重庆大学编译原理课程实验报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **21计卓1班** | | **姓名** | **李宽宇** |
| **实验题目** | **编译器设计与实现** | | | | |
| **实验时间** | **2024.6.6** | | **实验地点** | **竹园四栋** | |
| **实验成绩** |  | | **实验性质** | **□验证性 □设计性 ■综合性** | |
| 教师评价：  □算法/实验过程正确；□源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | |
| 一、实验目的  以系统能力提升为目标，通过实验逐步构建一个将类C语言翻译至汇编的编译器，最终生成的汇编代码通过GCC的汇编器转化为二进制可执行文件，并在物理机或模拟器上运行。实验内容还包含编译优化部分，帮助深入理解计算机体系结构、掌握性能调优技巧，并培养系统级思维和优化能力。 | | | | | |
| 二、实验项目内容  本次实验将实现一个由 SysY (精简版 C 语言，来自 <https://compiler.educg.net/>) 翻译至 RISC-V 汇编的编译器，生成的汇编通过 GCC 的汇编器翻译至二进制，最终运行在模拟器 qemu-riscv 上  实验至少包含四个部分: 词法和语法分析、语义分析和中间代码生成、以及目标代码生成，每个部分都依赖前一个部分的结果，逐步构建一个完整编译器  **实验一**：词法分析和语法分析，将读取源文件中代码并进行分析，输出一颗语法树  **实验二**：接受一颗语法树，进行语义分析、中间代码生成，输出中间表示 IR (Intermediate Representation)  **实验三**：根据 IR 翻译成为汇编  **实验四(可选)**：IR 和汇编层面的优化 | | | | | |

|  |
| --- |
| **三、实验内容实现**   1. 实现哪些内容   实验一（58/58）词法分析和语法分析，将读取源文件中代码并进行分析，输出一颗语法树，具体实现如下：  词法分析：  1.1 任务：对输入的字符流进行处理，再输出词法单元(Token)流。  1.2 思路：通过一个扫描器（scanner）实现——读取输入字符串并生成一系列的tokens。用有限状态自动机（DFA）来处理输入，在某些转移过程中累计接受到的字符，在适当的时候生成token。关键是实现DFA，DFA状态转移表如下：  表1 DFA状态转移表  1.3 代码与具体实现：  (1)Token 为 TokenType 和 value 的二元组。  其中，value在DFA中累积，TokenType包含了各种可能的标记类型。每个枚举常量代表一种特定的标记类型，例如 IDENFR 代表标识符，INTLTR 代表整数字面量，FLOATLTR 代表浮点数字面量,各种运算符号和分隔符段落符也都是token。  （2）有限状态自动机（DFA）来处理输入，在某些转移过程中累计接受到的字符，在适当的时候生成token。定义DFA结构体如下：  其中，next函数是DFA的状态转移函数，也是实验一重点，实现见(4)，用cur\_state记录当前状态，cur\_str记录当前累计的字符。  （3）扫描器（Scanner），将字符依次输入dfa的next函数，当dfa发出生成信号时，产生token，进而将字符串输入转化为 Token 串。其执行函数代码如下：  （4）有限状态自动机（DFA）的状态转移函数next函数，具体实现如下，其接受参数字符 input 和当前状态，返回布尔值 tkSignal，当其为真时，就代表已经获得了一个完整的token。此外状态机一共有五种状态，代码如下  根据输入的类型和当前状态进行转移，转移表见“思路部分”，首先用swtich判断当前状态，在判断输入，下面以State::IntLiteral下的转移为例  如上图，其中包括4类合法输入，考虑到不同进制整数，在接受到a-f A-F,以及x和X时，都是合法的数字。  当前状态为 IntLiteral 的情况，也就是说，当前正在解析一个整数字面量（例如：123），如果是空白字符，表示当前数字结束。将 buf（代表当前Token的缓冲区）的类型设置为 INTLTR（整数字面量）。