stage-2

计13 沈佳茗 2021010745

一、思考题

1. 我们假定当前栈帧的栈顶地址存储在 sp 寄存器中，请写出一段 **risc-v 汇编代码**，将栈帧空间扩大 16 字节。（提示1：栈帧由高地址向低地址延伸；提示2：risc-v 汇编中 addi reg0, reg1, <立即数> 表示将 reg1 的值加上立即数存储到 reg0 中。）

   ```
   addi rsp, rsp, -16
   ```

2. 有些语言允许在同一个作用域中多次定义同名的变量，例如这是一段合法的 Rust 代码（你不需要精确了解它的含义，大致理解即可）：

   ```
   fn main() {
     let a = 0;
     let a = f(a);
     let a = g(a);
   }
   ```

   其中`f(a)`中的`a`是上一行的`let a = 0;`定义的，`g(a)`中的`a`是上一行的`let a = f(a);`。

   如果 MiniDecaf 也允许多次定义同名变量，并规定新的定义会覆盖之前的同名定义，请问在你的实现中，需要对定义变量和查找变量的逻辑做怎样的修改？（提示：如何区分一个作用域中**不同位置**的变量定义？）

   在 namer 中，如果遇到一个变量，首先查看它是否已经在符号表中。如果不在符号表中，则进行声明，如果有 init_expr 的话，将该符号的值设为 init_expr 的值。如果在符号表中可以找到该符号，则检查其他部分是否合法（如 init_expr 是否合法），如果均合法，则不增删符号，使用原先的符号，只更改该符号的值。

二、实验内容

1. 语义分析阶段

   在 frontend/typecheck/namer.py 中，修改 visitIdentifier，visitDeclaration, visitAssignment 函数。

   在 visitIdentifier 中，首先查看符号表中是否已定义了此标识符，如果没有定义则抛出错误，否则用 setattr 将该标识符(Identifier 对象) 的 symbol 属性设为在符号表中查找到的符号。

```
def visitIdentifier(self, ident: Identifier, ctx: Scope) -> None:
    """
    1. Use ctx.lookup to find the symbol corresponding to ident.
    2. If it has not been declared, raise a DecafUndefinedVarError.
    3. Set the 'symbol' attribute of ident.
    """
    sym = ctx.lookup(ident.value)
    if sym is None:
        raise DecafUndefinedVarError(ident.value)
    else:
        ident.setattr("symbol", sym)
    # raise NotImplementedError
```

在 visitDeclaration 中，首先查看符号表中是否已定义了此标识符。如果已经定义了，就抛出错误（重复定义）。如果没有定义，就新创建一个 symbol，并把它放到当前作用域的符号表中，使用 setattr 将当前 Declar 对象的 symbol 属性设为这个符号，如果有 init_except 则进一步检查 init_except 是否合法。

```
def visitDeclaration(self, decl: Declaration, ctx: Scope) -> None:
    """
    1. Use ctx.lookup to find if a variable with the same name has been
declared.
    2. If not, build a new VarSymbol, and put it into the current scope using
ctx.declare.
    3. Set the 'symbol' attribute of decl.
    4. If there is an initial value, visit it.
    """
    sym = ctx.lookup(decl.ident.value)
    if sym != None:
        # decl.accept(self, ctx)
        raise DecafDeclConflictError(decl.ident.value)
    else:
        new_varsymbol = VarSymbol(decl.ident.value, decl.var_t.type)
        ctx.declare(new_varsymbol)
        decl.setattr("symbol", new_varsymbol)
        if decl.init_expr is not NULL:
            decl.init_expr.accept(self, ctx)
```

在 visitAssignment 中，首先检查左值是否在符号表中。如果不在则抛出错误（未定义），接着分别检查等式的左值和右值是否合法。

```
def visitAssignment(self, expr: Assignment, ctx: Scope) -> None:
    """
    1. Refer to the implementation of visitBinary.
    """
    sym = ctx.lookup(expr.lhs.value)
    if sym is None:
        raise DecafUndefinedVarError(expr.lhs.value)
    else:
        expr.lhs.accept(self, ctx)
        expr.rhs.accept(self, ctx)
```

2. 中间代码生成阶段

在 frontend/tacgen/tacgen.py 中，修改 visitIdentifier，visitDeclaration, visitAssignment 函数。

在 visitIdentifier 中，使用 setattr 将 ident 中的 val 属性设置为 symbol 属性中的 temp。

```
def visitIdentifier(self, ident: Identifier, mv: TACFuncEmitter) -> None:
    """
    1. Set the 'val' attribute of ident as the temp variable of the 'symbol'
attribute of ident.
    """
    temp = ident.getattr('symbol').temp
    ident.setattr('val', temp)
    # raise NotImplementedError
```

在 visitDeclaration 中，获取该 Declaration 对象的 symbol，并新建一个 Temp 对象（新的临时变量）把它保存在 symbol 中。如果存在 init_expr ，则先访问 init_expr 完成 init_expr 中间代码的生成，再使用赋值语句将 init_expr 的值（val 属性，Temp 对象）赋给 symbol 中保存的 temp。

```
def visitDeclaration(self, decl: Declaration, mv: TACFuncEmitter) -> None:
    """
    1. Get the 'symbol' attribute of decl.
    2. Use mv.freshTemp to get a new temp variable for this symbol.
    3. If the declaration has an initial value, use mv.visitAssignment to set
it.
    """
    symbol = decl.getattr("symbol")
    symbol.temp = mv.freshTemp()
    decl.setattr("symbol", symbol)
    if decl.init_expr is not NULL:
        decl.init_expr.accept(self, mv)
        mv.visitAssignment(decl.getattr("symbol").temp,
decl.init_expr.getattr('val'))
```

在 visitAssignment 中，首先访问 rhs，生成有关 rhs 的中间代码。然后获取 lhs symbol 中的临时变量 temp，使用赋值语句将 rhs 的值（val 属性，Temp 对象）赋给 temp。最后将该赋值表达式的 val 属性设为右值的 val 属性。

```
def visitAssignment(self, expr: Assignment, mv: TACFuncEmitter) -> None:
    """
    1. Visit the right hand side of expr, and get the temp variable of left hand
side.
    2. Use mv.visitAssignment to emit an assignment instruction.
    3. Set the 'val' attribute of expr as the value of assignment instruction.
    """
    expr.rhs.accept(self, mv)
    temp = expr.lhs.getattr('symbol').temp
    mv.visitAssignment(temp, expr.rhs.getattr('val'))
    expr.setattr('val', expr.rhs.getattr('val'))
```

3. 后端

   在 backend/riscv/riscvasmemitter.py 中新增 RiscvAsmEmitter.RiscvInstrSelector.visitAssign 函数，将中间代码的赋值语句与 Riscv 中的 mv 指令对应。

```
def visitAssign(self, instr: Assign) ->None:
    self.seq.append(Riscv.Move(instr.dst, instr.src))
```