实验四 目标代码生成

姓名：肖丹妮　　学号：201220199　　专业：计算机科学与技术专业

一、实验环境与编译方法

实验环境与手册一致；利用提供的Makefile文件进行编译。

二、实验实现的功能

- 完成了实验要求即：能够将实验三中得到的中间代码经过指令选择、寄存器选择以及栈管理之后，转换为MIPS32汇编代码。

三、实验内容与核心代码

3.1 指令选择

由于在实验三生成中间代码选择的是线性IR，所以选择了最简单的指令选择方式：逐条将中间代码对应到目标代码上。

```
if (head != NULL)
{
    ICLnode *cur = head->next;
    while (cur != head)
    {
        switch (cur->code.kind)
        {
            case LAB:
                Generate_LAB(cur, fp);
                break;
            ……

            cur = cur->next;
        }
    }
}
```

3.2 寄存器分配

寄存器分配算法：

选择的是朴素寄存器分配算法。

数据结构

```
//变量描述符
typedef struct VarDesc *VarDescp;
typedef struct VarDesc{
    char name[5];
    int reg;//寄存器
    int offset;//在栈中的偏移量
```

```
    VarDescp next;//在栈中相邻的下一个变量
}VarDesc;
//寄存器描述符
typedef struct RegDesc *RegDescp;
typedef struct RegDesc{
    char name[5];//名字
    char alias[5];//别名
    int no;//编号
    int free;//是否空闲
    VarDescp var;//保存的变量
}RegDesc;
```

寄存器处理相关函数：

```
RegDescp New_RegDesc(int no);//初始化寄存器描述符
bool Init_Regs();//初始化32个寄存器的信息，名字、别名、编号、初始空闲、保存变量为NULL
int Get_RegForVar(FILE *fp, Operandp var);//返回变量所在寄存器，首先根据变量的名字在栈中搜索此变量，若返回结果为空，说明变量还未保存在栈中，因此用此变量的名字初始化一个变量描述符，令全局变量stackOffset减4，变量描述符的offset=stackOffset，并且为此变量分配一个寄存器；若返回结果不为空，则将此变量的值加载到寄存器中。
bool SW_Reg(FILE *fp, int regno);//将寄存器的值写回栈，这里是简化处理，所有溢出的变量都会保存到栈里面
bool LW_Reg(FILE *fp, int regno, VarDescp var);//将栈中变量加载到寄存器
bool Print_Reg(FILE *fp, int regno);//向文件中写入寄存器别名
```

3.3 栈管理

函数调用的数据流转移：

参数传递：

实参：用Argn记录当前处理的实参编号：an的编号是n-1，a1的编号是0。前四个参数存进a0、a1、a2、a3四个寄存器，后面则都存进栈中。
形参：用Paramn记录当前处理的形参编号：p1的编号是n-1，pn的编号是0。paramsumn记录全部的形参个数。最后四个形参存进的值是a0、a1、a2、a3四个寄存器中的值，前面的形参存进的值是((paramsumn-Paramn+1)*4)($fp)中的值。
这个地方容易出错，刚开始忽略了sw与lw的次序应该相反，即先存进栈里的值应该后取出来。

保存返回地址：

在调用函数之前，应该将$ra保存进栈，在调用完函数之后，将$ra从栈中取出来

```
fprintf(fp, "\tsubu $sp, $sp, 4\n");//保存返回地址
fprintf(fp, "\tsw $ra, 0($sp)\n");
if(cur->code.u.assign.left->kind==VARIABLE || cur->code.u.assign.left->kind==TEMPVAR)
    regx=Get_Reg(fp, cur->code.u.assign.left);
fprintf(fp, "\tjal ");//跳转到被调用函数处
PrintOprand(fp, cur->code.u.assign.right);fprintf(fp, "\n");
fprintf(fp, "\tmove ");//保存返回值
Print_Reg(fp, regx);
fprintf(fp, ", $v0");fprintf(fp, "\n");
SW_Reg(fp, regx);
fprintf(fp, "\tlw $ra, 0($sp)\n");//取返回地址
fprintf(fp, "\taddi $sp, $sp, 4\n");
```

数组与结构体：

```
bool Generate_DEC(ICLnodep cur, FILE* fp)
{
    VarDescp arrayHead = malloc(sizeof(VarDesc));
    stackOffset-=cur->code.u.dec.size;//在栈中申请数组size或者结构体size大小的空间
    arrayHead->offset=stackOffset;//数组或结构体首地址(偏移量)是低端
    strcpy(arrayHead->name, cur->code.u.oneop.op->name);
    Push_Var(arrayHead);//将数组或结构体描述符压进栈中
    fprintf(fp, "\tsubu $s0, $fp, %d\n", -stackOffset);
    fprintf(fp, "\tsw $s0, %d($fp)\n", arrayHead->offset);
    return true;
}
```

四、实验感悟与心得

写实验四的过程中也发现了实验三的错误，实验三中处理函数实参时，只用了一个记录实参的数组，于是当函数调用的参数也是函数调用时就会出现参数混乱的情况，解决办法是将记录实参的数组变成二维数组，每个函数调用都会有单独的实参数组。