stage2 实验报告

姓名：郑友捷 学号： 2021010771

工作内容

step5： 局部变量与赋值

为了实现局部变量与赋值操作，主要修改在语义分析与中间代码生成部分，但也对后端riscv代码生成做了修改。

1. 在语义分析部分（对应`frontend/typecheck/name.py`），完善了包括声明语句、赋值语句、二元运算、变量标识字符的语义分析函数

   其中一部分代码逻辑如下，其他部分逻辑类似

   ```
   def visitDeclaration(self, decl: Declaration, ctx: ScopeStack) -> None:
       temp = ctx.findConflict(decl.ident.value)
       # 判断是否重名
       if temp != None:
           raise DecafDeclConflictError(decl.ident.value)
           new_symbol = VarSymbol(decl.ident.value, decl.var_t.type)
           ctx.declare(new_symbol)
           decl.setattr("symbol", new_symbol)
           if decl.init_expr != NULL:
               decl.init_expr.accept(self, ctx)
   ```

2. 在中间代码生成部分（对应`frontend/tacgen/tacgen.py`），完善了对变量标识字符、声明语句、赋值语句的中间代码生成。生成时先按照符号表读取每一个子部分的节点的符号，然后递归生成值之后赋予到节点的val属性。
3. 在目标代码生成部分（对应`backend/riscv/riscvasmemitter.py`），完善了`visitAssign`函数，实现了赋值语句的目标代码生成。

step6：if语句和条件表达式

对分支语句的修改集中在中间代码生成部分。

分支语句的条件语句`cond`，若条件正确则执行`then`，否则执行`otherwise`。则在执行生成中间代码时操作如下：

1. 首先要对条件语句进行递归生成tac，并插入分支语句，若`cond`不成立则跳转到`skiplabel`。
2. 递归`then`语句并在`then`语句之后插入`skiplabel`。再插入分支语句，无条件跳转到`exitlabel`。
3. 再递归`otherwise`语句并插入`exitlabel`。

   原理说明：若条件语句不成立，则直接跳转到`skiplabel`，实现跳过`then`语句的功能，否则则按照顺序执行`then`语句，并跳转到`exitlabel`，从而跳过`otherwise`语句。自此便实现了分支语句的功能。

思考题

1. 汇编代码为`addi sp sp -16`

2. 在语义分析部分 `frontend/typecheck/name.py` 中修改：

    1. 对于声明，不检查变量是否重名。
    2. 对于声明，若存在初始化语句，先递归检查它的初始化语句，之后再检查左边的变量声明语句。若声明的变量出现在了初始化语句中，则因为先递归检查了初始化语句，获得了初始化语句的值，因此可以将这个变量直接重新声明覆盖而不会影响正确性。
    3. 对于变量查找，无需进行逻辑修改，仍是寻找当前作用域中的符号，这就意味着若使用了多次定义的变量，则默认这个变量是最近定义的那个。

3. `python` 框架：

    本程序处理悬吊else问题的逻辑是：令else与最近的未匹配的if进行匹配。

    处理方式：在语法分析时，本框架将statement划分为两种，一种是 `statement_matched`，代表if与else成功匹配，另一种是 `statement_unmatched`，代表仅有单个if，未与其他else匹配。

    在制定产生式时，框架制定如下产生式：

    ```
    statement_matched : If LParen expression RParen statement_matched Else
    statement_matched
    statement_unmatched : If LParen expression RParen statement_matched Else
    statement_unmatched
    ```

    规定若if与else同时出现，则if与else中间的主体语句一定是 `statement_matched`，即if与else中间的语句也必须是保证if与else进行匹配的，则在这里面不会出现悬吊问题，则最外层的else必定与同层的if互相匹配，所以保证了else会与最近的未匹配的if进行匹配。

    那么对于多余的if语句，则制定了如下产生式：

    ```
    statement_unmatched : If LParen expression RParen statement
    ```

    即令if单独成句，然后对statement进行递归，则此时statement内部的else会与同层的if完成匹配，最外层的if就是单独的，从而保证了else与最近的未匹配的if进行匹配。

4. 在 `frontend/tacgen/tacgen.py` 的 `visitCondExpr` 函数中，逻辑修改为如下：

1. 依次递归访问 `cond、then、otherwise` 语句获取条件表达式的值。
2. 初始化两个标签 `skiplabel/exitlabel`，并加入条件判断语句：若 `cond` 语句成立，则跳转到 `skiplabel`
3. 加入赋值语句，将 `then` 语句的值赋予给 `expr` 节点，并加入无条件跳转语句调到 `exitlabel`
4. 加入标签 `skiplabel`，再加入赋值语句，将 `otherwise` 语句值赋予给 `expr` 节点，再加入 `exitlabel`。

上述逻辑实现了先求 `then` 与 `otherwise` 语句的值之后，根据条件 `cond` 语句的值进行选择赋值，从而做到不短路。