stage-4 实验报告

秦若愚 2019011115

实验内容

step-9

实验目标

- 支持函数的声明和定义
- 支持函数调用

实验内容

在词法语法分析方面，为了解析函数的定义、声明以及调用，我新增了 `Parameter` 和 `Call` 这两个 AST 节点，并为 `Function` 节点新增加了 `parameters` 属性。 同时，在 frontend/parser/ply_parser.py 中增加了对应的解析语法。

在语义分析方面，我在 frontend/typecheck/namer.py 中定义（或改动）了 visitProgram、visitFunction、visitParameter 和 visitCall 这四个函数。在 visitProgram 中将访问逻辑由只访问 main 函数改为了访问所有函数体；在 visitFunction 中新增了函数定义相关的检查、函数符号的维护、新建函数参数作用域以及调用 visitParameter 对参数进行检查等；在 visitCall 中完成了函数调用相关的检查以及调用 visitParameter 对参数进行检查等。

在中间代码生成阶段，我在 frontend/tacgen/tacgen.py 中新定义了 visitParameter 和 visitCall 这两个函数，并改动了 transform 函数。在 transform 中将只生成 main 函数的 TAC 指令改为了生成所有已定义的函数的 TAC 指令，同时通过调用 visitParameter 为带有参数的函数分配各个参数的 Temp；在 visitCall 中生成调用函数的 TAC 指令。

在目标代码生成阶段，我在 utils/riscv.py 中新增了 Call 指令以及为了方便生成代码而设计的 NativeFPLoadWord、NativeMove 等指令，并在 backend/riscv/riscvasmemitter.py 的 RiscvInstrSelector 类中实现了函数调用的代码生成。在 backend/riscv/riscvasmemitter.py 的 RiscvSubroutineEmitter 类中，我新增了函数中 callee-saved 寄存器的保存和回复，以及参数传递的相关内容。在 backend/reg/bruteregalloc.py 中，我按照调用约定实现了调用函数时的 caller-saved 寄存器保存与恢复以及参数传递部分的代码生成。

step-10

实验目标

- 支持全局变量

实验内容

在词法语法分析方面，我在 frontend/parser/ply_parser.py 中新增了对全局变量的解析，将全局变量的声明与定义归入 `Declaration` 类中。

在语义分析方面，我在 frontend/typecheck/namer.py 的 visitDeclaration 函数中新增了对全局变量符号的维护、重复定义的检查以及初始化表达式不是整数字面量的检查等。

在中间代码生成阶段，我在 frontend/tacgen/tacgen.py 的 visitIdentifier 和 visitAssignment 函数中加入了要访问的表达式是否是全局变量的判断，若是则通过在 utils/tac/tacinstr.py 中定义的 visitLoadSymbol、visitLoadWord、visitStoreWord 实现对全局变量的访问与赋值。同时将全部的全局变量信息存于 TACProg 中，以便打印 TAC 代码和生成目标代码时使用。

在目标代码生成阶段，只需分别将 visitLoadSymbol、visitLoadWord、visitStoreWord 这三条 TAC 指令翻译为 riscv 代码，以及在程序开始时增加声明全局变量的指令（需根据是否初始化区分 .data 和 .bss 两类）即可。

思考题

step-9

1. 答：

```
int func(int a, int b) {
    return a - b;
}
int main() {
    int x = 1;
    return func(x = x + 1, x);
}
```

main 函数的返回值有 1 和 0 两种可能的结果。

2. 答：

如果所有寄存器完全由 caller 一方保存，则每次调用函数时都要保存所有当前用到的寄存器，而一个函数往往会被多次调用，这大大增加了代码的复杂程度，因此不可行。而又因为 ra 寄存器需要保存返回地址，a0 寄存器需要保存返回值，这些寄存器不可能是 callee-saved 寄存器，所以所有寄存器完全由 callee 一方保存也不可行。

因为 ret 是一个函数的最后一条指令，此时 ra 必须设成返回地址，因此 callee 即使保存了 ra 的值也没有机会恢复，所以 ra 寄存器必须是 caller-saved 寄存器。

step-10

1. 答：

第一种可能：

```
auipc rd, delta[31:12] + delta[11]
addi rd, rd, delta[11:0]
```

第二种可能：

```
auipc rd, delta[31:12]
ori rd, rd, delta[11:0]
```

其中 delta 是目标地址 symbol 和当前指令地址 pc 之间的偏移：delta = a - pc。