重庆大学编译原理课程实验报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | |  | | **姓名** |  |
| **实验题目** | **编译器设计与实现** | | | | |
| **实验时间** | **2023/06/08** | | **实验地点** | **DS3301** | |
| **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 | | | | | |
| 1. 实验内容实现 2. **实现哪些内容**   实验一到实验三的所有内容，但浮点数未处理。   1. **IR库的使用，如何使用静态库链接，如何使用源代码来构建库？结合CMakelist说明。**   采用源代码构建库，那么修改CMakeList.txt如下：  # link\_directories(./lib)  # --------------------- from src ---------------------  aux\_source\_directory(./src/ir IR\_SRC)  add\_library(IR ${IR\_SRC})  aux\_source\_directory(./src/tools TOOLS\_SRC)  add\_library(Tools ${TOOLS\_SRC})  此时src下的ir和tools下面的源文件将会参与整个构建过程；  如果使用静态库链接，则修改如下：  link\_directories(./lib)  # --------------------- from src ---------------------  # aux\_source\_directory(./src/ir IR\_SRC)  # add\_library(IR ${IR\_SRC})  # aux\_source\_directory(./src/tools TOOLS\_SRC)  # add\_library(Tools ${TOOLS\_SRC})  此时，由预先编译好后生成的静态库libIR.a和libTools.a在整个项目构建的链接阶段参与构建。   1. **如何处理数组作为参数的情况，为什么可以这样做？**   首先明确：数组本身可以视为指针，下标的访问实质是指针的运算。如果是数组作为参数需要注意这样一串文法：  FuncRParams -> Exp { ',' Exp }  Exp -> AddExp  AddExp -> MulExp { ('+' | '-') MulExp }  MulExp -> UnaryExp { ('\*' | '/' | '%') UnaryExp }  UnaryExp -> PrimaryExp  PrimaryExp -> LVal  LVal -> Ident {'[' Exp ']'}  其中FuncRParams表示传入的一系列参数，数组作为参数会从LVal这个节点一路传递上去。因此在LVal节点使用getptr的ir获取指针，然后一路往上传就可以了。  然而在后端中，处理相对来说比较复杂：如果是数组作为参数，在传递参数时要先判断是全局还是局部，如果是局部，要结合sp和offset把地址算出来；全局就la传地址；如果该函数有数组作为入参，由于在函数初始化时，入参作为参数也会在运行时栈分配一个地址，那么如果来自于a0-a7这种参数寄存器，直接移动寄存器即可，如果存放在栈空间中，那么还需要用sp+offset算出指针存放的位置，然后读出来才能得到这个指针地址。   1. **如何支持短路运算？**   一共有两处文法有短路运算，分别介绍。  ①LOrExp -> LAndExp [ '||' LOrExp ]  这个文法的短路是为有一项为真则跳过剩余运算。  ②LAndExp -> EqExp [ '&&' LAndExp ]  这个文法的短路是为有一项为假则跳过剩余运算。 | | | | | |
| 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测试程序如何运行**  一共有四个python脚本：build.py, run.py, score.py, test.py.其中： build.py：一键编译整个工程 run.py：命令行接收四个选项：s0,s1,s2,S，分别用来跑词法分析，语法分析，IR以及生成汇编，生成的结果被放在output下面 score.py:将run.py生成的结果与ref相比较（利用diff工具），并给出最后得分。test.py: 前三个脚本用途之和。  **②执行了什么命令**  build.py:  os.system("cd ../build && cmake .. && make")  调用CMake工具和makefile工具来构建整个项目  run.py:  cmd = ' '.join([compiler\_path, testcase\_dir + src, step, "-o", output\_dir + fname + "." + oftype])  遍历/test/testcase下的所有用例，并调用compiler跑结果，放在output下。  score.py:  cmd = ' '.join(["diff", ref\_dir + file, output\_dir + file, '-wB']  调用diff工具比对结果并打分。  比较特殊的是S（汇编）。首先执行了这条命令：  cmd = ' '.join(["riscv32-unknown-linux-gnu-gcc", output\_dir + file, "sylib-riscv-linux.a", '-o', exec\_file])  这条命令的用处是将生成的汇编与静态库链接在一起，生成可执行文件。下一步则是运行二进制可执行文件：  if os.path.exists(input\_file):      cmd = ' '.join([cmd, "<", input\_file])  cmd = ' '.join([cmd, ">", output\_file])  这段代码重定向输入输出，生成最后的结果。  cmd = ' '.join(["diff", ref\_file, output\_file, '-wB'])  最后是调用diff工具进行结果比对。  **③汇编如何变成rv**  首先用自己的编译器生成risc-v汇编，然后和sylib-riscv-linux.a静态库链接起来，最后生成二进制可执行文件并运行。 | | | | | |
| 五、实验总结   1. 实验过程中所遇到的问题及解决办法   遇到的问题：部分样例不能通过，在实验三中有的汇编程序能够正常生成，但执行会报错，如segment fault。  解决办法：通过打印中间过程，gdb等调试工具逐步排查问题。常见的问题包括但不限于：  ①实验二中，ir测评机提示找不到源操作数，这种情况大多数是由于某处程序的类型声明错误。  ②实验二中的一大堆细节问题，比如部分测试点的数组的初始化不涵盖所有情况（如int a[10] = {1,2}）,临时变量的重复等等  ③最常见的段错误，一般都是非法访存导致的结果。   1. 对实验的建议   实验二降低一点难度，如设置前导作业。 | | | | | |