编译原理实验4报告

姓名：张旭钦

学号：201240058

在实验4代码生成中，我使用的是朴素寄存器分配方案，指令执行前会将操作数从内存加载到寄存器，该指令执行完会将执行结果写回内存器。该方案只需要使用最多三个寄存器，因此实现简单，当然执行效率也比较低下。

（1）操作数在内存中的地址

每一个函数在处理时（遇到 FUNCTION 指令），我都会为其中的每一个变量绑定一个相对于fp的偏移量，并将栈帧减去record_size。这个偏移量其实就是一个简单的hash，将op映射到对应的偏移量。

这样子我只需要通过如下函数即可查询变量相对于栈帧的偏移量：

```
/// @brief 获得一个变量的栈帧偏移（在此之前已经alloc_offset）
int get_offset(Operand op) {
    if (op == NULL) return -inf;
    int off = -inf;
    if (op->kind == OP_VARIABLE) {
        off = offsets[op->u.var_no];
    } else {
        off = offsets[op->u.tmp.tmp_no + var_count];
    }
    Panic_ON(off == -inf, "Operand should have been allocated a stack
offset.");
    return off;
}
```

（2）函数调用声明

处理完操作数在内存中的偏移量，比较麻烦的部分就是callee和caller之间的归约了。

为了简单处理，规定：

$fp存储旧的$fp;
$fp + 4存储返回地址;
$fp + 8开始为参数;
$fp - 4为第一个局部变量，以此类推……

因此我对于PARAM指令的处理方式：

```
reg_t *reg = get_reg();
fprintf(spm_code_file, "  lw %s, %d($fp)\n", reg->name, param_offset);  //
reg = ($fp + param_offset)对应的值
fprintf(spm_code_file, "  sw %s, -%d($fp)\n", reg->name, get_offset(p-
>code->u.one.op));  // reg = $t0
RELEASE_REG(reg);
param_offset += 4;
```

（3）调试方式

由于采用了最简单的实现方式，因此实验四的工作量就是调试以及正确性测试了。
我采用的调试工具是Vscode拓展插件的dashmips。该工具支持单步和断点调试，并且可以十分方便的查看内存和寄存器的值，十分方便地帮助我调试生成目标代码。限制于篇幅，调试的细节此处略去。

如下图所示：