编译原理Lab4实验报告

1. 完成情况：本次实验完成了所有要求，通过了OJ所有测试。
2. 编译方法：`make`生成可执行文件; `./parser <input>  out1.s` 编译input测试用例，将mips汇编代码输出到`out1.s`中。
3. 在实验3中间代码生成的基础上，我选择了新开一个文件`codegen.c`, 来实现IR到Mips指令的翻译
4. 首先对调用约定进行规定：所有变量(v*)，临时变量(t*)都存在栈上，sp指针指向栈底第一个元素，调用函数时调用者将IR中参数依次压栈（IR中实参传递为倒序），并将返回地址RA也一起压栈，然后跳转到被调用者。被调用者根据自己栈帧的大小，计算出栈帧的大小（除去调用者给出的参数，返回地址）并扩栈。函数返回前，弹栈，并将返回值存在寄存器V0中。调用者恢复返回地址RA，将返回地址和参数一并弹栈，并将V0中值赋给对应变量。
5. 对寄存器分配，我选择了最朴素的寄存器分配方式：将变量全部溢出到内存中，这样指令模式的选择就非常的简单，非常的方便。
    1. 首先对于每个函数，在寄存器分配的时候计算出其栈帧（活动记录）的大小——遍历每条指令，找它的左值，如果左值是未分配栈上空间的普通变量，临时变量，或者DEC变量，则为它分配栈上空间，其中普通变量占4Bytes，DEC x size的数组则占size大小。
    2. 我在IR的Operand中新建了一个`sp_offset`字段，在寄存器分配的时候为每个变量记录其分配到的的地址栈帧。
6. 指令翻译部分，我是在我的IR架构上，对Module中每个函数，函数中逐条IR做线性的翻译。
    1. 首先当然是照抄手册上的read, write等框架
    2. 指令翻译的时候经常会需要取左值，因为我们所有变量都在内存中，我定义了一个`mem2reg`的函数，将给定变量对应的值读到寄存器中：

```
void mem2reg_unary(RegNum reg, Operand op){
    if (op->kind == CONSTANT) {
        fprintf(ostream, "\tli %s, %d\n", regName[reg], op->u.value);
    }
    else if (op->kind == ADDRESS) { // &t1
        fprintf(ostream, "\taddi %s, %s, %d\n", regName[reg],
regName[SP], op->u.pt->sp_offset);
    }
    else if (op->kind == REFERENCE) { // *t1
        fprintf(ostream, "\tlw %s, %d(%s)\n", regName[reg], op->u.pt-
>sp_offset, regName[SP]);
        fprintf(ostream, "\tlw %s, %d(%s)\n", regName[reg], 0,
regName[reg]);
    }
    else {
        fprintf(ostream, "\tlw %s, %d(%s)\n", regName[reg], op-
>sp_offset, regName[SP]);
    }
}

void mem2reg_binary(RegNum reg1, RegNum reg2, Operand op1, Operand op2){
    mem2reg_unary(reg1, op1);
    mem2reg_unary(reg2, op2);
}
```

这样我们所有读取变量右值的操作都只需要调用这两个函数一次就能解决，在不同的翻译模式里，只需要考虑如何将寄存器中的值赋值给左值/在各种地方使用即可

3. 对于 FUNCT 类型的 IR，我首先生成一个 `<funcname>:`, 然后生成一条扩栈语句。与此对等地，在函数的每一个出口，也就是 RETURN 类型的 IR，在使用 `jr $ra` 之前，我们都需要增加一条弹栈语句。
4. 对于READ WRITE, 我并没有遵守我自己定义的调用约定，而是使用了手册上的实现，使用寄存器传参。

7. 实现的时候，由于我选择的寄存器分配方式比较简陋，所以指令翻译部分基本上很快就实现完了，基本没遇到什么困难。困难之处主要在于设计栈帧方面。我一开始的设计是，对于每个函数，在调用的时候就将其栈帧分配完成。但是这样就带来了一个问题：main函数的栈帧没有被分配。事实上即使没有分配，对于一些本地的简单样例，spim上也是可以运行的。但是提交到OJ上就卡在了64分，我尝试了很多可能，比如说考虑函数名与指令名重名的问题（事实证明OJ没有测），$-1/2$ 和 $(-1)/2$ 在 IR 中与手册中给出的除法指令不匹配的问题（事实上OJ也没测），都没有找到问题。后来我通过spim上的单步执行发现了问题，为main函数单独判断了一下，在入口和出口处增加了扩栈弹栈指令。结果我把扩展指令生成在了 `main:` 的上面，因此卡住了，又过了好久才发现问题。最后就干脆全部改成被调用者扩栈了。
8. 感想：如果考虑到寄存器分配的话，目标代码生成还是很困难的，由于时间关系更高级的寄存器分配也没有时间实现了。那么使用自己的框架做的实验（前四个）就到此为止了，回头看来从零开始自己写一个能跑的编译器还是很有挑战性的，自己的代码能力，调试能力也得到了很大的提升。