编译原理 lab4 报告

201220116 王少杰

2023 年 5 月 25 日

1 运行方式

在 code 文件夹下面使用 make 编译生成可执行文件 parser, 然后使用命令 > 文件路径/parser 文件路径/测试样例名.cmm 文件路径/测试样例名.s 得到.s 文件。之后将.s 文件导入 spim 执行即可。

2 实验总结

这次实验主要将实验三的 IR 转化为对应的汇编语言，本次实验主要处理编译过程的后半部分，也就是后端。实验的难点主要在如何为中间变量选取寄存器以及活动空间的布局。指令选择一目了然，为了布局活动空间可能会加一些辅助的指令。

3 指令选择

指令选择十分简单，就是一条一条翻译，也有固定的翻译方法，只是要注意 $sp 的使用，例如：

```
// addi $t1, $t1, d 减法，d取负数就可以
```

```
//函数调用
/*
fprintf(dest, "addi $sp, $sp, -8\n");
fprintf(dest, "sw $ra, 0($sp)\n"); //返回地址压栈
fprintf(dest, "sw $fp, 4($sp)\n"); //原来的帧压栈
```

```
fprintf(dest, "jal %s\n",ir->info.call.func);
fprintf(dest, "lw $ra, 0($sp)\n");
fprintf(dest, "lw $fp, 4($sp)\n");
fprintf(dest, "addi $sp, $sp, 8\n");
*/
```

---

4 栈的分配

我的活动记录设置如下：在新的函数开始后，使用 move 指令设置新的

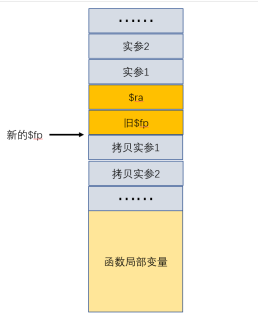

图 1: 活动记录

$fp，然后将所有的实参拷贝到本函数记录里，即从 $fp+8,$fp+12,$fp+16，... 拷贝到 $fp-4,$fp-8,$fp-12。

5 寄存器选择

我选择的是朴素寄存器算法，只要是运用到变量就要去内存中去取，我们如何定位变量对应的内存地址？因为 $sp 在一直变化所以我们选择 $fp 这个固定帧地址来作为一个起始地址。我们首先设置了一个全局数组 offset 用来表示对应标号中间变量对应的偏移量（相对于本活动的 $fp)，例如 t1 对应 offset[1]。这样的话我们每次翻译 IR 遇到变量时，如果出现过，那么直接获得偏移量去加载，否则将当前偏移量再-4 当作这个变量新的偏移量保存进数组即可。每当我们要使用一个变量我们就使用 lw 命令，如果要保存就使用 sw 命令。下面展示的是函数是检查数组中是否已经有偏移，没有则赋予新的偏移。

```
void Assign_offset_and_renew_offset(FILE *dest,struct IR* ir,int
    label_index)
  {;
     if(!has_offset(label_index))//保存了相应的偏移
     {
        if(ir->kind==DEC_T)//数组要分配更多的空间
        {
           fprintf(dest, "addi $sp, $sp, %d\n",-ir->info.dec.size);
           cur_offset_dec(ir->info.dec.size);//偏移量减数组的大小
           set_offset(label_index, var_offsets.cur_activity_offset);
        }
        else
        {
           fprintf(dest, "addi $sp, $sp, -4\n");
           cur_offset_dec(4);//偏移量-4
           set_offset(label_index, var_offsets.cur_activity_offset);
        }
     }
  }
```