《编译原理》Lab4实验报告

姓名：李诗宇

学号：3210100999

实验目的

本次实验的目标是从实验3生成的中间表示出发，生成 RISC-V 32 的汇编代码。

实验亮点

1.指令翻译

中间代码与汇编代码格式不一样，我们对每一种中间代码采用一个特定的模式，翻译成对应的目标代码。最直接的方式就是将每一条中间代码翻译成一条或多条目标代码：

中间代码	目标代码	中间代码	目标代码
a = b + c	add reg(a), reg(b), reg(c)	a = b - c	sub reg(a), reg(b), reg(c)
a = b + #t	addi reg(a), reg(b), t	a = b - #t	addi reg(a), reg(b), -t
a = b	mv reg(a), reg(b)	a = #t	li reg(a), t
LABEL label:	label:	GOTO label	j label
a = CALL f	jal f ; move reg(x), a0	RETURN a	mv a0, reg(a) ; ret
x = *y	lw reg(x), 0(reg(y))	*x = y	sw reg(y), 0(reg(x))
IF x > y GOTO label	bgt reg(x), reg(y), label	IF x <= y GOTO label	ble reg(x), reg(y), label
x = &y	la reg(x), y	GLOBAL x:	x:
.WORD #k	.word k	FUNCTION f:	f:

我采用了c++流运算符的一些性质，简化了翻译过程（本质上是一个字符串处理程序）：

```
std::vector<std::string> translateToRISC(const std::vector<std::string>& intermediateCode) {
    std::vector<std::string> riscvCode; //由于生成的RISC-V分为.text和.data两段，我将分析中间代码的结果分别存在riscvCode与dataCode
    std::vector<std::string> dataCode;  //分析结束时再合并在一起
    ...

    for (const auto& line : intermediateCode) {
        std::istringstream iss(line);
        std::string token;
        iss >> token;

        if (token == "GLOBAL") {
            std::string var;
            iss >> var;
            dataCode.push_back(".data");
            dataCode.push_back(var);
        } else if (token == ".WORD") {
            std::string value;
            iss >> value;
            dataCode.push_back(".word " + value.substr(1));
        } else if (token == "FUNCTION") {
            std::string func;
```

```
            iss >> func;
            ...
        }

    riscvCode.insert(riscvCode.end(), dataCode.begin(), dataCode.end());
    return riscvCode;
}
```

2.寄存器分配

我采用实验指导中的朴素寄存器分配方法，把所有临时变量都存储在内存中，也就是栈上。 每翻译一条中间代码之前我们把要用到的变量先加载到寄存器中，得到计算结果后又将结果写回内存。

我对寄存器在内存专门分配了一段空间，栈顶地址存在`gp`中:

```
_minilib_start:
    la sp, _stack_top
    mv gp, sp
    lui, t3, 0x100
    sub, t3, x0, t3
    add gp, gp, t3
    call main
```

然后，对于局部变量`tx`，我将其存在离栈顶偏移量为`4*x`的位置（`sw tx, 4*x(gp)`）。我采用`map<std::string, int> stackOffset`将局部变量的名字map到栈的偏移量：

```
if (stackOffset.find(lhs) == stackOffset.end() && lhs[0] != '*') {  //每当发现一个新的局部变量被定义
                std::string temp;
                temp = lhs.substr(1);
                stackOffset[lhs] = 4 * stoi(temp);
                tempqueue.push_back(stoi(temp));
            }

            if (rhs.find('+') != std::string::npos) {   //对add指令的翻译
                std::string eq, a, b;
                std::istringstream rhsIss(rhs);
                rhsIss >> eq >> a >> token >> b;
                riscvCode.push_back("lw t0, " + std::to_string(stackOffset[a]) + "(gp)");
                if (b[0] == '#') {      //addi指令
                    riscvCode.push_back("addi t1, t0, " + b.substr(1));
                } else {
                    riscvCode.push_back("lw t1, " + std::to_string(stackOffset[b]) + "(gp)");
                    riscvCode.push_back("add t1, t0, t1");
                }
                riscvCode.push_back("sw t1, " + std::to_string(stackOffset[lhs]) + "(gp)");
            }
```

