编译原理实验四实验报告

191220163 计算机科学与技术系 张木子苗

完成的实验要求

在本次实验中，我完成了**全部的实验任务**，代码**可以通过手册提供的测试样例。**

提供的测试方法

在Test目录下，我一共准备了2个代码文件用于测试：

`test1.cmm` 和 `test2.cmm` ，对应实验指导文件里提供的2个测试用例。

为了方便测试，我为我的代码写了个简单的脚本文件，`Code/Test.sh`，在**Code目录下命令行界面输入**：

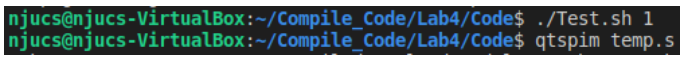
```
./Test.sh n
```

即可自动编译并且执行对源文件 `testn.cmm` 的目标代码生成，并且将结果写入到文件 `temp.s` 中。注意，**文件 `temp.s` 在 `Code` 目录下!**

如果需要测试其它用例，**请将新的测试用例放在Test目录下，命名格式为 testn.cmm（n不要为1-2）。在Code目录下命令行界面输入**：

```
./Test.sh n
```

即可自动编译并且执行对该测试用例的目标代码生成，并且输出到文件中。



代码的实现思路

数据结构

构造两个结构体：

```
struct RegisterDescriptor_
{
    int free;//寄存器是否空闲
    char name[20];
};
struct VaribleDescriptor_
{
    int reg_num;//变量所在的寄存器编号
    Operand op;
    VaribleDescriptor next;
};
```

分别作为寄存器描述符和变量描述符，用于寄存器分配。

实现思路

此次试验的核心在于函数调用时参数的传递。主要涉及的中间代码有：`IR_FUNCTION`，`IR_CALL`，`IR_RETURN`

对于其他的中间代码，我们只需要用用 `allocate` 和 `ensure` 两个函数（思路同手册上实现）进行寄存器分配，再直接根据手册上的对应关系翻译成目标代码即可。

而函数调用时参数的传递和寄存器值的保存与恢复，核心在以下4个函数：

```
//调用者在调用函数之前,保存a0-a3,t0-t10
void push_regs_saved_by_caller(FILE* fp);
//调用者在调用函数之后,恢复a0-a3,t0-t10
void pop_regs_saved_by_caller(FILE* fp);
//被调用者在进入函数时,保存s0-s7
void push_regs_saved_by_callee(FILE* fp);
//被调用者在return之前,恢复s0-s7
void pop_regs_saved_by_callee(FILE* fp);
```

IR_CALL的实现

已知：$\$a0-\$a3$ 用于储存前4个函数参数，$\$t0-\$t9$ 为调用者保存，$\$s0-\$s7$ 为被调用者保存。

为了防止寄存器的值被覆盖，所以我们每次进行函数调用之前，都利用函数 `push_regs_saved_by_caller` 将$\$a0-\$a3$，$\$t0-\$t9$ 压栈保存。之后开始传递参数，如果参数数量小于4个，则直接保存在 $\$a0-\$a3$ 中，否则还需要将参数压栈。

参数压栈后，利用指令 `move $fp, $sp` 将当前 $\$sp$ 保存到 $\$fp$，方便被调用者在获取参数时，根据和 $\$fp$ 的偏移量找到第5个之后的参数。

在调用之前，还需要用以下语句将当前 $\$ra$ 压栈，再进行函数调用：

```
addi $sp, $sp, -4
sw $ra, 0($sp)
jal function
```

调用返回后，先恢复 $\$ra$ 的值；如果参数个数大于4个，还需要 pop 参数，恢复栈指针。

```
lw $ra, 0($sp)
addi $sp, $sp, 4
addi $sp, $sp, 4*(param_num - 4)
```

完成了以上处理后，利用函数 `pop_regs_saved_by_caller` 恢复寄存器的值，

最后用语句 `move return_reg, $v0`，接收返回值。

IR_FUNCTION的实现

每次执行 IR_FUNCTION 代码，代表调用了新的函数，我们先调用 `push_regs_saved_by_callee`，保存$\$s0-\$s7$。然后清空变量描述符表，并释放所有已经被保存了的寄存器（$\$s0-\$s7$，$\$t0-\$t9$）

```
for(int i = 8; i < 26; i++)
    Reg_List[i].free = 1;
cur_varible_list = NULL;
```

之后从 $\$a0 - \$a3$ 处获取参数、分配寄存器，并且将信息插入变量描述符表。如果参数数量大于4个，我们还需要根据 $\$fp$，从栈中获取参数。

IR_RETURN的实现

return 时，先将返回值存入 $\$v0$，再调用 `pop_regs_saved_by_callee` 恢复 $\$s0 - \$s7$，之后直接用 `jr $ra` 返回。

```
move $v0, return_reg
pop_regs_saved_by_callee(fp);
jr $ra
```