调用 reset() 函数复位状态。准备解析下一个Token。设置 tkSignal 为 true，表示成功解析并生成了一个Token。终止当前的 switch case。  语法分析：  1.4 任务：将vector<Token>转成一颗抽象语法树，树上的每个节点都表示源代码中的一种结构。  1.5 思路：算法主要参考了编译原理\_中科大(华保健) https://www.bilibili.com/video/BV16h411X7JY/  我们将定义一系列辅助函数 First\_XX，这些函数用于计算某个语法规则的 FIRST 集合。FIRST 集合表示从某个非终结符号出发，可以生成的第一个终结符号集合。递归下降法生成抽象语法树：递归下降解析器是一种自顶向下的解析方法，其中每个非终结符号对应一个函数。我们将从左到右解析输入的 token，逐步构建抽象语法树(AST)。如果匹配成功，则继续解析；如果匹配失败，则回退。  1.6代码与具体实现：  (1) FIRST 集合求解:采用根据表达式进行递归的方式求解，以CompUnit为例，其表达式为：    那么可能的分支有两个Decl和FuncDef，合并这两个分支的FIRST集就得到了Compunit的FIRST集  当递归到终结符时，返回tokentype集合，以Btype为例    由于Btype递归到了int 和 float，已经确定了其FIRST集， 所以其函数如下  (2) 从根节点CompUnit开始构造抽象语法树，以MulExp的解析为例：    据此，一个乘法表达式 MulExp，它由一个 UnaryExp 表达式开始，后跟零个或多个由乘法运算符 (\*)、除法运算符 (/) 或取模运算符 (%) 分隔的 UnaryExp 表达式。代码实现如下  **实验二**（58/58）接受一颗语法树，进行语义分析、中间代码生成，输出中间表示 IR (Intermediate Representation) 具体实现如下：  1.7 任务：对输入的字符流进行处理，再输出词法单元(Token)流。需要实现 Analyzer 类，完成 ir::Program::get\_ir\_program(CompUnit\*); 接口，该接口接受一个源程序语法树的根节点 Comp\*，对其进行分析，返回分析结果ir::Program。  1.8 思路：  (1)源程序代码中的各种顺序、结构信息都存储在树中，可以通过深度遍历语法树按源程序的顺序来分析源程序。  (2) 不同作用域中同名变量，为变量名加上与作用域相关的前缀（作用域编号），用一张表来存储这个作用域里所有变量的名称和类型  1.9 具体代码实现  (1) SymbolTable：scope\_stack管理所有作用域，functions存储所有函数，由于实现的没有类方法，所有函数都是全局函数，    (2) ScopeInfo，包括编号、作用域名称，用域入口根节点,block为一个作用域整体，作用域映射到符号表入口表    (3) 符号表入口，符号表用来存储程序中相关变量信息，包括类型，作用域，访问控制信息。map\_str\_ste是string类型变量名到STE的映射    (4) SymbolTable中包括以下函数等待实现：  add\_scope：输入一个新作用域，将信息记录在scope\_stack中  exit\_scope：退出作用域，弹出信息    get\_scoped\_name：获取作用域名称，为了在不同的作用域中处理相同的名称，将origin id更改为具有作用域信息的新id，在这种情况下，我们有两个变量，名称都为“a”，更改后它们将是“a”和“a\_block”    get\_operand：使用输入名称获取operand    get\_ste：使用输入的名称获得正确的ste    (5) 实验核心工作，需要实现 Analyzer 类，完成 ir::Program get\_ir\_program(CompUnit\*);接口，该接口接受一个源程序语法树的根节点 Comp\*，对其进行分析，返回分析结果 ir::Program。下面进行详细说明get\_ir\_program 的实现。  