《编译技术》课程设计

文 档

学号：\_\_\_\_\_\_\_\_16061146\_\_\_\_\_\_\_\_\_

姓名：\_\_\_\_\_\_\_\_金泽晖\_\_\_\_\_\_\_\_\_\_\_

2018年 11月 18日

## 一．需求说明

### 1．文法说明

获取的文法为难度等级3的扩充c0文法，文法描述参考链接：

<https://github.com/Nerdary/BUAA_Compiler_Design/blob/master/%E6%89%A9%E5%85%85c0%E6%96%87%E6%B3%95.docx>

### 2．目标代码说明

目标代码为基于MIPS指令集的32位汇编代码

|  |  |  |
| --- | --- | --- |
| 指令 | 示例 | 含义 |
| add | add $t1 $t2 $t3 | $t1 = $t2 + $t3 |
| sub | sub $t1 $t2 $t3 | $t1 = $t2 - $t3 |
| mul | mul $t1 $t2 $t3 | $t1 = $t2 \* $t3 |
| div | div $t1 $t2 $t3 | $t1 = $t2 / $t3 |
| beq | beq $t1 $t2 6 | if($t1==$t2)goto PC+4+24 |
| bne | bne $t1 $t2 6 | if($t1!=$t2)goto PC+4+24 |
| j | j 10000 | goto 10000 |
| jar | jal 10000 | $31 = PC + 4;goto 10000 |
| jr | jr $31 | goto $31 |
| slt | slt $t1 $t2 $t3 | if($t2<$t3)$t1=1 else$t1=0 |
| lw | lw $t1 10($t2) | $t1 = memory[$t2+10] |
| sw | lw $t1 10($t2) | memory[$t2+10] = $t1 |
| syscall | (code in $v0) | system call |

