编译原理第四次实验报告

161220115 汤佳铭 161220179 周科

实验目标

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--源代码翻译成MIP32指令序列，并在SPIM Simulator上运行

实验思路

数据结构

寄存器描述符

```
struct Reg_Obj_ {
    char name[8]; //寄存器名字
    int reg_index; //寄存器索引
    bool is_free;  //寄存器空闲状态
    Operand place; //寄存器保存的操作数
} Reg_Arr[32];//32个寄存器
```

栈中存在的变量维护

```
struct Operand_list_{
    Operand op; //变量操作数
    int offset;// 变量相对FP的偏移量
    Operand_list next;
    Operand_list prev; //利用双向链表来维护
};
Operand_list OpHead,OpTail;
```

实验三改动

删除了一些激进的优化，对于实验三来说，*x作为一个操作数是合法的，这也衍生了*x = *y等等一系列中间代码，这些对于实验三来说是很好的优化，但是对于实验四来说却大大增加的翻译的难度，所以我们删除了这些优化，以方便实验四的中间代码翻译。

具体调整思路如下：若*x在中间代码里面出现，且为右值时，增加一条t = *x指令，并用t作为后面使用的替代品

实验接口

```
void Init_Reg_Arr(); //初始化寄存器
void Init_ObjCode(); //初始化operand_list链表
Operand_list Search_Operand_list(Operand key); //搜索栈中是否出现过此变量
void Insert_Operand(Operand key, int offset);//将此变量插入栈
void Traversal_InterCodes(FILE *fptr);//遍历中间代码链表
void Traversal_InterCode(InterCode i,FILE *fptr);//对单个中间代码进行处理
int Allocate_Reg(Operand place, bool flag,FILE *fptr); //对每个变量分配寄存器
void Free_Reg(FILE *fptr);
bool Match_Operand(Operand t1, Operand t2);//判断两个操作数是否相等
```

寄存器分配算法

我们采用的是朴素寄存器分配算法，当Traversal_InterCode()接受到一条中间代码时，我们会根据它的类型来调用若干条Allocate_Reg来给其分配必要的寄存器，然后在打印完对应的MIP32指令之后调用一次Free_Reg释放所有寄存器

指令翻译

按照讲义上的规则所编译，额外处理见前述实验三改动。

过程调用部分

关于寻找函数形参实参的存储位置

原本构思的是每次独立地读入一句中间代码，把它生成为一句或者多句目标代码。直到出现arg、param类型的中间代码时，我们发现，此时的中间代码是无法割裂开单独来看的，单看一句代码我们是无法知道形参存在了哪或者是实参应该存在哪。解决方案是，每次检测到arg类型的中间代码且它的前节点非arg类型，我们设置它为一个新链表的头。同样的，每次检测到arg类型的中间代码且它的后节点非arg类型，我们设置它为此新链表的尾巴。这个新链表就是一个函数的所有arg，其长度为length，设置一个全局变量num，从i=1到length，依次取链表中的节点载入翻译，并把num设为I 。此时翻译函数翻译para类型中间代码时，根据全局变量num的数值就可以推算出它所被存储的位置（a0~a3或者所在内存的具体位置）。相同的，arg类型中间代码的翻译也是如此，只不过它的num需要length到1倒序输出罢了。

关于函数嵌套调用

每次调用函数前，需要把ra压栈，随后fp、sp改变为被调用者函数所用，函数调用结束后再返回，sp、fp的原值是会丢失的。本该是把值写进内存来解决的，但是当时没搞懂sp、fp的具体关系，后来偷懒想了种简单的解决方案，就是用闲置的t9、t10来分别存储sp、fp的旧值，调用结束后再恢复。在单个函数调用下是可性的，但是函数嵌套调用时出现了问题。分析后发现，当f调用g，g调用h情况下，f调用g时sp、fp被存入寄存器，g再调用h，又存了一次新的sp、fp进寄存器，此时f的sp、fp被覆盖掉，无法恢复，故出错。后来搞懂了sp、fp的具体关系，发现每次call一个新函数，新的fp指向旧的sp后便不再动了，相当于用fp存着sp的旧值，而fp的旧值就随着ra存入内存就好了，调用结束后再依次恢复sp、fp、ra即可。