编译原理第四次实验

实验报告

171860691 王辛有 171860005 宋斯涵

一、 实现功能

完成了从三地址代码到汇编代码的转换。
其中完成的具体细节为：

1） 指令选择
2） 寄存器选择
3） 栈管理

二、 重要的数据结构说明

表示寄存器结构（每个域的含义见注释）

```
struct Register
{
    char name[32];//寄存器的名字（例如：$v0）
    int free;//该寄存器当前是否为空闲
    Var variables;//该寄存器中储存的变量
}regs[32];
```

表示变量的数据结构（每个域的含义见注释）

```
//链表结构
typedef struct Var_
{
    char name[32];//变量的名称(由于每个变量都不相同，所以变量采用名等价)
    int reg;//该变量的当前值储存在哪个寄存器
    int offset;//该变量在栈中的位置（相对于栈顶指针的偏移量）
    Var next;//指向下一个节点
}Var_;
```

三、 实现方法

指令选择

通过遍历中间代码链表，根据中间代码的类型，按照实验指导书 P100 的内容来完成指令的选择，整体框架是一个 while 循环，来遍历每一条中间代码，并将汇编代码输出到结果文件中。

寄存器选择

寄存器结构的定义如第二部分中所述，为了确保程序的正确性和可调试性，我们采用了朴素寄存器分配的方法，首先将所有的变量或临时变量放到内存中，在每翻译一条中间代码时选出一个空闲寄存器，将需要用到的寄存器加载到变量中，得到计算结果后又将结果写回内存，并重新将寄存器标记为空

闲。
在这里的一个难点是需要将所有变量放在内存中，我们选择了在刚进入一个函数时，就先遍历整个函数，然后将所有的变量都加载到内存中，之后按照相对于$fp 的偏移量来寻址。

栈管理

参照实验参考书上的关于调用过程中栈管理的办法进行处理。
和参考书中稍有不同的有两点：

1. 无需进行关于调用者保存寄存器和被调用者保存寄存器的入栈和恢复
   原因：由于本实验中我们采用的是**朴素寄存器分配法**，故每处理完一条指令都会把相应的值写到内存中，然后将相应的寄存器设置为 free。鉴于一条指令使用的寄存器不会那么多，所以我们选择只应用 8-15 号寄存器，且每处理完一条语句就进行一次寄存器的恢复，故不需要在函数调用时进行整体的寄存器的保存和恢复。
2. 没有使用存放被调用者参数的寄存器 a0-a3。
   原因：在传参时，对于每一个参数，我们都采用了压栈的方法对其进行处理。在被调用者接收形参时，根据 PARAM 指令的个数，依次对参数进行寻址。这样处理不需要对参数的个数进行计数，相对简单一些。

四、 编译指令

1. bison -d -v syntax.y
2. flex lexical.l
3. gcc main.c SymbolTable.c SemanticAnalysis.c HelpFunc.c tree.c syntax.tab.c Intercode.c ObjectCode.c -lfl -ly -o parser