symbol\_table添加全局作用域，装载全局函数（IO库函数名称到对应库函数的映射,装载库函数），    get\_lib\_funcs是库函数的名称和函数指针的map，实现如下    从compunit开始DFS遍历AST,详细见(7)    为全局函数添加return中间表示    scope\_stack[0].table作为全局域的map<string, STE>，将全局变量从scope\_stack[0].table转移到irProgram.globalVal，将多维数组还要展平成1维，计算长度    最后返回irProgram  (6)在语义分析这一步，还需要进行进制转换，语法生成树中的整数可能是二进制、八进制、十六进制，检测数字字符串的开头，使用stoi的第3个参数即可实现进制转换，具体实现如下，    (7) 从compunit开始DFS遍历AST，就以CompUnit为例，语法生成树结构的产生是根据下面的表达式：    因此这里也是当前节点的判断子节点是否为表达式右部，如果有子节点，第一个（下标0）分析 Decl ,还要声明全局变量，也有可能是FuncDef，之后，如果子节点大于1，说明有可选部分，递归调用自身函数。    (8) analyzeStmt是整个语义分析最复杂的分析函数，不仅表达式长，而且涉及类型转换（RETURNTK），IFTK的cond涉及到短路运算，后面问题6详细讨论    其中BREAKTK和CONTINUETK，由于不知道stmt的大小和i的位置，这里写成\_\_unuse\_\_， 在WHILE中处理（检测到Operand("break"））, 并替换为goto指令    analyzeStmt整体实现大致如下    **实验三**（58/58）根据 IR 翻译成为汇编，并能在rsic-V上执行  1.10 任务:根据完成目标代码生成，得到rsic-V指令集的汇编，即从实验二的ir::Program这一入口，完成初始化全局变量（写到.data），初始化函数（先写到.globl并标记@function，再在代码段将处理ir::Function 转成汇编指令）。  1.11 思路：生成的汇编文件应 遵循 riscv ABI 规范，这样我们编译器的生成的汇编才可以使用库函数，正确的被加载，并在执行后正确的返回。从实验二的ir::Program，完成初始化全局变量，包括整型/浮点型变量/数组，在汇编可以使用 .space .word 等伪指令声明。未初始化的全局或静态变量（包括数组）放入 BSS 段。初始化函数：写函数头，清空栈、跳转表、寄存器，统计操作数，将calleeSavedReg函数的stack关联起来，存函数参数（优先把参数放到寄存器中，对于超过8个参数的情况，这些参数不会被放置在寄存器中，而是直接压入栈中），为函数中的操作数分配栈空间，逐一处理函数中的指令（处理指令时，涉及到寄存器的分配和释放，这里自行实现），最后是函数返回前还原上下文。可以先实现整数的寄存器和相关指令，再做适当修改就可以实现浮点数的寄存器和相关指令。  1.12 具体代码实现  (1)寄存器枚举类和浮点数枚举类：  整数寄存器（rvREG）整数寄存器用于存储和操作整数数据。在RISC-V架构中，通常有32个整数寄存器：  ZERO: 始终为零的寄存器，用于硬件优化和减少特例处理。  RA: 返回地址寄存器，用于存储函数调用的返回地址。  SP: 栈指针寄存器，用于指向当前栈顶。  GP: 全局指针寄存器，指向全局变量。  TP: 线程指针寄存器，指向线程局部存储。  T0-T2: 临时寄存器，短期保存临时数据。  S0-S1: 保存寄存器，保存跨函数调用的数据。  A0-A7: 函数参数寄存器，用于传递函数参数和返回值。  S2-S11: 更多保存寄存器。  T3-T6: 更多临时寄存器。  浮点寄存器（rvFREG）  浮点寄存器用于存储和操作浮点数数据。在RISC-V架构中，同样有32个浮点寄存器，每个寄存器的名字和功能如下：  F0-F7: 浮点临时寄存器，短期保存临时浮点数据。  FS0-FS1: 浮点保存寄存器，保存跨函数调用的浮点数据。  FA0-FA7: 浮点函数参数寄存器，用于传递浮点函数参数和返回值。  FS2-FS11: 更多浮点保存寄存器。  FT8-FT11: 更多浮点临时寄存器。  (2) 变量寻址，用map实现根据变量名栈中查找变量  (3) 函数调用过程通常分为以下六步  (4) 调用者将参数存储到被调用的函数可以访问到的位置，op1.name为函数名，des为函数返回值，首先将获取指令的所有参数，  存入对应的寄存器  为调用分配栈空间  此外，如果函数有返回值的话，还要存储A0和FA0寄存的值，避免丢失  (4) 跳转到被调用函数起始位置；    (5) 被调用函数获取所需要的局部存储资源，按需保存寄存器(callee saved registers)  (6) 执行函数中的指令，生成指令，就是根据操作符去调用相应的指令，用map对应指令和函数指针  (7) 将返回值存储到调用者能够访问到的位置，恢复之前保存的寄存器(callee saved registers)，释放局部存储资源；返回调用函数的位置（调整栈指针的指向）。  (8) 寄存器分配相关函数  (9) 将ir::Instruction都翻译成汇编，包括alu 运算逻辑运算，访存与指针，调用返回，goto，移位，空，cvt\_f2i cvt\_i2f  (10)以cvt\_i2f为例，首先为指令分配寄存器，汇编指令生成，保存结果，释放寄存器  **实验四(rank0.20)**：IR 和汇编层面的优化  任务：编译优化是编译器的一个重要部分，旨在改善生成的目标代码的质量和性能。通过应用各种优化技术，可以减少程序的执行时间、减少资源消耗，并提高代码的质量和可维护性。  思路：构造 SSA 形式的中间表示——确定基本块，构建控制流图，插入 φ 函数，行变量重命名，更新使用处；把寄存器当缓存用，基于数据流分析的寄存器分配算法。  方案：通过短路运算提高性能  **选择以下三个问题回答：**  如何处理数组作为参数的情况，为什么可以这样做？  如何支持短路运算？  在函数调用的过程中，汇编需要如何实现，汇编层次下是怎么控制参数传递的？是怎么操作栈指针的？   1. 如何处理数组作为参数的情况，为什么可以这样做？   ①语义分析时，数组作为函数参数，会根据参数列表的特征[]，其类型会被存储为IntPtr或者FloatPtr，二维数组还要记录第二个维度的长度，代码如下  IR->汇编时：数组的起始地址会被传递给函数，不论是整数还是浮点数，前8个参数通过a0到a7寄存器传递，剩下的参数通过栈传递。代码如下：  实际上，如果数组比较小，也可以被分解成单个元素来传递  ②在编译的语言中，数组作为参数是以数组名进行传递的，例如fun(int x[]), 其中间表示为类型为IntPtr或者FloatPtr，值是指向其第一个元素的指针，而数组在栈的存储上是连续的，且每个元素大小都是32bit（单精度浮点数和整数），有了第一个元素的地址就可以加上偏移量找到数组元素   1. 如何支持短路运算？   通常用于逻辑运算符（如 && 和 ||）的计算。当一个逻辑表达式的结果已经确定时，短路运算会立即停止后续运算，以提高效率和避免不必要的计算。  在语义分析这一步进行，这一步会生成的指令序列  对于或运算，其指令顺序应该如下  初值赋0，一旦为true就要跳转至结束——instructions.size() + 1，代码如下  对于与运算，其指令顺序应该如下  初始为1，一旦为false就要跳转至结束——instructions.size() + 1，代码如下   1. 在函数调用的过程中，汇编需要如何实现，汇编层次下是怎么控制参数传递的？是怎么操作栈指针的？   将函数地址加载进来，跳转到该地址并将返回地址保存在 ra。前8个参数通过寄存器 a0 到 a7 传递,超过8个参数的部分通过栈传递（栈顶指针+(i-8)\*4 内存地址处依次存取），浮点数存到浮点数的函数参数寄存器，。进入函数时通过加法移动栈指针，结束再移动回去。  举例说明，写了一个调用函数的demo，  其汇编文件头部要记录函数名：  然后是函数的实现，将栈指针-280，为函数分配空间，然后存储寄存器值到栈中  由于已经将fa0存入栈中，从sp偏移量100取函数参数  将返回值存储到fa0  函数执行结束，将sp移动回去，ret伪指令（jr ra）  main调用这个函数，使用伪指令call，参数前8个参数通过寄存器 a0 到 a7 传递,超过8个参数的部分通过栈传递（栈顶指针+(i-8)\*4 内存地址处依次存） |
