编译原理实验-lab4

- 161220137 吴刚
- 161220133 温宗儒

数据结构

- 此次实验在实验三基础上进行，对中间代码生成的循环链表遍历，根据不同类型的中间代码生成不同类型的目标代码。保存的结构在targetCode.h文件中。

类型信息

- Register结构用于保存寄存器，程序中声明长度为32的register数组，用于保存32个寄存器。
- Var_t结构用于保存寄存器中变量的一些信息。

```
typedef struct Var_t_* Var_t;
struct Var_t_{
    char* name;//变量名
    int reg;
    int offset; //偏移量
    struct Var_t_* next;
};


typedef struct Register_* Register;
struct Register_{
    char* name; //寄存器名
    Var_t var;  //此寄存器中保存的数据
};
```

目标代码生成过程

指令选择

- 由于实验三以双向链表的形式生成中间代码，所以此次实验重新遍历中间代码链表，并将相应的中间代码根据指令表进行翻译，即可生成目标代码。

```
InterCode interCode = begin->next;  //interCode链表的开头无用，所以第二个为起始
while(interCode->kind != CODE_BEGIN){
    printTargetCode(interCode);//遍历每一个节点，根据类型生成相应的目标代码
    interCode = interCode->next;
}
```

寄存器分配

- 在头文件中我们定义了寄存器和变量的结构，当遇到变量时，通过getReg(Operand operand)函数获取相应的寄存器下标，如果在变量表中没有当前变量，说明为新变量，则创建该变量的信息并保存在变量链表中。

```
int getReg(Operand operand){
    char* name = NULL;
    ……

    Var_t var = findVar(name);
    ……
    if(var == NULL){
        ……//创建新的变量
        addVar(var);//加入变量链表中
    }else{
        ……
        lwReg(i,var);//读取当前变量信息
    }
    return i;
}
```

- 按照变量访问顺序，对t0~t7进行分配，当指令执行结束之后，利用swReg()函数，将寄存器中所存的变量放入内存中。
- 查询变量利用findVar(char* name)函数，遍历变量表，查找是否有同名变量。

栈管理

- 传参： 前四个参数利用寄存器a0~a3，多余的参数按顺序存入栈中，在函数中利用fp和偏移量寻址得到参数值。
- 活动记录： 函数调用前做准备，将fp和返回地址压入栈中，当函数结束进行return时，将栈帧返回到相应位置即可。