## 全局定义：

### CONST\_STRING

所有printf的字符串要进行定义

调用函数defineConstStr()，命名\_STRING\_{}

生成label：.asciiz “string”

### VAR\_INT

定义全局变量，类型为int，暂不区分int和char，

调用函数defineGlobalInt()，命名\_GLOBAL\_VAR\_{}

生成label .space 4

### VAR\_INT\_ARRAY

定义全局数组变量，类型为int，暂不区分int和char,

调用函数defineGlobalIntArray()，命名\_GLOBAL\_ARR\_{}

生成label .space len\*4

注意：

## 符号表与运行栈的维护

SymbolTable :: getStackScopeBytes()

获取函数在运行栈中的空间

第一部分，在符号表中的空间，包括参数，函数内定义的常量和变量

即从scopeStack .back()->symbolStack .size()

第二部分，返回地址压栈($ra)，临时变量池压栈(regpool)，（寄存器压栈）

要维护 extra\_scope\_offset

该函数返回栈顶$sp到当前函数头部的距离

SymbolTable::getStackOffsetBytesByIdent(string ident)

计算符号表中一元素，相对当前栈顶$sp的偏移量

首先，计算符号表中该元素到函数头部scopeStack .back()的距离bytes。

相对栈顶偏移量offset = getStackScopeBytes() - bytes – 4

所以该标识符ident代表的变量位于 offset($sp)

pushRegToStack 或 popRegFromStack

在函数运行过程中，将传入参数代表的寄存器，push/pop到栈中

并维护extra\_scope\_offset，+4：addScopeOffset(4)或-4 subScopeOffset(4)

Midcode：PUSH\_REG POP\_REG

Mips：

addi $sp, $sp, -4 lw $reg, ($sp)

sw $reg, ($sp) addi $sp, $sp, 4

## 表达式计算

思路：先转换为逆波兰表达式，按照其逆波兰表达式计算顺序生成代码

符号：优先级 加=减<乘=除<左括号=右括号<负号

数据结构：op\_stack，obj\_stack，\*\_stack\_stack用于处理表达式嵌套

过程：

表达式开始，前表达式op与obj压栈，清空op与obj的临时栈。（**exprStart**）

将数据压入obj\_stack，符号压入op\_stack （**pushOp，pushExpObj\_\***）

压入符号前，若op\_stack不为空且当前符号优先级大于等于栈顶符号（**checkOp**），则弹出栈顶符号，并生成相应运算指令，循环直至不满足上述条件（**popOp**）。

**genExprVal()**：表达式符号全部结束后，生成表达式结果，循环弹出栈顶符号生成相应运算指令，最终返回保存表达式结果的临时变量(寄存器)

**exprEnd ()**：\*\_stack\_stack弹出栈顶元素，恢复前表达式的op\_stack与obj\_stack

相关指令生成：

**pushExpObj\_\***

exprPushObj\_GlobalVar

从label取

midcode：LOAD\_LABEL newTemp() 🡨 getGlobalVarLabel

mips：lw $t, label

exprPushObj\_StackVar

从栈中取，sp+偏移量

Midcode：LOAD\_STACK newTemp()🡨 offset

Mips：lw $t, offset($sp)

exprPushObj\_StackArr

（已计算出地址），从栈中取，寄存器所指地址

Midcode：LOAD\_STA\_ARR newTemp()🡨 addr

Mips：lw $t, ($addr)