| **四、实验测试**   1. 测试程序是如何运行的？执行了什么命令？你的汇编是如何变成RISV程序并被执行的？   （1）测试程序是如何运行的？ 测试组合了编译和执行和对比结果。  具体而言，测试有三个文件build.py，run.py，score.py，test.py可供执行：  build.py: 进入到 build 目录, 执行 cmake .. & make  run.py: 运行可执行文件 compiler 编译所有测试用例, 打印  compiler 返回值和报错, 输出编译结果至 /test/output  score.py: 将 run.py 生成的编译结果与标准结果进行对比并打分  test.py 编译生成 compiler 可执行文件，执行并生成结果，  然而在实际使用中，使用test.py最为方便s0/s1/s2分别对应实验一的两个部分和实验二，实验三S参数并不能正常使用，检查复现没写S  为了方便测试实验3，编写了一个test.sh编译生成 compiler 可执行文件，执行并生成结果，内容如下：  （2）执行了什么命令？  ①编译：首先进入 /build 若CMakeList修改后应执行 cmake 命令  1. cd /build 2. cmake .. 如果一切正常没有报错 执行make命令 3. Make。  值得注意的是，根据指导书的提示，需要在CMakLists.txt中设置CMAKE\_C\_COMPILER和CMAKE\_CXX\_COMPILER的路径。  而且cmake\_minimum\_required也需要根据报错信息自行修改，最后(VERSION 3.10)在我的WSL2的ubuntu上的docker中可以正常运行  ②执行：  1. cd /bin 2. compiler <src\_filename> [-step] -o <output\_filename> [-O1]  -step: 支持以下几种输入 s0: 词法结果 token 串  s1: 语法分析结果语法树, 以 json 格式输出  s2: 语义分析结果, 以 IR 程序形式输出  e : 执行 IR 测评机，从 xx.sy 读入源文件，重定向 xx.in 作  为 IR 程序  的标准输入，并将 IR 的标准输出输出到 <output\_filename> 中  S : RISC-v 汇编  (3) 你的汇编是如何变成RISV程序并被执行的？  实验三在执行python3 run.py S后会生成对应汇编文件，执行python3 score.py S就会生成.exe和输出文件，根据score.py，首先会执行一条命令：gcc将汇编文件生成目标代码.o,与静态链接库.a连接，-o代表了生成可执行程序  之后执行生成的可执行程序，使用qemu-riscv32.sh执行.exe，接收.in输入，将输出定向到.out文件。 |
| **五、实验总结**  实验过程中所遇到的问题及解决办法  （1）实验二浮点数，windows环境下（个人电脑环境问题），导致16进制浮点数C++一直读0  解决方案：用ubuntu拉取docker镜像后的环境，就可以正常读取16进制浮点数了。  （2）实验三中，浮点数转整数的处理是向下取整而非四舍五入  解决办法fcvt.w.s指令后写rst,保证即可。  （3）在实验三遇到out不正确，.s太长难以定位错误  解决办法：从测试用例中进一步抽取demo，执行，用fout的指针输出每条指令，使得.s阅读起来更加容易，或者直接修改.s，添加一些输出观察中间变量。  （4）16进制浮点数难以判断其值  解决方案：写一个java脚本  对实验的建议  （1）实验深入思考后，整体方案不难，难度主要在细节和实验量上，建议多出点实验细节的教程，可以发到b站视频里，理论和实践向结合，结合的桥梁不仅需要学生的思考，也需要课程关于实验的帮助。  （2）实验一有作业一和二铺垫，好评。  （3）实验量上，特别是实验三和在期末和夏令营冲突，对学生的关键发展（升学）有很大的阻碍作用，建议如果在大三下开设课程的话，减少实验量，增加学时（也就是减少别的课程时间），或者换一个学期开设，大三下由于毕业实现时间太短，各门课程太密集。  （4） 实验四虽然是可选，但课程提供帮助有限，不太符合成绩来考察课程掌握程度的，建议以后不要计入成绩，加分是支持的，毕竟能反应学生水平。  学到了很多东西，个人感觉是重大最充实，实验设计水平最高的一门课，收获很大，是可以写到简历中项目经历里的高质量实验，真心感谢老师和历届助教和师兄师姐。 |