编译原理实验报告

实验四：目标代码生成

191220008 陈南曈

924690736@qq.com

一、运行方式

```
$ make
$ ../parser inputfile outputfile
$ make clean
```

二、实现功能

任务编号：11

（一）目标代码生成

1、基本数据结构

本实验中，使用了变量描述符，寄存器描述符，和用于存放变量描述符表的栈

```
struct VarDesc_ {
    Operand op;
    int regIndex;
    int offset;
    VarDesc next;
};
struct RegDesc_ {
    char* name;
    VarDesc var;
    int age;
};
struct VarListStack_ {
    int offset;
    VarDesc varList;
    VarListStack prev;
    VarListStack next;
};
```

2、指令选择

大部分使用了与手册上相同的指令选择，但难点在于有关地址存取的指令选择，

3、寄存器分配

在本实验中，我将8 ~ 25号寄存器全部当做任意使用的寄存器，单号给寄存器分配满时，我使用了一个类似前进先出的寄存器替换算法，我将每个寄存器的数据结构中设置一个 age， 初始值为1，每当向某个寄存器存入内容时，我就将其他所有寄存器的 age 加一，当 age 达到 18 时，我就将该寄存器中的内容溢出到内存中，并将当前内容存入到该寄存器中，然后将该寄存器的 age 置为 1。这样我就能同时使用18个寄存器，提高变量存取的效率。

```
struct RegDesc_ {
    char* name;
    VarDesc var;
    int age;
};
```

4、临时寄存器$v1

在手册上提到了，由于函数的返回值只会返回一个值，存在寄存器`$v0`中，因此`$v1`可以用作其他用途，而我将该寄存器用作临时保存地址的寄存器，由于只有这一个用途，因此我不需要考虑寄存器是否需要清空或溢出。

```
// x := &y + z
fprintf(file, "\taddi $v1, $fp, %d\n", regList[op2RegIndex]->var->offset);
fprintf(file, "\tadd %s, $v1, %s\n", regList[op1RegIndex]->name,
regList[op3RegIndex]->name);
```

例如该中间代码，我就会将y的地址暂时存于`$v1`中，然后用于地址计算。

5、栈管理

本实验中栈管理十分重要，我通过在变量描述符中存入offset，在变量描述符表的栈中存入当前函数的offset，以此来辅助栈的寻址，存取等操作。

三、心得和建议

实验四难度较上一次实验有所提升，原因在于自己对汇编指令不够熟悉，在指令选择和寄存器的分配上都遇到了一些困难，但通过仔细地思考和画图梳理，还是能理清楚。最大的难点在于栈的管理，地址很有可能会计算错，如何管理offset，何时存取变量都有一定的讲究。

总之在该实验后，我对寄存器和栈管理相关的只是有了更深的理解，也弄清了一些以前没太理解的内容，有所收获。