实验报告—step3

2018011340 计 83 郭峥岩

一、 实验内容

1. 整体框架

本次实验中 step3 的整体架构与之前大致相同，不需要进行特别的改动。

Step3 中实现的重点是语法分析过程中的优先性、结合性等性质的包含，执行过程中栈的数据的变化等。

2. Step3 的功能实现

（1） 实现程序的语法规范的改动：

增加了加减乘除模以及 括号的运算规则，此时重点在于通过左结合或者右结合来实现各个运算之间的优先级，正确得到最终的返回结果。具体的语法可以参照助教给出的语法：

expression： additive

additive： multiplicative | additive ('+'|'−') multiplicative

multiplicative： unary | multiplicative ('∗'|'/'|'%')unary

unary： primary | ('−'|'~'|'!')unary

primary： Integer | '(' expression ')'

（2） IR 类的更改和产生：

增加了一种 IR 类：二元操作符类，同时在类中存储操作符，在生成 IR 的时候根据运算符种类生成对应相应的 IR 指令。正确设定 AST 的遍历顺序，将对应的 IR 种类按照顺序存储到列表。

（3） 汇编代码的产生：

具体的生成过程同 step2，根据上述的 IR 代码，设定好二元操作符对应的汇编代码，按照顺序写入即可。

二、 思考题

1. 请给出将寄存器 t0 中的数值压入栈中所需的 riscv 汇编指令序列；请给

出将栈顶的数值弹出到寄存器 t0 中所需的 riscv 汇编指令序列。

解答：

将寄存器 t0 的数值压入栈中的汇编指令序列：

$$\text{addi}\ \ \text{sp,sp,}-4$$

$$\text{sw}\ \ \text{t0,0(sp)}$$

将栈顶的数值弹出到寄存器中所需的指令：

$$\text{lw}\ \ \text{t0,0(sp)}$$

$$\text{addi}\ \ \text{sp,sp,4}$$

2. 语义规范中规定“除以零、模零都是未定义行为”，但是即使除法的右操作数不是 0，仍然可能存在未定义行为。请问这时除法的左操作数和右操作数分别是什么？请将这时除法的左操作数和右操作数填入下面的代码中，分别在你的电脑（请标明你的电脑的架构，比如 x86-64 或 ARM）中和 RISCV-32 的 qemu 模拟器中编译运行下面的代码，并给出运行结果。（编译时请不要开启任何编译优化）

解答：经过插入之后的代码如下所示：

```
#include <stdio.h>
int main () {
   int a = -2147483648;
   int b =  −1;
   printf("%d\n", a / b);
   return 0;
}
```

上述代码中，我们使$a = -2147483648$ (即 -2^{31}), $b = -1$。则此时使用除法按照原本正确的值会产生2^{31}，但是由于超过 32bit 整数的最大范围，因此会产生未定义的行为。即实现了即使右操作数不为 0，仍有可能产生未定义行为的要求。

程序的运行结果如下：

电脑架构为X86 – 64：

```
guozy@guozy-virtual-machine:~/Program Files$ gcc -o0 test.c -o test
guozy@guozy-virtual-machine:~/Program Files$ ./test
Floating point exception (core dumped)
```

在 RISCV-32 的 qemu 模拟器中:

```
guozy@guozy-virtual-machine:~/Program Files$ riscv64-unknown-elf-gcc -march=rv32
im -mabi=ilp32 -O0 test.c
guozy@guozy-virtual-machine:~/Program Files$ qemu-riscv32 a.out
-2147483648
```

三、 参考资料

助教所写的代码：md-dzy branch

四、 总结

本次编译原理的实验 step3 的重点在于二元指令的汇编代码的生成：

1. 运用上学期学习的自动机的相关知识进行设计，通过右递归的形式设计出合适的语法结构，确保二元运算能够具备优先性
2. 在遍历生成的语法分析树生成中间代码的时候，注意递归和访问节点的顺序。
3. 注意各个运算的汇编代码，具体的运算及生成原则在数据结构课程中有所涉及，总体而言难度不大。