exprPushObj\_ImmInt

使用add，立即数与$0运算，存入临时寄存器

Int->string，char->int->int

Mips：addu $t $0 imm

运算指令生成：

Add，sub，mul，div，两元运算，neg一元运算使用sub实现

### 因子

整型常量(integer)：调用exprPushObj\_ImmInt

字符型常量(char)：转换成int，调用exprPushObj\_ImmInt

标识符：查符号表checkSymbolIsGlobal，checkSymbolIsConst，checkSymbolIsChar

若为常量(const int或const char)，转换为整型常量，调用exprPushObj\_ImmInt

全局变量，调用exprPushObj\_GlobalVar，传入label

局部变量，调用exprPushObj\_StackVar，传入偏移量offset

数组标识符：

计算数组下标地址(getArrAddr）, 调用 exprPushObj\_StackArr

表达式：新建表达式栈

## 数组

### 数组的定义

全局数组：见前文全局定义部分，调用defineGlobalIntArray()，设置label，并使用.asciiz开具4\*len的大小的空间

局部数组：作为变量会加入符号表，在复合语句compStatement中，结束varDef后会调用openStackSpace开具相应空间

### 数组的存取

调用getArrAddr()计算数组地址

首先计算数组下标×4的值

若是全局数组，则地址位于label+数组下标×4

若是局部数组，则地址位于数组名在符号表中相对$sp的偏移量，再减去数组下标×4，再加上$sp

返回存储着数组元素地址的寄存器（暂设为$k0）

存数组元素（作为右值）

Midcode：SAVE\_STA\_ARR

Mips：sw $target reg, ($addr)

取数组元素（作为左值）

Midcode：LOAD\_STA\_ARR

Mips：lw $target reg, ($addr )

注意：由于临时使用$k0寄存器，数组元素的地址计算完要立刻使用！！！

## 赋值语句

全局变量赋值

Midcode：SAVE\_LABEL reg 🡪label

Mips：sw $reg label

局部变量(栈)赋值

Midcode：SAVE\_ STACK reg 🡪offset($sp)

Mips：sw $reg offset($sp)

数组赋值

计算数组下标地址

Midcode：SAVE\_STA\_ARR reg 🡪 ($addr)

Mips：sw $reg ($addr) 暂时用$k0

## 跳转语句

### 条件

根据逻辑符号类型，和左右表达式生成指令

midCodes. genBranch(cmpToken. type, lexprVal, rexprVal);

