码和目标代码的两周最为紧凑。虽然给了两周时间，但是期间还要完成其他科目的工作，实际上依然是半周写完程序。而考期时的优化也由于时间分配问题没有实现，只做了一部分的DAG图消除公共子表达式的工作，因为存在bug而不能使用。

编译课设有点类似于OO课的每周作业，不同的是，OO虽然很多地方不规定，但编译的规定却是在讨论区接二连三的出现，每次ddl前偶然翻看时发现硬性规定又要修改代码……希望课程组能为下一届提前制定好详细的要求，如果要补充硬性规定，可以提前通过微信群通知。

值得一提的是在测试过程中，我的助教刘一帆学长真的是编译课设中体验最好的部分了……希望编译能越办越好吧，也希望自己在以后的课程中能更好的利用时间，完成要求。