编译实验LAB-C4：短路求值

这个短路求值实验有一些难度，我思考了一下午怎么设计条件之间的跳转，踩了好多坑，终于在晚上通过了评测。在讲解实现方式前先记录一些刚刚踩的坑：

- while循环也需要短路求值！！！实验手册给的样例只包含了if语句的情况，我一开始还以为只有if需要短路求值，while保持不变，后来仔细思考并和同学交流后才知道，太坑了。。。
- 条件跳转label需要嵌套！每一层的条件语句都需要实现短路求值！我一开始的实现方式是只有最外面一层的&&或||具备短路求值功能，例如`(a&&b)||c`的表达式只有`||`具有短路求值的特性，内层的`&&`还是按以前的方法执行。之后也是仔细思考并和同学交流后才知道我理解的有偏差，于是乎重构。。。

根据上面踩的坑总结的经验教训，我摸索出一条可行的路作为了我最终实现的思路。

解决多层条件语句的跳转label的储存

在复杂的条件语句中，每一部分条件在计算为真或假时都需要跳转到不同的label，如果用单全局变量保存的话只能记录最外层跳转的label（例如if语句的if和else部分），内部的语句短路所需跳转到的label无法保存。

因此我设想通过两个全局栈来实现，其中一个保存当前条件语句执行为真的情况下需要跳转到的label，另一个保存执行为假的情况下需要跳转到的label。

```
private Stack<String> condTrue = new Stack<>(); // 存储条件成立对应的跳转label
private Stack<String> condFalse = new Stack<>(); // 存储条件不成立对应的跳转label
```

这样每次需要根据短路原则跳转时，只需获取栈顶的元素跳转过去即可。

IF-ELSE语义分析部分的修改

原本我的实现方法是把所有条件通过`and/or`指令计算出来得到一个结果，然后通过`visit(ctx.cond());`获取这个结果，再判断真假并跳转到不同的label。为了实现本次的短路求值实验，我采用了"控制流法"，即抛弃所有`and/or`的计算，每个条件计算都根据结果进行跳转，如果符合短路规则就直接跳转到出口，否则跳转到下一个条件计算。

```
String true_reg = "%r" + regId++;
String false_reg = "%r" + regId++;
condTrue.push(true_reg);
condFalse.push(false_reg);

visit(ctx.cond());
String end_reg = "%r" + regId++;

// TRUE-BLOCK
System.out.println(true_reg.substring(1) + ":");
visit(ctx.stmt(0));
System.out.println("    br label " + end_reg);

// FALSE-BLOCK
System.out.println(false_reg.substring(1) + ":");
if(ctx.stmt(1) != null) {
  visit(ctx.stmt(1));
}
System.out.println("    br label " + end_reg);
//
//            // END-BLOCK
System.out.println(end_reg.substring(1) + ":");
```

WHILE语义分析部分的修改

与if-else部分的修改基本一致，只是条件跳转方式不太一样，注意即可。

条件跳转的实现方式

LOrExp在条件表达式的最上层，我们在此层需要为程序保存控制流出口的label，分别压入两个全局栈中。对应于OR的短路规则，如果条件为真则跳转到出口，否则跳转到下一个条件计算。具体实现见下面的代码。

```
public String visitLOrExp(SysYParser.LOrExpContext ctx) {
  if(ctx.lOrExp() == null) {
    return visit(ctx.lAndExp());
  }
  else {
    // 短路求值
    String inner_lor = "%r" + regId++;
    condTrue.push(condTrue.peek());
    condFalse.push(inner_lor);
    visit(ctx.lOrExp());
    System.out.println("\n" + inner_lor.substring(1) + ":");
    visit(ctx.lAndExp());
    return null;
  }
}
```

LAndExp在LOrExp的下层，是短路求值的最下层语法结构，因此也承担着输出跳转指令 `br` 的功能。输出跳转指令有两个时机：

1. 这个lAndExp中只包含EqExp，即已经是计算表达式的最小单位，不存在逻辑关系了，此时输出一条br语句，根据EqExp的计算结果，对应两个栈进行跳转。这里的跳转代表LandExp的最后一个条件元素的跳转，并且额外的，只有OR关系的表达式的跳转也是在此输出。
2. 在进入左递归的新的LAndExp并计算完成之后，根据LAndExp的计算结果和两个栈进行跳转。这里的跳转代表LAndExp中间的（除了最后一个外）的条件元素的跳转。

```
public String visitLAndExp(SysYParser.LAndExpContext ctx) {
  if(ctx.lAndExp() == null) {
    String eqExpVal = visit(ctx.eqExp());
    String res_reg = "%r" + regId++;
    System.out.println("    " + res_reg + " = icmp ne i32 " +
eqExpVal + ", 0");
    System.out.println("    br i1 " + res_reg + ", label " +
condTrue.pop() + ", label " + condFalse.pop());
```

```
    return res_reg;
  }
  else {
    // 短路求值
    String inner_land = "%r" + regId++;
    condTrue.push(inner_land);
    condFalse.push(condFalse.peek());
    visit(ctx.lAndExp());
    System.out.println("\n" + inner_land.substring(1) + ":");
    String eqExpVal = visit(ctx.eqExp());
    String eq_result = "%r" + regId++;
    System.out.println("    " + eq_result + " = icmp ne i32 " +
eqExpVal + ", 0");
    System.out.println("    br i1 " + eq_result + ", label " +
condTrue.pop() + ", label " + condFalse.pop());
    return eq_result;
  }
}
```