# STAGE1: 常量表达式 实验报告

邢竞择 2020012890

# 1 Step 2

#### 1.1 实验内容

支持单目运算符取负 - 、按位取反 ~ 、逻辑非!

- 在 scanner.1 中添加了~!的识别匹配
- 在 parser.y 中添加了 BitNotExpr NotExpr 的语法
- 在 SemPass2 中增加了新节点的类型检查规则
- 给新增的语法树节点增加了生成 ir 的规则
- 在 void RiscvDesc::emitTac 中增加了新指令三地址码的生成
- 在 void RiscvDesc::emitInstr 增加了汇编指令 NOT SEQZ SNEZ 指令的发射

#### 1.2 思考题

-~2147483647

## 2 Step 3

#### 2.1 实验内容

支持双目运算符 + - \* / % 以及()

- 在 scanner.1 中添加了 \* / %的识别匹配
- 在 parser.y 中添加了相应的 5 个语法
- 在 SemPass2 中增加了新节点的类型检查规则
- 给新增的语法树节点增加了生成 ir 的规则
- 在 void RiscvDesc::emitTac 中增加了新指令三地址码的生成
- 在 void RiscvDesc::emitInstr 增加了汇编指令 ADD SUB MUL DIV REM 指令的发射

#### 2.2 思考题

在 x86-64 下,添加-00 选项编译成 riscv 平台的可执行程序,并在 qemu 中运行

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

#### 输出结果为

此外, 我尝试使用 linux 的 gcc 加上 -O0 编译, 程序在运行时抛出异常, 输出

floating point exception

推测此时各种检查都正常开启,导致溢出被察觉。

### 3 Step 4

### 3.1 实验内容

支持二元的比较算符、逻辑与、逻辑或

- 在 scanner.1 中添加了 && || == != <= >= < > 的识别匹配
- 在 parser.y 中添加了相应的语法
- 在 SemPass2 中增加了新节点的类型检查规则
- 给新增的语法树节点增加了生成 ir 的规则
- 在 void RiscvDesc::emitTac 中增加了新指令三地址码的生成
- 在 void RiscvDesc::emitInstr 增加了汇编指令 SLT GRT AND OR XOR 指令的发射

以上操作除了 LES 以外,均无法翻译成单一的汇编指令,因此在 emitBinaryTac 拦截这些虚假的 Risc-V 指令,发射多条汇编指令来完成操作,例如:

```
1
       switch(op) {
2
       case RiscvInstr::EQU:
3
           addInstr(RiscvInstr::SUB, _reg[r0], _reg[r1], _reg[r2], 0, EMPTY_STR, NULL);
4
           addInstr(RiscvInstr::SEQZ, _reg[r0], _reg[r0], NULL, 0, EMPTY_STR, NULL);
5
           break;
6
7
       default:
           addInstr(op, _reg[r0], _reg[r1], _reg[r2], 0, EMPTY_STR, NULL);
8
9
```

### 3.2 思考题:短路求值

若根据表达式左侧的结果已经可以推断整个表达式的结果,那么可以避免计算表达式右侧的值。如果右侧值计算繁琐、消耗大,则短路可以提高性能;如果右侧值的正确计算依赖于表达式左侧值为1或为0,则短路求值能够让给程序员避免使用嵌套语句,使代码更加简洁。编译器提供的这一自动常数优化是非常优秀的特性,广受欢迎。