**《编译原理》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **姓名** | | **龙俊桦** | | | **年级** | **2017** |
| **学号** | | **20174302** | | | **专业、班级** | **计算机科学与技术05班** |
| **实验名称** | **目标代码生成程序的设计与实现** | | | | | |
| **实验时间** | **2020.11.19** | | **实验地点** | **A主410** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 □设计性 □综合性** | | |
| 教师评价：  完成实验内容，达到实验要求，实验数据和结果正确，报告内容详实。  程序质量得分：7，实验报告得分：2，实验项目总得分：  评价教师签名：张敏 | | | | | | |
| 一、实验目的  在词法分析、语法分析及语义分析的接触上，编写一个程序对使用类C语言书写的源代码翻译为中间代码/目标代码，并打印翻译结果 | | | | | | |
| 二、实验项目内容  一）实验内容  设计并实现实验一中的C语言子集的目标代码生成程序，并打印分析结果。要求实现以下功能：  1、必做项：   1. 能够输出抽象语法树/四元式的中间代码，应至少包括以下代码类型：赋值语句、算术运算操作（加减乘除），跳转语句、分支与循环语句及其他基本语 2. 在四元式基础上，生成汇编语言形式的目标代码   2、可选项：   1. 生成过程/函数的目标代码   二）实验要求：   1. 自由选用程序设计语言（java，python，c/c++）作为实现语言，手工编写语法分析程序。 2. 提交实验报告及源代码。实验报告需严格遵循学校文档规范，内容包含对应文法、语法分析测试用例。   三）C语言子集:  数据类型: **int**, 无符号整数, 取值范围0-9999  int a;  int a,b;  int a = 1;  算术运算符: **+**,**-**  a = b + 1;  a = b + c;  赋值运算符: **=**  a = 1;  关系运算符: **==** ,**>**,**<,<>,>=,<=**  a = (b==c);  a = (b>c);  a = (b<c);  逻辑运算符: **&&**, **||, !**  a = (b&&c);  a = (b||c);  a = (!b);  条件语句: **if**  if(a==b)  {  };  循环语句: **while**  while(a==b)  {  };  输入,输出: **get**,**put**  get(a);  put(a);  语句结束符: **;**  **加分项：**  条件语句 **if else**  if(a==b)  {  };  else  {  };  四）目标语言  在本次实验中假设数据类型只包含整数类型，不包含如浮点数、数组、结构和指针等其它数据类型， 目标语言选定为MIPS32指令序列，中间代码及MIPS32指令对应关系如下表所示。其中reg(x)表示变量x所分配的寄存器。 | | | | | | |
| 三、实验过程或算法（源程序）   1. 中间代码生成   上一次实验我们最后得到了一棵AST。在这个部分，我们将会遍历AST来获得对应的四元式。  我们使用一个结构体来表示四元式，如下所示：    然后对于AST不同的节点编写对应的处理函数，例如对于INT节点的处理如下：    具体代码见generateMiddle.cpp。  在中间代码生成阶段，最为复杂的便是对于if-else语句以及while语句的处理，因为在生成代码的过程中，你并不知道你要goto的位置，所以这里我选择把if while语句的位置以及还没有处理的部分(if的body部分和while的body部分)使用一个队列记录下来，然后在处理完了所有的程序后，对两个队列进行处理。    例如对于IF节点的处理程序如上图所示：  goto\_indexs用来记录if语句的位置，方便后续处理时goto回来。  tobe\_process用来记录if body的节点，在所有程序分析完了后进行处理。    上图是所有程序处理完了后，对于if body部分的处理，这个时候通过获取goto\_indexs里的项，就可以知道goto的位置了。  具体代码见generateMiddle.cpp。   1. 目标代码生成   中间代码生成完毕后，结果存在一个vector<Quad\*>里，如果要生成目标mips代码，只需要对中间代码做一个简单的翻译就行了。  具体代码如下所示：  //generateTraget  vector<string> generateTarget(vector<Quad\*> quads)  {      vector<string> res;      int len = quads.size();      string s= "";      for(int i=0;i<len;i++){          s = "";          switch(quads[i]->type){              case LABEL:                  s = quads[i]->ans;                  break;              case QASSIGNV:                  s = "move "+regWrapper(quads[i]->ans)+","+regWrapper(quads[i]->op1);                  break;              case QASSIGNC:                  s = "li "+regWrapper(quads[i]->ans)+","+quads[i]->op1;                  break;              case QADD:              case QSUB:              case QBITAND:              case QBITOR:                  s = string\_quad[quads[i]->type]+" "+regWrapper(quads[i]->ans)+",";                  if(isNumber(quads[i]->op1))                      s += quads[i]->op1;                  else                      