实验四 目标代码生成

组长：褚有刚（学号：171860526）

组员：夏宇（学号：171180554）

一、实验目的

在词法分析，语法分析，语义分析和中间代码生成程序的基础上，将`C--`源代码翻译为`MIPS32`指令序列（包含伪指令），并在`SPIM Simulator`上运行，实现一个可以实际运行的编译器；

二、实验流程

1. 数据结构设计

- 寄存器
  为了简单起见，仅使用`mips`提供的`t0 -> t9`寄存器作为通用寄存器，使用`\$v0`作为返回值，使用`\$ra`作为当前函数的返回值
  每个寄存器的结构如下
  - 寄存器名：输出目标代码的时候使用
  - 使用标志位
  - 是否被修改：减少`STORE`的次数
  - 加锁标志：防止寄存器的内容之间发生覆盖
  - 寄存器中的局部变量
- 局部变量列表
  为了简单起见，栈空间管理采用和`x86`类似的结构，通过`\$fp`寄存器来访问栈中的数据，
  因此每个局部变量除了需要保存变量本身的信息之外，只需要保存一个相对`\$fp`的偏移量即可
  对于函数的参数，偏移量为正数，并且第`k`个函数参数的内存地址为`\$fp + 8 + 4 * k`，对于函数内部的变量，偏移量为负数

2. 翻译模式

- 固定代码的翻译
  `.data`； `.globl`； `.text`中`read`和`write`函数的翻译直接向文件输出内容即可；
- 代码段的翻译
  - 每条语句的翻译，基本上按照 取数 -> 计算 -> 存数 的固定模式进行翻译
  - 在基本块的出口处，将被修改的变量全部写回内存
  - 对函数的入口和出口的翻译使用`x86`的模式，在入口处`push %ebp, mov %ebp, %esp`，在出口处`mov %ebp, %esp，pop %ebp`，一点不同是`\$fp + 4`中存储的是被调用者的`\$ra`，当前函数的`\$ra`在寄存器中

3. `alloc_reg`和`get_reg`函数的设计

- `alloc_reg`
  `alloc_reg`函数的作用是为变量分配寄存器，如果变量是第一次出现还会为变量分配存储空间
- `get_reg`
  `get_reg`函数的作用将变量`LOAD`到寄存器中并返回，如果变量就在寄存器中则直接返回，否则调用`alloc_reg`为变量分配寄存器，分配结束之后保证变量都有了存储空间，然后再将变量`LOAD`到该寄存器中
- 设计的目的
  - 对于对变量的写操作不需要将内存中原来的数据再加载到寄存器中，可以直接调用`alloc_reg`函数获取该变量的内存地址，然后向其中写入，如果变量之前就存在则会覆盖，如果变量是第一次使用则完成了初始化；
  - 对于变量的读操作由于必须要从内存中读取相应的内容，因此不需要做改变

三、遇到的问题

对于`x := y / z`的翻译模式，刚开始的时候使用了`div reg(x), reg(y), reg(z)`的翻译模式，后来发现实验手册上给的是`div reg(y), reg(z)`，虽然自己测试也没有问题，但是最后还是改成了和实验手册上相同的模式