3.栈的管理

函数调用中栈的管理是本实验的核心部分。

ARG与PARAM的翻译

我们在CALL一个函数前要将其参数放入参数寄存器，在中间代码中，我们用ARG简化了这一操作。具体而言，我们要将将实参传递到 a0-a7 这几个寄存器中，读取参数的时候再从这几个寄存器中取出即可。 当函数的参数个数超过调用规范中规定的个数时，我们则需要将多余的参数保存到栈上：

```
else if (token == "ARG") {  //我用全局变量argcounter来记录本次函数调用执行到了第几个参数
            std::string var;
            iss >> var;
            if(argcounter < 8)
                riscvCode.push_back("lw a"+ std::to_string(argcounter++) + ", " +
std::to_string(stackOffset[var]) + "(gp)");
            else{
                riscvCode.push_back("lw t0, " + std::to_string(stackOffset[var]) + "(gp)");
                riscvCode.push_back("sw t0, 0(s2)");    //s2为当前运行栈顶指针
                riscvCode.push_back("addi s2,s2,4");
            }
```

我们在运行一个函数前要将其参数从参数寄存器取出并压入栈中，在中间代码中，我们用PARAM简化了这一操作。

```
else if (token == "PARAM") {
            std::string var;
            iss >> var;
            paramsOffset[var] = paramscounter++;////我用全局变量paramscounter来记录本次函数压参入栈执行到了第几个参数
            if(paramscounter <= 8)
                riscvCode.push_back("sw a" + std::to_string(paramsOffset[var]) + ", " +
std::to_string(4*paramsOffset[var]+8) + "(sp)");
            else{
                riscvCode.push_back("lw t0, 0(s2)");
                riscvCode.push_back("addi s2,s2,-4");
                riscvCode.push_back("sw t0, " + std::to_string(4*paramsOffset[var]+8) + "(sp)");
        }
```

对于 CALL ，我们要将当前函数在CALL之前所有的临时变量对应的寄存器全部压入栈中，我采用了`std::vector<int> tempqueue`来记录当前函数在CALL之前所有的临时变量的序号：

```
if (stackOffset.find(lhs) == stackOffset.end() && lhs[0] != '*') {
                std::string temp;
                temp = lhs.substr(1);
                stackOffset[lhs] = 4 * stoi(temp);
                tempqueue.push_back(stoi(temp));//每发现一个临时变量，将其记录
            }
```

```
else if (rhs.find('C') != std::string::npos){
                std::string func, call;
                std::istringstream rhsIss(rhs);
                rhsIss >> call >> call >> func;
                if(func != "read" && func != "write"){ //read和write做优化，不需要压栈
                    for(int i = 0; i < tempqueue.size(); i++){
                        riscvCode.push_back("lw t2, " + std::to_string(4*tempqueue[i]) + "(gp)");
                        riscvCode.push_back("sw t2, 0(s2) ");//压栈
                        riscvCode.push_back("addi s2, s2, 4");
                    }
                    riscvCode.push_back("jal " + func);
                    for(int i = tempqueue.size() - 1; i > 0; i--){
                        riscvCode.push_back("addi s2, s2, -4");
                        riscvCode.push_back("lw t2, 0(s2) ");
                        riscvCode.push_back("sw t2, " + std::to_string(4*tempqueue[i]) + "(gp)");//函数调用返回
后弹栈恢复
                    }
                }
                else riscvCode.push_back("jal " + func);
                riscvCode.push_back("sw a0, " + std::to_string(stackOffset[lhs]) + "(gp)");
                argcounter = 0;
            }
```

对于数组变量，全局数组直接逐字翻译，局部数组的声明DEC，我们则需在栈中给该变量开辟空间：

```
else if (token == "DEC") {
            std::string var, num;
            iss >> var >> num;
            if (stackOffset.find(var) == stackOffset.end() && var[0] != '*') {
                std::string temp;
                temp = var.substr(1);
                stackOffset[var] = 4 * stoi(temp);
                reg_cn = stoi(temp);
                tempqueue.push_back(stoi(temp));
            }
            riscvCode.push_back("sw s2, " + std::to_string(stackOffset[var]) + "(gp)");//将当前栈指针作为数组变量
的首地址
            riscvCode.push_back("addi s2, s2, " + std::to_string(stoi(num.substr(1))));//按照数组size更新栈指针
        }
```

其他

编译

在提交的zip文件中包含了`Makefile`文件，只需进入Makefile所在的目录执行：

```
make
```

`src`文件夹中的源代码即可完成编译，生成执行文件`compiler`，即可进行测试。