逻辑为：**失败时跳转**到指定(结束或else)位置（原文法语义是成功时执行）

EQL🡪BNE lexprVal, rexprVal, label

NEQ🡪BEQ

GRE🡪BLE

GEQ🡪BLT

LSS🡪BGE

LEQ🡪BGT

若只有一个表达式无比较运算符，则为>0执行，即<=0跳转，生成BLEZ指令

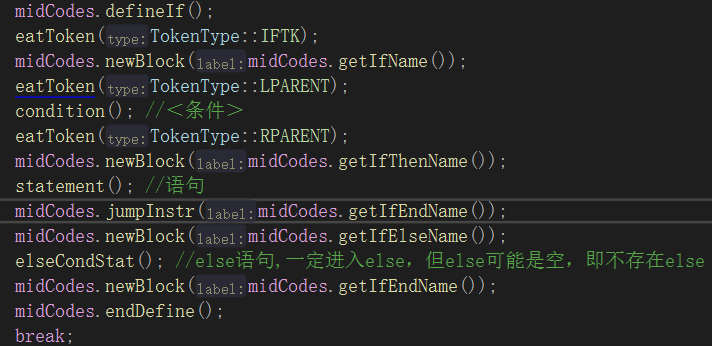
对于跳转到的label，要根据BranchDefine的类型进行判断

IF\_DEFINE跳转到else label（因为即使没有else语句一定生成了else的label）

FOR,WHILE,DOWHILE跳转到相应的END label

### 条件语句（BranchDefine IF\_DEFINE）

＜条件语句＞ ::= if '('＜条件＞')'＜语句＞［else＜语句＞］



newBlock(Get\*Name)函数生成相应的label的block

defineIf新建BranchDefine结构，记录id，并且压入branch\_stack中

先生成if name //未用到，仅作标记使用，

注意then在condition后循环语句之前，即循环开始的位置

If循环语句结束后生成跳转到end的指令

Else一定会生成label但不一定有语句

在else后生成end label

endDefine\*，从branch\_stack弹出当前结构

### for循环语句

for'('＜标识符＞＝＜表达式＞;＜条件＞;＜标识符＞＝＜标识符＞(+| -)＜步长＞')'＜语句＞

0、defineIf新建BranchDefine结构，记录id，并且压入branch\_stack中

1、＜标识符＞＝＜表达式＞

＜标识符＞ - for\_init\_token 🡨 ＜表达式＞结果，调用assignToIdent，为for\_init\_token进行初始化

2、getForStartName()＜条件＞

在<条件>前生成START，然后进行条件判断与跳转

3、＜标识符＞＝＜标识符＞(+| -)＜步长＞

记录step\_ident\_token🡨标识符；step\_op\_token🡨(+| -)；step\_val\_token🡨＜步长＞

4、getForBodyName()＜语句＞

在<语句>前生成循环体Body标志，未用到，仅作为标记使用

5、执行完语句，利用表达式计算循环变量step\_ident\_token的值

首先将循环变量step\_ident\_token压数据栈；然后根据step\_op\_token生成相应运算符号压符号栈；将step\_val\_token生成立即数压数据栈，最后将表达式计算生成的值赋给step\_ident\_token（调用assignToIdent）

6、生成无条件跳转到START的指令，重新进行条件判断

7、生成END label，结束FOR\_DEFINE的定义(调用endDefine)

### While与dowhile循环语句

define\*开始定义，再调用getWhileName()生成label

再循环语句(while)或条件判断(dowhile)结束后生成无条件跳转到开头label的跳转语句

调用getWhileEndName()生成结束标志

endDefine结束定义

## 输入语句(scanf)

Scanf支持同时读多个变量的值，先将所有标识符存入scanfIdentList中

在符号表(局部+全局)中按名查找标识符代表的symbol，调用**findSymbolAndScope**

调用**scanf2midInstr**

判断symbol是否是全局变量，类型是INT还是CHAR

全局INT

Midcode：SCAN\_GLOBAL\_INT label (调用**midCodes .getGlobalVarLabel**)

Mips：li $v0, 5

Syscall

sw $v0, label

全局CHAR

Midcode：SCAN\_GLOBAL\_CHAR label (调用**midCodes .getGlobalVarLabel**)

Mips：li $v0, 12

Syscall

sw $v0, label

局部INT

Midcode：SCAN\_INT offset (调用symbolTable. getStackOffsetBytesByIdent)

Mips：li $v0, 5

Syscall

sw $v0, offset($sp)

局部CHAR

Midcode：SCAN\_CHAR offset (调用symbolTable .getStackOffsetBytesByIdent)

Mips：li $v0, 5

Syscall

sw $v0, offset($sp)

## 输出语句(printf)

**tie(expr\_type, expr\_ans\_reg) = expr()**，接收表达式返回的类型和结果所在寄存器

字符串输出 先定义字符串(defineConstStr)，再调用printStrInstr输出

Midcode：PRINT\_STRING string

Mips：la $a0 label

li $v0, 4

syscall

INT输出

Midcode：PRINT\_INT expr\_ans\_reg

Mips：move $a0, $reg

li $v0, 1

syscall

CHAR输出

Midcode：PRINT\_CHAR expr\_ans\_reg

Mips：move $a0, $reg

li $v0, 11

syscall

LINE输出换行

Midcode：PRINT\_LINE

Mips：li $a0 ‘\\n’

li $v0, 11

syscall

## 返回语句

生成跳转指令，跳转到函数返回的label

若有返回值，则将先调用moveReg()将返回值表达式的结果送到$v0

## 函数

### 函数定义

调用defineFunc()，生成相应function name为label的Func结构，并加入midCode中的funcs中。

将参数，函数局部常量和变量加入符号表中，**VarDef后开具相应的栈空间**

然后将$ra压栈，完成相应语句列的指令生成，**生成函数返回的label**，再将$ra退栈

调用**popStack()**，退出上述开具的栈空间

调用**clearScopeOffset()**，将extra\_scope\_offset清零

再函数返回的label后，生成真正的跳转指令(已经退回$ra)

### 函数调用

**pushRegPool()**将临时变量寄存器(内存)池压栈

在函数调用传入的参数表(funcCallInputParaList函数)中，调用pushRegToStack()对每一个参数进行压栈操作，并维护extra\_scope\_offset+4

然后调用subScopeOffset()，extra\_scope\_offset – 4\*para\_num，因为进入函数后会重新开具参数的空间

调用callInstr()生成jal跳转指令，到函数的label

**popRegPool()**临时变量寄存器(内存)池出栈

代码生成