编译原理 lab4 实验报告 201220180 李全昊

一. 实验的功能及实验内容

在词法分析、语法分析、语义分析和中间代码生成程序的基础上，将 C--源代码翻译为 MIPS32 指令序列（可以包含伪指令），并在 SPIM Simulator 上运行

二. 实验步骤

本次实验我采用的是线性 IR：如果你的程序使用了线形 IR，那么最简单的指令选择方式是逐条将中间代码对应到目标代码

本次实验让我印象最深的是对于栈和寄存器的操作。生成指令序列的过程本质上就是在和寄存器和栈打交道，要把数据在这两个地方挪来挪去

对于栈帧的设计，我和 pdf 中保持相同

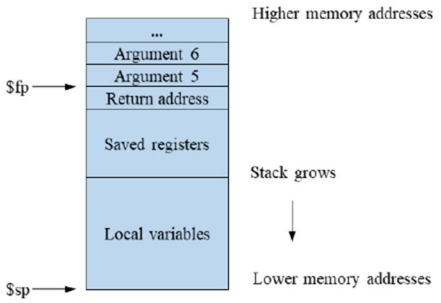

图21. 函数的活动记录结构。

对于调用者来说：

调用者 f 在调用函数 g 之前需要将保存着活跃变量的所有调用者保存寄存器 live1、live2、…、livek 写到栈中，之后将参数 arg1、arg2、…、argn 传入寄存器或者栈。在函数调用结束后，依次将之前保存的内容从栈中恢复出来

```
1  sw live1, offset_live1($sp)
2  …
3  sw livek, offset_livek($sp)
4  subu $sp, $sp, max{0, 4 * (n - 4)}
```

115

```
5  move $a0, arg1
6  …
7  move $a3, arg4
8  sw arg5, 0($sp)
9  …
10 sw argn, (4 * (n - 5))($sp)
11 jal g
12 addi $sp, $sp, max{0, 4 * (n - 4)}
13 lw live1, offset_live1($sp)
14 …
15 lw livek, offset_livek($sp)
```

对于被调用者来说：

我们首先要负责布置好本函数的活动记录。如果本函数内部还要调用其它函数，则需要将$ra 压栈；如果用到了$fp 还要$fp 压栈并设置好新的$fp。随后，将本函数内所要用到的所有被调用者保存的寄存器 reg1、reg2、…、regk 存入栈，最后将调用者由栈中传入的实参作为形参 p5、p6、…、pn 取出

```
subu $sp, $sp, framesizeg
sw $ra, (framesize_g - 4)($sp)
sw $fp, (framesize_g - 8)($sp)
addi $fp, $sp, framesize_g
sw reg_1, offset_reg1($sp)
…
sw reg_k, offset_regk($sp)
lw p_5, (framesize_g)($sp)
…
lw p_n, (framesize_g + 4 * (n - 5))($sp)
```

在Epilogue中，我们需要将函数开头保存过的寄存器恢复出来，然后将栈恢复原样：

```
lw reg_1, offset_reg1($sp)
…
lw reg_k, offset_regk($sp)
lw $ra, (framesize_g - 4)($sp)
lw $fp, (framesize_g -8)($sp)
addi $sp, $sp, framesize_g
jr $ra
```

整体代码写的时候主要参见实验指导

三. 实验总结

这次实验还是比较困难的，花了不少时间