- 占海川 2021050009

实验内容

- stage-0 主要完成了环境相关配置，并初步阅读代码框架，对编译器结构、AST的遍历模式有了一个初步认识
- stage-1
    1. step2: 在`tacgen.py/visitUnary`中加入了从`UnaryOp`到`TacUnaryOp`的一元运算符的映射关系; 在`riscvasmemitter.py/visitUnary`中加入了从`TacUnaryOp`到`RvUnaryOp`的一元运算符的映射关系; 在`tacinstr.py/Unary`中运算符类型判断; 最终在`riscv.py`里的枚举类中加入所需汇编指令
    2. step3: 与step2类似, 在两文件的`visitBinary`中加入不同类之间二元运算符的映射关系; 最终在`riscv.py`里的枚举类中加入所需汇编指令
    3. step4: 相比step3的区别是除`<`外, 其他`TacBinaryOp`无法翻译为一条汇编代码, 需要在`riscvasmemitter.py/visitBinary`中针对不同操作符, 基于`<`实现

思考题

- step1
    1. 代码可以正常编译. `namer`与`type`的`transform`接受`AST`为输入, 前者负责扫描各位置的标识符(函数名/变量名), 构建符号表, 后者负责进行类型检查(int/int[]). 而此代码块不会涉及到以上两种检查, 因此不会报错
    2. 报错: `Syntax error: EOF` 首先, 所有错误类型在`error.py`中定义, 无返回值处理在语法分析部分的`ply_parser.py`中, 当main函数缺少返回值, 即`p_error`的参数`t`为空时, `p_error`会将`EOF`放入`error_stack`中, 最终在`main.py/step_parse`中打印出来
    3. AST, TAC与汇编都需要使用visitor模式遍历操作节点; 同时三类操作符之间不一定是线性映射关系, 如`TacBinaryOp`的`&&/||/<=`等操作符没有汇编指令与其直接对应, 因此需要定义三种运算符类型
- step2

```
int main() {
    return -~2147483647;
}
```

- step3 左操作数为-2147483648, 右操作数为-1. 在x86架构下, 运行结果为`Floating point exception`, 在riscv模拟器下, 运行结果为`Floating point exception (core dumped)`
- step4 短路求值的优点在于短路之后不必继续运算后面的表达式, 节省了时间开销, 对程序员来说也可以把易于计算的值放在表达式前端, 以进一步节省时间