s += regWrapper(quads[i]->op1);                  s += ",";                  if(isNumber(quads[i]->op2))                      s += quads[i]->op2;                  else                      s += regWrapper(quads[i]->op2);                  break;              case QGOTO:                  s = "j "+quads[i]->ans;                  break;              case QEND:                  s = "end";                  break;              case QIFEQUAL:              case QIFNOTEQUAL:              case QIFGT:              case QIFGTOREQUAL:              case QIFLESS:              case QIFLESSOREQUAL:                  s =  string\_quad[quads[i]->type]+" ";                  if(isNumber(quads[i]->op1))                      s += quads[i]->op1;                  else                      s += regWrapper(quads[i]->op1);                  s += ",";                  if(isNumber(quads[i]->op2))                      s += quads[i]->op2;                  else                      s += regWrapper(quads[i]->op2);                  s += ",";                  s += quads[i]->ans;                  break;              case QPUT:              case QGET:                  s = string\_quad[quads[i]->type];                  break;              default:                  cout<<"unexpected"<<endl;          }          res.push\_back(s);      }      return res;  } | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程   1. 自定义测试样例: 2. 简单if-else语句   这里展示一个if-else语句代码生成的例子来具体说明。  源代码如下所示：    先经过中间代码生成得到四元式如下：    然后经过目标代码生成得到结果如下：    可以看到，0~3进行了a的定义和赋值，第4行判断a是否为1。  如果a为1，跳转到第9行，把a+1赋给a后，跳转到第八行结束。  如果a不为1，就进入第五行，把a-1赋给a后结束。  所以可以看到，上面的代码生成是正确无误的。   1. while嵌套if语句   这里展示一个while嵌套if语句代码生成的例子来具体说明。  源代码如下所示：    先经过中间代码生成得到四元式如下：    再经过目标代码生成得到结果如下：    可以看到，0~3行在进行a的初始化，第四行判断a是否<=99  如果不是，结束。  如果是，跳转到第6行，判断a是否小于50。  如果a<50，跳转到11行，11~13行对a加2，然后14行执行，跳转到10行，10行又跳转到4行，再次进行while循环的判断。  如果a不是<50，那么7~9行对a加1，第10行执行，跳转到第4行，再次进行while循环的判断。  所以可以看到，上面的代码生成是正确无误的。   1. 必选测试样例 2. 简易计算器   由于之前分析得到，这里的程序存在ans可能没有值的语义错误，所以我们稍微增加一行，确保ans是有值的。  //输入数据num1,num2,op，根据op确定操作进行运算，最后输出运算结果ans  int num1,num2,op,ans;  get(num1,num2,op);  ans = 1; //新加的  if(op==0)  {  ans = num1 + num2;  };  if(op==1)  {  ans = num1 - num2;  };  if(op==2)  {  ans = num1 & num2;  };  if(op==3)  {  ans = num1 | num2;  };  put(ans);  对该源代码进行中间代码生成，结果如下：    进行目标代码生成，结果如下：    经过分析，该中间代码是正确无误的。   1. 跑马灯   //循环输入op，改变输出结果out，输入0则结束程序  int num0,num1,out,op;  num1 = 3333;  num2 = 6666;  num3 = 9999;  op = 1;  while(op>0)  {  if(op==1)  {  out = num1;  };  if(op==2)  {  out = num2;  };  if(op==2)  {  out = num3;  };  put(out);  get(op);  };  对该源代码进行中间代码生成，结果如下：    进行目标代码生成，结果如下：    经过分析，该中间代码是正确无误的。 | | | | | | |
| 五、实验总结  这一次实验主要就是对之前生成的AST进行一个简单的翻译，唯一的难点就是if-else语句以及while语句的处理，具体的处理方法在上文已经说过了。  其中，中间代码和目标代码生成部分编写的代码都在generateMiddle.cpp，main函数在generateMiddleTest.cpp。  经过这次实验，我了解了中间代码和目标代码生成的过程。经过四次实验，我最终将源代码经过词法分析、语法分析、语义分析、代码生成，得到了mips代码，收获很大。 | | | | | | |