**《编译原理》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **姓名** | | **谭勇** | | | **年级** | **2019级** |
| **学号** | | **20194278** | | | **专业、班级** | **计科4班** |
| **实验名称** | **目标代码生成程序的设计与实现** | | | | | |
| **实验时间** | **2022.5.18** | | **实验地点** | **DS3401** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 □设计性 □综合性** | | |
| 教师评价：  □算法/实验过程正确；□源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □报告规范；  实验代码得分（满分7分）：  实验报告得分（满分3分）：  评价教师签名： | | | | | | |
| 一、实验目的  在词法分析、语法分析及语义分析的接触上，为编译器实现语义分析、代码生成功能。 | | | | | | |
| 二、实验项目内容  一）实验内容  需根据文法规则及语义约定，采用自顶向下的语法制导翻译技术，进行语义分析并生成目标代码（**MIPS** 形式）。完成编译器，将源文件（统一命名为testfile.txt）编译生成MIPS汇编并输出到文件（统一命名为mips.txt），具体要求包括：  a）需自行设计四元式中间代码，再从中间代码生成MIPS汇编，请设计实现输出中间代码的有关函数，本次作业不考核，后续会有优化前后中间代码的输出及评判(输出文件命名为学号\_姓名\_优化前/后中间代码.txt)。  b）若选择此项任务，后续可参加竞速排序，请提前预留代码优化有关的接口，并设计方便切换开启/关闭优化的模式  c）自行调试时，可使用Mars仿真器（使用方法见“Mars仿真器使用说明.docx”），提交到平台的编译器只需要能按统一的要求生成MIPS汇编代码文件即可。  d）**此类提交的编译器请仅读取testfile.txt文件并生成相应的MIPS代码，编译器自身不要读入标准输入中的内容。**  【输入形式】testfile. txt为符合文法要求的测试程序，另外可能存在来自于标准输入的数据。  【输出形式】 按如上要求将目标代码生成结果输出至mips.txt中，中文字符的编码格式要求是UTF-8。  【特别说明】   1. 本次作业是为了让同学们尽快实现一个完整的编译器，测试程序中仅涉及常量说明、变量说明、读语句、写语句、赋值语句，无函数定义及调用，无数组声明及引用。 2. 本次作业仅用正确的测试程序进行考核，但产生的编译器应具有错误处理能力。 3. 本次作业未要求优化生成代码，实现优化的同学会有加分。请在报告中写明采用优化方法和优化测试样例，建议尝试提交竞速排名作业进行测试。 | | | | | | |
| 三、实验过程或算法（源程序）  第四次实验基于前三次实验（词法分析、语法分析、错误处理）。    目标代码生成是编译的最后阶段，将编译器此前生成的中间代码和符号表以及其他相关信息作为输入，输出与源程序语义等价的目标程序代码。  从四元式到目标代码的生成需要结合符号表，因为需要明确变量的作用域，检验是否满足语义规则等信息。对于跳转和返回，需要结合自己的运行栈设计结合以及符号表中存储的信息进行必要的运行栈加减操作。  其中，四元式的结构如下表所示：   |  |  |  |  | | --- | --- | --- | --- | | **分组** | **编号** | **四元式** | **说明** | | **基本算术运算** | 1 | ADD  SRC1  SRC2  DST | 加：DST = SRC1 + SRC2 | | 2 | SUB  SRC1  SRC2  DST | 减：DST = SRC1 - SRC2 | | 3 | MULT  SRC1  SRC2  DST | 乘：DST = SRC1 \* SRC2 | | 4 | DIV  SRC1  SRC2  DST | 除：DST = SRC1 / SRC2 | |  |  |  |  | | **普通赋值** | 5 | ASSIGN  SRC1  DST  ---- | DST = SRC1 | | **数组存取** | 6 | STOA  VALUE  ARRAY  OFF | ARRAY[OFF] = VALUE | | 7 | GETA　ARRAY  OFF  VALUE | VALUE = ARRAY[OFF] | | **函数调用** | 8 | CALL  SRC1  ----  ---- | 调用函数 | | **逻辑判断跳转** | 9 | BNE  SRC1  SRC2  LABLE | SRC1 != SRC2则跳转到LABEL | | 10 | BEQ  SRC1  SRC2  LABLE | SRC1 == SRC2则跳转到LABEL | | 11 | BLE  SRC1  SRC2  LABLE | SRC1 <= SRC2则跳转到LABEL | | 12 | BLT  SRC1  SRC2  LABLE | SRC1 < SRC2则跳转到LABEL | | 13 | BGE  SRC1  SRC2  LABLE | SRC1 >= SRC2则跳转到LABEL | | 14 | BGT  SRC1  SRC2  LABLE | SRC1 > SRC2则跳转到LABEL | | **无条件跳转** | 15 | GOTO  LABLE  ----   ---- | 跳转到LABEL | | **传参数** | 16 | PARA  SRC  DST     ---- | 把SRC作为参数传给函数DST | | **函数返回** | 17 | RETURN  DST  ----   ---- | DST作为返回值返回 | | 18 | RETURN  ----   ----   ---- | 无返回值函数返回 | | **函数结束** | 19 | END  ----   ----   ---- | 函数结束 | | **读写指令** | 20 | PRINTS   DST   ----     ---- | 写字符串 | | 21 | PRINTC  DST   ----     ---- | 写字符 | | 22 | PRINTF  DST   ----     ---- | 写整型 | | 23 | SCANC  DST   ----     ---- | 读字符 | | 24 | SCANF  DST   ----     ---- | 读整型 |   对应的MIPS指令大致如下：   |  |  |  |  | | --- | --- | --- | --- | | **编号** | **指令** | **例子** | **例子说明** | | **1** | j | j main | 跳转到main | | **2** | subi | subi $sp, $sp, 4 | $sp = $sp – 4  这个指令用于获取一段栈空间 | | **3** | sw | sw $ra, 0($sp) | 把寄存器$ra中的值存到地址0($sp) | | **4** | bnez | bnez $t1, label | 如果$t1 != 0, 则跳转到label | | **5** | beqz | beqz $t1, label | 如果$t1 == 0, 则跳转到label | | **6** | bgez | bgez $t1, label | 如果$t1 >= 0, 则跳转到label | | **7** | bgtz | bgtz $t1, label | 如果$t1 > 0, 则跳转到label | | **8** | blez | blez $t1, label | 如果$t1 <= 0, 则跳转到label | | **9** | bltz | bltz $t1, label | 如果$t1 < 0, 则跳转到label | | **10** | li | li $t0, 1 | $t0 = 1 | | **11** | add | add $t0, $t1, $t2 | $t0 = $t1 + $t2 | | **12** | lw | lw $t0, 0($sp) | 将地址0($sp)中的值加载到$t0 | | **13** | sub | sub $t0, $t1, $t2 | $t0 = $t1 - $t2 | | **14** | la | la $t0, addr | 把地址addr加载到$t0 | | **15** | sll | sll $t0, 2 | $t0 = $t0 << 2 | | **16** | addi | addi $t0, $t1, 4 | $t0 = $t1 + 4 | | **17** | jr | jr $ra | 跳转到$ra存储的地址处 | | **18** | move | move $t0, $t1 | $t0 = $t1 | | **19** | mult | mult $t0, $t1 | (Hi, Lo) = $t0 \* $t1  Hi和Lo是存储乘除法指令使用的寄存器 | | **20** | mflo | mflo $t0 | $t0 = Lo | | **21** | div | div $t0, $t1 | Lo = $t0 / $t1  Hi = $t0 % $t1 | | **22** | sw | sw $ra, 0($sp) | 把$ra的内容存入地址0($sp) |   存储分配方案    关键代码列举  寄存器通过两个bool数组以及一个map存储, 用来模拟物理寄存器。    测试程序中有写语句，作用是字符串输出，但MIPS无法通过寄存器存储这些字符串。所以我将输出字符串以及全局变量、常量存储在内存中，需要的时候再写入相关寄存器。    生成的MIPS四元组用一个struct结构来保存。    内存中的数据保存以及输出：    mips目标代码，存放在.text文件中。  目标代码实现：  全局变量和常量声明代码如下：    写语句：    MIPS字符串：    表达式结果：    换行符：    处理输出值的表达式，得到表达式答案存储的寄存器。如果当前符号是左括号则直接压入符号栈；如果是右括号，再不断拿运算数栈的栈顶两个运算数和符号栈的栈顶符号进行相应运算，并将结果放置数值栈的栈顶，直到遇到左括号。      然后需要定义运算符优先级。    MIPS数值运算：  Eg. add      mul:    div:    读语句:    赋值语句：  处理赋值语句时，首先通过map::reg获取被赋值变量的寄存器，获得赋值表达式答案寄存器，再将表达式答案寄存器答案赋值给被赋值变量的寄存器。 | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程  平台运行结果： | | | | | | |
| 五、实验总结  迄今为止最难的个人项目。尽管不算注释和废代码，最终长度大概也有几千行了吧。这一个月下来，我大致理解了编译器工作的大致流程，同时也精进了C++技巧，可谓收获颇丰。  一些不足之处：   1. 代码管理问题，代码一场就管理混乱，而人的内存又非常小，这导致经常需要切换上下文，切换时间非常长，痛苦至极。 2. 可读性特别差，由于代码管理混乱再加命名，知识点本就掌握不熟悉等问题，造成代码可读性差。这一点在debug时非常明显。还有函数名、类名取得也不好，什么Block、temp、Node名字随便用，各个函数关系比较散。   总体来看虽然写起来很费劲，但其实也只是完成了基础的功能，还是在有原理指导的情况下。各大语言背后的编译器得复杂成啥样。。之前学习JVM地时候还以为自己已经学了很多东西，现在看来连门都没有入，不过皮毛而已。 | | | | | | |