stage-4 实验报告

计 93 王哲凡 2019011200

实验内容

step9：函数

对于函数，在词法分析时，需要添加 function_list 以替代原有的 program 翻译的 function，则函数则可解析如下：

```
def p_function_def(p):
    """
    function : type Identifier LParen parameter_list RParen LBrace block RBrace
    """
    p[0] = Function(p[1], p[2], p[4], p[7])
```

其中 parameter_list 负责了函数参数列表（形参）的解析，使用了 Parameter 类的列表表示。

对应还需添加函数调用的解析：

```
def p_postfix(p):
    """
    postfix : Identifier LParen expression_list RParen
    """
    p[0] = FunctionCall(p[1], p[3])
```

其中 expression_list 负责了函数调用参数列表（实参）的解析，使用了 Expression 类的列表表示。

在符号表建立、检查阶段，需要添加上述类对应的 visit 函数。其中对于 visitFunction，需要检查函数类型的命名合法性，手动维护函数对应的局部作用域（考虑到参数的特殊性）；对于 visitFunctionCall，则需要检查函数名的存在性（且为函数符号）以及形参与实参的数量对应。

中间代码生成时，需要分别对各个函数进行中间代码生成，其中我添加了 PARAM A 用于 XX = CALL FUNC 之前，用于表示函数调用的实参，以及调用函数的行为，具体生成时需要注意函数调用时每个实参的获取。

目标代码生成时，我添加了 Push、LoadRet、Call、GetFP 等指令，分别使用 sw、mv、call、addi 汇编实现，分别表示函数参数压栈、函数返回值加载、函数调用和获取原有 SP 的值，为了使用 FP 寄存器，我将其加入了 CalleeSaved。

对于调用函数形参对应中间寄存器的指令，我通过 emitLoadFromStack 来对应获取真实的实参值。

在 GetFP 之前，我们需要保存 RA 寄存器，而在恢复 SP 之前，我们则需要复原。

寄存器分配算法中，我采用了 Call 指令统一分配，将所有使用过的 CallerSaved 寄存器保存在了栈上，并在调用结束获取返回值之后复原。

step10：全局变量

首先，在进行目标代码生成阶段时，为了方便后续代码，我将 RiscvAsmEmitter 的构造函数增加了 globalVars 以方便处理全局变量。其中，需要根据初始化情况进行分类：

```
for var in self.globalVars:
    if isinstance(var, Global):
        if var.initialized:
            self.printer.println(".data")
            self.printer.println(".global " + var.symbol)
            self.printer.printLabel(Label(LabelKind.BLOCK, var.symbol))
            self.printer.println(".word " + str(var.init))
        else
            self.printer.println(".data")
            self.printer.println(".global " + var.symbol)
            self.printer.printLabel(Label(LabelKind.BLOCK, var.symbol))
            self.printer.println(".word 0")
```

在解析阶段，可以在原有的 function_list 基础上添加新的规则以完成全局变量的词法分析：

```
def p_global_var(p):
    """
    function_list : declaration Semi function_list
    """
    p[0] = [p[1]] + p[3]
```

在符号表建立阶段，对于函数列表需要根据是否为全局变量进行分类，若为则将符号名添加至全局作用域（此时也需检查对应的命名冲突）。

在中间代码生成阶段，我讲全局变量相关信息全部放入了 ProgramWriter 中用以交给后续的目标代码生成处理，而在 visitIdentifier 和 visitAssignment 的过程中，都需根据是否访问的为全局变量，在进行对应的内存单元读和写。此处我新加入了 LoadSymbol、LoadMem、StoreMem 三种中间代码用以分别表示，获取符号地址、获取符号值、写回符号值，其对应类型均为 InstrKind.SEQ。

而中间代码向目标代码传递全局变量信息的方式，则为建立全新的 TACInstr 类型 Global，其记录了所需的符号名、初始化情况等信息。

实验思考题

step9：函数

```
int func(int x, int y) {
    return x + y;
}
int main() {
    int a = 1, b = 2;
    return func(a += 1, a + b);
}
```

在从左到右的顺序中，返回值为 6，而从右到左的顺序，则返回值为 5。

step10：全局变量

1. 可能翻译成：

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

其中 delta = a - PC。

也可以翻译为：

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

delta 同理。