Stage-4 实验报告

吴垒 2020010916

实验内容

函数感觉有点难的离谱，花了 4 天多的时间；全局变量就简单很多了。

Step9：

在 frontend/lexer/lex.py 添加 Comma 用于匹配逗号。

在 frontend/parser/ply_parser.py 中修改 p_func_def 以增加非终结符 paramlist 作为函数形参列表，添加 p_func_def 用于函数声明的语法翻译，增加 p_postfix_expression 用于函数调用的语法翻译，非终结符 expression_list 为函数调用对应的实参值列表。

在 frontend/ast/tree.py 中为 function 添加子属性 params，并新增 parameter 节点用于存储参数、Call 节点用于存储函数调用。

在 frontend/scope/scope.py 中添加 scopekind.formal 为函数作用域类型，该作用域需要包含函数的形参。

在 frontend/symbol/funcsymbol.py 中添加 hasbody 属性表示此处是仅声明或还有定义。

在 frontend/typecheck/namer.py 中修改 visitProgram 依次访问每个函数并在检查是否重名后在全局作用域声明，修改 visitFunction 完成函数声明与定义的访问、作用域的建立，修改 visitBlock 保证在进入函数的一级 block 时不会再次建立一个作用域导致 parameter 被默认隔离到独立作用域。

将 namer 中的递归访问逻辑 copy 进 typer，构建 typer 框架，并在 typer 中比对同名 function 声明时的参数数量是否相同。

在 utils/tac/tacinstr.py 中添加 Param 与 Call 指令，分别对应将一个寄存器设置成下一个函数参数、调用函数。

在 frontend/tacgen/tacgen.py 中修改 transform 为所有函数生成 tac 码，定义 visitCall 生成函数调用对应的 Param 与 Call 的代码。

在 utils/riscv.py 添加 Call、Store、SPAd、Load 指令，分别对应 native 转换之前的函数调用、将寄存器内容存储到某一地址、SP 指针增加、加载某地址中内容到寄存器。

为 backend/subroutineinfo.py 添加 numArgs 子属性用于传递当前编译的函数的参数个数。

在 backend/riscv/riscvasmemitter.py 中为 RiscvInstrSelector 添加 paraNum 子属性，表示目前已存储的即将调用的子函数的参数个数（即在本函数内上一次 Call 之后的 Param 数），增加 visitParam 将参数从栈顶在预留出 callerSaved 寄存器的空间之后向下存储，增加 visitCall 函数将所有 callerSaved 寄存器保存，并将栈内参数依次装载到 a0-a8 或压入栈顶，同时增加栈指针完成函数参数压栈的过程。在 call 之后再保存函数返回值、恢复栈指针、恢复 callerSaved 寄存器并重置 paraNum。为 RiscvsubroutineEmitter 添加参数 numArgs 用于记录当前正在分配寄存器的函数的参数个数，并修改 emitLoadFromStack，添加判断当 temp.index 小于 numArgs 时说明该寄存器为函数参数，直接从 A0-A7 或栈中加载。修改 nextLocalOffset 的初值，并修改 emitLabel 以存储 FP 用于参数访问的地址计算。

在 backend/reg/bruteregalloc.py 的 localAlloc 中添加 tac 的函数参数寄存器与函数 Riscv 寄存器的绑定，以防止寄存器分配机制将该寄存器分配出去导致存储值改变。

Step10：

在 frontend/ast/tree.py 中为 program 添加 globalScope 属性用于全局作用域传递。

在 frontend/parser/ply_parser.py 中修改 p_program 添加全局变量声明的匹配。

在 frontend/typecheck/namer.py 中修改 visitDeclaration，判断如果当前作用域为全局作用域且参数有初始值则需要直接将初始值添加到参数中，伴随全局作用域一起直接传递到 Riscv 生成阶段。修改 visitIdentifier 保证当前 Identifier 不是函数名。

在 frontend/symbol/varsymbol.py 为其添加 initialized 属性，表示是否为全局变量且有初始化。

在 frontend/tacgen/tacgen.py 中添加对 visitIdentifier、visitAssignment 的判断，若其符号有 temp 属性说明为局部变量，否则为全局变量，需要从 bss 或 data 中加载或存储。

在 utils/tac/tacinstr.py 中添加 Load、Store、LoadSymbol 类，表示加载某地址到寄存器或将寄存器内容存入某地址或加载某符号的地址的 tac 指令。

在 utils/tac/funcvisitor.py 中新增 visitLoadTemp、visitStoreTemp 指令，表示生成加载某地址到寄存器或将寄存器内容存入某地址对应的 tac 码。

在 utils/tac/tacprog.py 中为其添加 globalScope 属性，修改 main 完成对 globalScope 的传递。

在 utils/riscv.py 中添加 LoadSymbol 类，对应 Riscv 的 la 指令。

在 backend/riscv/riscvasmemitter.py 中为 RiscvAsmEmitter 添加 generateGlobal 方法用于依据传递过来的 globalScope 生成全局变量对应的 bss、data 段，并修改 backend/asm.py 调用该方法完成代码生成。为 RiscvAsmEmitter.selectInstr 添加 visitLoadSymbol、visitLoad、visitStore，分别根据相应的 tac 码生成对应的 Riscv 的 la、lw、sw 指令。

思考题

Step9：参数求值顺序导致不同返回值

```
int f(int x1, int x2) { return x1;}
int main(){ int a = 0; return f(a = a + 1, a = a + 2); }
```

Step9：callersave 与 calleesave

如果全部 caller save 的话，所有的函数调用的时候，不管 callee 会不会用到，对自己用到的寄存器都要写一堆 push pop 的代码，会极大程度增加代码量；由于一个函数会被多次调用，因此 callee save 方案比 caller save 好。然而一些指令运算是指定了结果的寄存器的，以及运算中大量的结果可以用完就不要了，根本.不在乎函数调用后能不能保存；如果全部都 callee save，对于自己用到的寄存器，不管 caller 有没有用到，只要 callee 用到了都要保存和恢复，也会导致开销过大。因此需要在两者间达成平衡，不全部由一方保存。

由于在离开函数、将控制权转移至父函数时子函数会调用 jr ra 会固定将子函数的返回地址（父函数运行到的位置）存入 ra 用于跳转，因此当控制权回到父函数的手中时 ra 存储的内容必然被修改为子函数的地址（即此时 pc 位置），故父函数不能要求 ra 在子函数调用前后保持不变，需要自己保存 ra。

Step10：la　v0, a 指令翻译

delta = addr(a) – pc

一种指令组合的翻译方式：auipc rd, delta[31:12]+delta[11]

addi rd,rd,delta[11:0]

另一种指令组合的翻译方式：lui rd, addr(a)[31:12]+addr(a)[11]

addi rd, rd, addr(a)[11:0]