实验四 目标代码生成

姓名： 张子谦
学号： 151220166
邮箱： zhangziqian1011@126.com

一、完成功能

1. 完成所有必做内容，即函数参数小于 4 个时的处理；
2. 用的是朴素的寄存器分配算法，因此代码较为冗长，但指令选择时稍作优化，因此也不算过于冗长。

二、实现方法

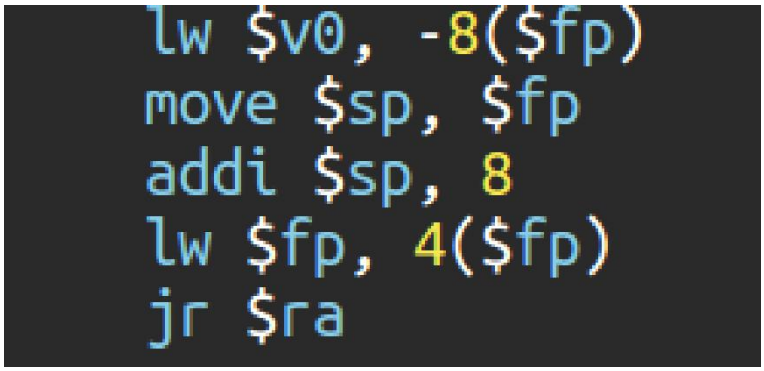
1. 指令选择和寄存器分配: 对于每条中间代码逐句生成目标代码，由于寄存器分配采用了朴素的算法，效率太低，因此将两者融合，对于每个类型的中间代码，只是将该读入寄存器的变量读入，该写回的写回，减少了目标代码的个数。逐条翻译的规则并没有完全照抄讲义上的，鉴于每条语句读入和写回相关变量，用了很多 lw 和 sw 来代替原本的 move 等指令。除此之外，还顺便实现了 x=&y 的翻译，使得 lab3 中以结构体，数组为参数的代码也可以进行有效翻译。x=&y 的翻译的对应指令:

```
case ASSIGN_ADDR:
    l=code->assign.left;
    r=code->assign.right;
    fprintf(file, "\tmove $t0, $fp\n");
    int rb=getBase(r,vBases);
    fprintf(file, "\taddi $t0, -%d\n",rb);
    storeVariable(l,vBases,0,file);
    break;
```

2. 栈管理: 用$fp 和$sp 对于栈进行联合管理。由于 lab3 的变量标号因为种种原因不是连续的，所以在每个函数体得到中间代码之后立即对于变量重新标号，使得它们的标号从 0 开始紧密的逐渐增长这样一来就可以轻松的计算出每个变量相对于$fp 的偏移量。在分配栈帧空间时，fp 和 sp 之间靠近 fp 一侧排列着 v0 到 vn,而靠近 sp 一侧排列着 t0 到 tm$fp 和$sp 的处理方式和 gcc 中的%ebp,%esp 类似：
函数体开始：

```
sw $fp, -4($sp)
addi $sp, -8
move $fp $sp
addi $sp, -12
```

函数体结束：



三、运行方式

1. 在 code 文件夹运行 make 可以在项目目录中产生可执行文件 parser，./parser testfile outfile 可以对 testfile 进行目标代码生成并在 outfile 内输出结果。
2. make test 可以测试 testcase/文件夹下面的所有.cmm 文件并输出到对应的.s 文件中。
3. make clean 清除所有 make 生成的文件。