stage-1 实验报告

秦若愚 2019011115

实验内容

step-2

实验目标

- 增加一元运算：取负 `-`、按位取反 `~` 、逻辑非 `!`

实验内容

- 在 frontend/tacgen/tacgen.py 的 TACGen 类的 visitUnary 函数中加入了 `~` 和 `!` 对应的 TAC 操作符（定义于 tacop.UnaryOp），分别为 NOT 和 SEQZ。
- 在 utils/tac/tacinstr.py 的 Unary 类的 __str__ 函数中，加入了对 UnaryOp.NOT 和 UnaryOp.SEQZ 的区分。

step-3

实验目标

- 增加二元运算：加 `+`、减 `-`、乘 `*`、整除 `/`、模 `%`
- 增加括号 `(` `)`

实验内容

- 在 frontend/tacgen/tacgen.py 的 TACGen 类的 visitBinary 函数中加入了 `-`、 `*`、 `/`、 `%` 对应的 TAC 操作符（定义于 tacop.BinaryOp），分别是 SUB、MUL、DIV、REM。

step-4

实验目标

- 增加比较大小和相等的二元操作：`<`、`<=`、`>=`、 `>`、 `==`、 `!=`
- 增加逻辑与 `&&`、逻辑或 `||`

实验内容

- 在 frontend/tacgen/tacgen.py 的 TACGen 类的 visitBinary 函数中加入了 `<`、`<=`、`>=`、 `>`、 `==`、 `!=`、 `&&`、 `||` 对应的 TAC 操作符（定义于 tacop.BinaryOp），分别是 SLT、LEQ、GEQ、SGT、EQU、NEQ、AND、OR。
- 由于 LEQ、GEQ、EQU、NEQ、AND、OR 这几个 TAC 操作符在 RISC-V 中没有实际对应的指令，因此需要选择合适的 RISC-V 指令来完成翻译工作。我在 backend/riscv/riscvasmemitter.py 的 RiscvInstrSelector 类的 visitBinary 函数中加入了翻译这些 TAC 操作符的代码。

思考题

step-2

1. 我们在语义规范中规定整数运算越界是未定义行为，运算越界可以简单理解成理论上的运算结果没有办法保存在32位整数的空间中，必须截断高于32位的内容。请设计一个 minidecaf 表达式，只使用`-~!`这三个单目运算符和从 0 到 2147483647 范围内的非负整数，使得运算过程中发生越界。

答：表达式如下：

```
-~2147483647
```

因为`~2147483647`=`-2147483648`，所以`-~2147483647`=`2147483648`，发生了越界。

step-3

1. 我们知道“除数为零的除法是未定义行为”，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）

```
#include <stdio.h>

int main() {
  int a = -2147483648;
  int b = -1;
  printf("%d\n", a / b);
  return 0;
}
```

答：除法溢出同样会导致未定义行为，代码见上。

在 x86-64 电脑中编译运行的结果为

```
Floating point exception
```

在 RISCV-32 的 qemu 模拟器中编译运行的结果为

```
-2147483648
```

step-4

1. 在 MiniDecaf 中，我们对于短路求值未做要求，但在包括 C 语言的大多数流行的语言中，短路求值都是被支持的。为何这一特性广受欢迎？你认为短路求值这一特性会给程序员带来怎样的好处？

答：短路求值的好处有以下两点：

（1）可以节约计算成本。在多个表达式求逻辑运算的情况下，若前面的表达式可以确定最终结果，则根据短路求值可以不计算之后的表达式，从而节约计算成本。

（2）可以使代码更简洁。可以构造一个表达式来使第二个可能会发生运行时错误的表达式成功运行，比如下面的这种情况：

```
bool is_valid_string(const char *p)
{
    return p != NULL && is_valid_character(p[0]);
}
```