### 3. 优化方案\*

1. 运用基本块算法，将中间代码划分为基本块

2. 通过构建DAG图消除局部公共子表达式

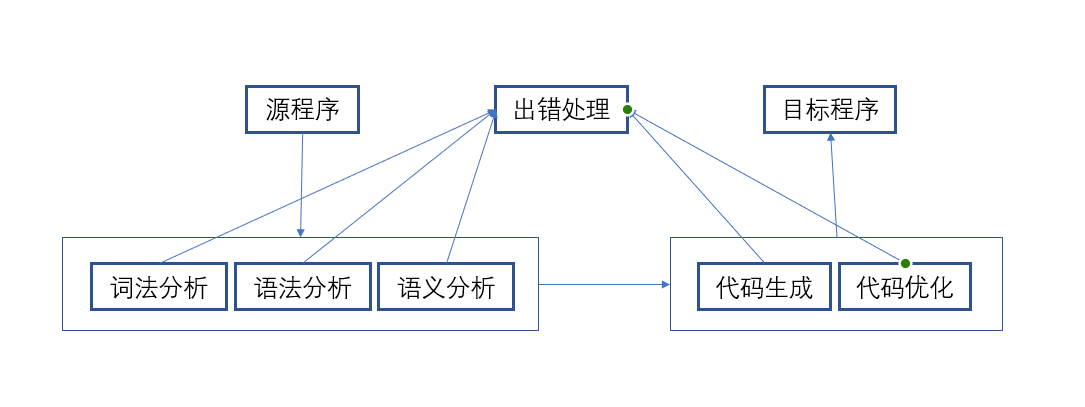
3. 从DAG图中导出中间代码的启发式算法

4. 通过数据流分析建立活跃冲突图

5. 使用着色算法进行全局寄存器的分配

## 二．详细设计

### 1．程序结构

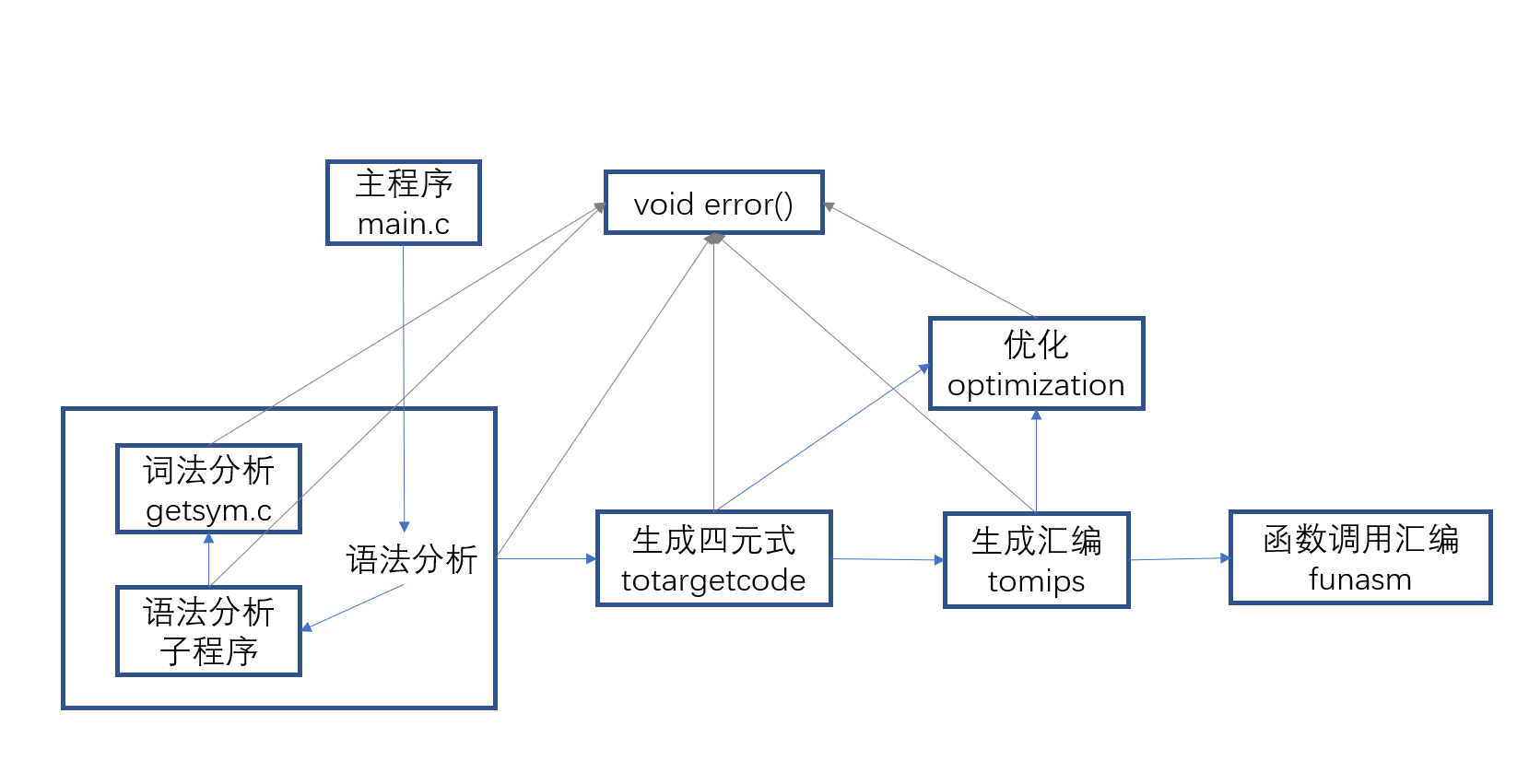


### 2．类/方法/函数功能

【描述各类/方法或函数的功能，以及关键算法】

|  |  |  |
| --- | --- | --- |
| 函数名 | 功能 | 关键算法 |
| getsym | 词法分析子程序 | 状态图法 |
| program | 程序分析主程序 |  |
| conststate | 常量声明 | 递归子程序法 |
| varstate | 变量声明 | 递归子程序法 |
| mainprogram | 主函数 | 递归子程序法 |
| returnedfunc | 有返回值函数调用 | 递归子程序法 |
| noreturnfunc | 无返回值函数调用 | 递归子程序法 |
| defhead | 声明头部 | 递归子程序法 |
| parameterlist | 参数列表 | 递归子程序法 |
| complexsentence | 复合语句 | 递归子程序法 |
| sentencesequence | 语句列 | 递归子程序法 |
| sentence | 语句 | 递归子程序法 |
| ifsentence | 条件语句 | 递归子程序法 |
| condition | 条件 | 递归子程序法 |
| forsentence | 循环语句 | 递归子程序法 |
| assignsentence | 赋值语句 | 递归子程序法 |
| readsentence | 读语句 | 递归子程序法 |
| writesentence | 写语句 | 递归子程序法 |
| returnsentence | 返回语句 | 递归子程序法 |
| expression | 表达式 | 递归子程序法 |
| term | 项 | 递归子程序法 |
| factor | 因子 | 递归子程序法 |
| tomips | 生成汇编 |  |
| funasm | 函数调用生成汇编 |  |
| optimization | 优化函数 | DAG图、引用计数 |
| error | 错误处理 |  |

