

编译原理 stage 4 报告

傅子轩 2020010742 计01

主要改动

1. `parser` 中增加了对函数定义参数列表和函数调用参数列表的识别。
2. `build_sym` 对 `CallExpr` 寻找对应函数的符号，对全局变量设置符号，并设置初值
3. `type_check` 中对 `CallExpr` 的返回值进行类型检查。
4. `translation` 增加对 `CallExpr` 的翻译，首先所有参数 `PUSH`，然后 `CALL`，对于全局变量的相关访问，把视作对地址的解引用，在 `varRef` 时 `LoadSymbol` 得到地址，`LvalueExpr` 和 `AssignExpr` 时进行相应的 `Load` 和 `Store`。
5. `tac`，新增 `Tac` 指令: `Load`, `Store`, `LoadSymbol`, `Push`, `Pop`, `Call`，前三个对应 `lw`, `sw`, `la`，这里 `PUSH` 的语义和通常一致，`POP` 则为从上一个函数的栈帧中弹出相应值，即使用 `FP` 作为基址且不修改寄存器。`CALL` 则是先将所有 `live` 的寄存器 `spill` 到栈上，再根据 `PUSH` 的次数修改 `sp`，`JAL Label`，返回时恢复 `sp`
6. `riscv_md` 按上述语义增加了到 `riscv` 指令的翻译，并修改了 `dataflow` 对应的位置使得数据流分析保持正确
7. 对于全局变量分配，遍历符号表，在对应 `.data` 或 `.bss` 段定义标签。

思考题

step 9

```
int foo(int x, int y) {
    return x-y;
}

int main() {
    int x = 10;
    return foo((x=x-1), (x=x-1))
}
```

若从左到右求值，则返回值为 1，若从右到左求值，则返回值为 -1

step 10

若使用 `pc` 相对寻址，可能翻译成如下情况：

(在 `offset_a` 的最高位不为1时)

```
auipc v0, offset_a[31:12]
addi v0, v0, offset_a[11:0]
```

若使用绝对寻址，则可以翻译成如下两种情况：

(在 `addr_a` 的最高位不为1时)

```
lui v0, addr_a[31:12]  
addi v0, v0, addr_a[11:0]
```

addr_a 高位均为0时

```
addi v0, zero, addr_a[11:0]
```