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]