### 3．调用依赖关系



### 4．符号表管理方案

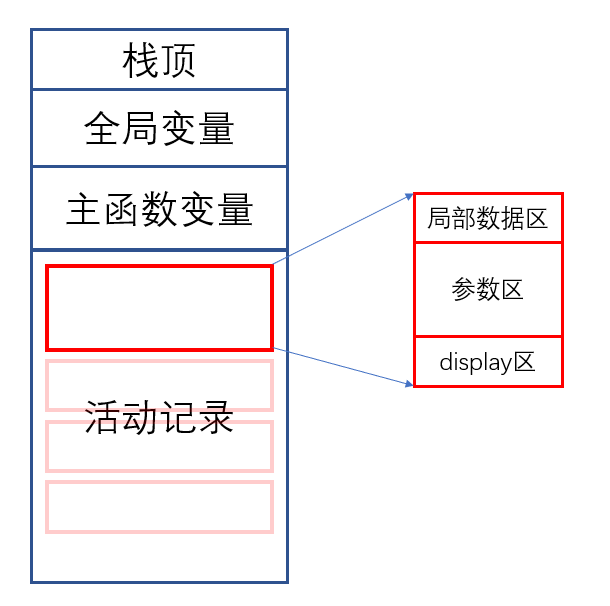
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 符号表设计 | | | | |
| 属性 | 类型 | | | |
| 常量 | 变量 | 数组 | 函数 |
| name | name | name | array name | func name |
| para\_num | -1 | -1 | -1 | para num |
| odject | constant | variable | array | function |
| type | int/char | int/char | int/char | int/char/void |
| ref | / | / | atab index | btab index |
| normal | / | / | / | 标识符为变量形参时为false |
| adr | 相应的值或ascii值 | 该变量在运行栈S中分配的存储单元相对地址 |  | 相应目标代码入口地址 |
| link | 指向同一分程序上一个标识符在符号表中的位置 | | | |

|  |  |
| --- | --- |
| 数组符号表（atab） | |
| element\_type | 数组元素的类型 |
| low | 下界 |
| high | 上界 |
| element\_size | 数组元素大小 |
| size | 数组大小 |

|  |  |
| --- | --- |
| 分程序表（btab） | |
| last | 该分程序中说明的当前最后一个标识符在符号表的位置 |
| last\_par | 指向过程或函数的最后一个参数在符号表的位置 |
| psize | 参数及在运行栈中内务信息区所占空间 |
| vsize | 局部变量、参数、内务信息区在运行栈所占空间 |

### 5．存储分配方案

采用存储分配方案，设立运行栈



### 6. 解释执行程序\*

本文法不需要解释执行程序

### 7. 四元式设计\*

1、 函数声明：

源码： int foo(int a, char c)

四元式：int foo()

para int a

para char c

2、 函数调用：

源码： I = foo(x, y)

四元式：push x

push y

call foo

I = RET

3、 函数返回：

源码： return (x)

四元式：ret x

4、 变量声明：

源码： int a, b, c;

四元式：var int a

var int b

var int c

5、 常量声明：

源码： const char c = ‘s’;

四元式：const char c ‘s’

6、 表达式：

源码： x = a \* (b + c / d) - e

四元式：t1 = c / d

t2 = b + t1

t3 = a \* t2

t4 = t3 – e

x = t4

7、 条件判断：

源码： x == y

四元式：x == y

8、 有条件跳转：

源码： /

四元式：bne label 1

9、 无条件跳转：

源码： /

四元式：j label 2

10、标号语句：

源码： Labal 1: x = a + b

四元式：Label 1:

x = a + b

11、数组赋值语句：

源码： a[i] = b

四元式：a[i] = b

12、数组取值语句：

源码： b = a[i]

四元式：t1 = a[i]

b = t1

13、输出语句：

源码： print(“string”,char\_array)

四元式：print “string”

print char\_array

14、输入语句：

源码： scanf(tmp)

四元式：t1 = input

tmp = t1

15、变量赋值语句：

源码： I = +16061146;

四元式：I = 16061146

### 8. 目标代码生成方案\*

数据结构：

struct Address{

char name[200]; // 变量名字

int addr; // 变量地址

int type; // 数据类型

int is\_global; // 全局变量标识

}addrlist[1024]; // 数组容量为1024

int sp = INITSTACK; // 栈指针

int fp = 0; // 帧指针

int ap; // 地址列表计数器

关键算法：

根据is\_global标签判断是否是全局变量，全局变量基地址为预先保存在寄存器中的值，局部变量的基地址为sp寄存器所指的地址。

目标代码生成程序的输入为DAG图，按图的节点生成顺序依次翻译成mips汇编代码。

### 9. 优化方案\*

1、利用DAG图消除公共子表达式

首先划分基本块，遍历所有四元式，绘制DAG节点图，将之后出现相同子树的四元式消除。

2、根据引用计数分配全局寄存器

在遍历四元式的同时对出现的变量进行引用计数，按降序排列后将引用最频繁的变量存入寄存器以优化访问的效率。

### 10. 出错处理

语法递归子程序、词法分析程序遇到错误时调用error()并输入相应的错误类型编号。error()负责打印相关信息到控制台并将错误日志写入error.txt文件中。跳过相关错误代码片段后继续编译直至输入的源代码文件结束。

## 三．操作说明

### 1．运行环境

codeblocks 17.12

生成的mips32指令通过mars1生成汇编代码

### 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】

注：【】内的文字为文档模板说明，完成的作业中需去掉。

标\*的章节需根据题目的难度进行取舍。