重庆大学编译原理课程实验报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2020级计科01班** | | **姓名** | **陈鹏宇** |
| **实验题目** | **编译器设计与实现** | | | | |
| **实验时间** | **2023.06.02** | | **实验地点** | **Ds3305** | |
| **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. 实现了哪些内容  * **实验一** 58/58   词法分析：词法分析阶段将源代码分解为一个个的词法单元（Token），例如关键字、标识符、运算符和常数等。结合PPT中的状态转换图，设计出有限状态机，通过定义词法规则和状态转换来识别和提取词法单元。  语法分析：语法分析阶段将词法单元序列组织成语法结构AST，根据文法定义，编写代码使用递归下降法实现LL(k)方法。需要注意的是这里并不是指LL(1)文法，由于一个节点的不同产生式的First集可能会有交集，所以我们应该向后看一位或者几位Token以此来确定是推导出哪一个式子。  如CompUnit -> (Decl | FuncDef) [CompUnit]，由于VarDecl和FuncDef都以int或者float开头，所以不能用第一个Token来判断，而可以通过判断而FuncDef的第二位token，LPARENT来进一步判断产生式。   * 实验二 57/58 不支持浮点数   实验二的IR生成阶段利用语法指导的翻译技术，深度优先遍历实验一生成的抽象语法树，并根据节点属性和动作来生成IR中间表示代码。在实现过程中，实现了类型推导和类型转换，主要是IntLiteral和Int的相互转换；实现了控制流语句的处理，需要处理分支语句和循环语句，这包括生成条件判断、条件跳转、循环变量的更新和循环终止条件的判断；实现了作用域和符号表管理，在IR生成过程中，需要正确处理变量的作用域和符号表的管理。这涉及到变量的声明、定义和使用，以及变量的生命周期和作用域规则。我考虑了常量传播优化以及特殊情况如逻辑短路和全局数组的处理。需要注意的是，全局数组声明后不需要添加alloc，添加alloc后IR执行机会使用系统调用，分配内存，导致之前已被初始化的数组内存被换为未初始化的随机数据内存。代码实现方面还有一些改进的余地，例如函数复用和优化代码结构，以提高代码的可维护性和可扩展性。   * 实验三 53/58 不支持浮点数   其实根据文档，我们能清楚的知道实验三的实现步骤，先处理全局变量program.globalVal，再遍历program.functions调用gen\_func对函数进行解析。在实现过程中，主要完成了以下内容：目标代码的定义和表示，通过官方文档了解了RISC-V指令的格式、寄存器的使用约定和指令的操作码等；寻址模式和内存访问，RISC-V采用基于加载/存储指令的内存访问模型，需要实现寻址模式和内存操作的生成，包括计算内存地址、加载和存储指令的生成，以及数组的访问；控制流指令的生成：生成RISC-V指令来支持高级语言中的分支语句和循环语句。条件跳转指令使用比较指令（如slt、sgt等）和分支指令（如beq、bne）来实现。无条件跳转指令使用j和jr等指令来实现；函数调用和参数传递：RISC-V遵循一种函数调用约定（ABI），规定了函数调用和参数传递的规则。实现函数调用和参数传递的生成，包括保存和恢复现场、参数的传递和返回值的处理。例如，将函数参数保存在特定的寄存器中、通过栈帧来管理局部变量和返回地址等。   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   在这个CMakeLists.txt文件中，使用了两种不同的方式来构建库：静态库链接和源代码构建库。   * 静态库链接   使用了link\_directories()指定了静态库文件的目录./lib。这表示编译器将在该目录下查找静态库文件。然后，通过add\_library()添加了名为jsoncpp的库，该库的源文件为./src/third\_party/jsoncpp/jsoncpp.cpp。这意味着将编译并生成名为libjsoncpp.a的库文件。同时我们自己编写的源文件即/src/front内的文件也会被打包成库为libFront.a。    .     * 源代码构建库   通过aux\_source\_directory()指定了源代码文件的目录，这里是./src/ir和./src/tools。然后，使用add\_library()分别创建了名为IR和Tools的库，并将对应目录下的源文件添加到库中。这样，便会使用由源代码构建的库而不是提供的静态库链接。    无论是静态库链接还是源代码构建库，最后都会通过target\_link\_libraries将可执行文件链接到相关库以构建最终的可执行文件。   1. 在IR中你如何处理全局变量的，这样的设计在后端有什么好处？后端中如何处理全局变量？   将全局变量的赋值初始化翻译为一条一条的IR语句，放入”global”函数中，然后再main函数的第一行调用global函数，并且将所有全局变量加入ir::Program.globalVal中，并且记录其是数组还是非数组，若是数组，记录其数组长度。  这样的设计在后端有以下好处，可维护性和可扩展性：将全局变量的初始化代码放入单独的"global"函数中，使得代码结构更清晰，方便维护和修改。将全局变量的信息记录在ir::Program.globalVal中，为后续的优化和处理提供了更多的上下文和信息；代码复用和优化，后端可以对"global"函数进行优化，例如常量传播、复制传播等，以提高代码的效率和性能；全局变量的统一管理，将全局变量的信息记录在ir::Program.globalVal中，可以方便地进行全局变量的管理和查询。后端可以根据这些信息进行针对性的优化和处理，例如减少未使用的全局变量的生成，优化全局变量的存储方式等。   1. 在函数调用的过程中，IR测评机发生了什么？   通过阅读ir\_executor.cpp，当调用call操作时，先会判断是否是库函数，如果是，则执行库函数，如果有返回值，则将返回值放入des变量中。    如果不是库函数，会先生成一个上下文，如果该函数在program中，则初始化上下文。    之后将检查该函数的合法性，如返回值类型是否一致，参数列表类型是否一致，以及上面的是否在program中。如果都合法，则将变量插入到上下文中的mem中，设置结果写入的内存地址为变量的内存地址。将该cxt加入cxt\_stack中，并且指明当前上下文cur\_ctx为cxt，下一次循环就会开始执行该函数中的IR指令，当该函数执行完毕后，cxt栈便会出栈。  .   1. 如何支持短路运算？   在实验2生成IR指令时，需支持短路运算。具体实现在LandExp文法翻译时。在分析完EqExp后，判断EqExp的值是否为0，如果为0则直接跳转该语句的最后一条指令，为1则顺序执行。  对于立即数情况，我们能直接判断并选择忽略后续表达式，对于变量情况需要添加goto语句，跳转到最后一句，而具体偏移量需要最后来判断。      评分标准：必须回答1，2-9中选择3个或以上问题进行回答，至少3个缺一项扣2分 | | | | | |
| 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？   此部分文档阐述的比较清晰。当配置完所有环境，完成某个实验的代码后（以实验二为例），可以通过该命令**compiler <src\_filename> [-step] -o <output\_filename> [-O1]**对单个用例进行调试分析，也可以直接用过run.py运行所有测试用例。其实，在实验中通常直接运行test.py，如果是实验二，则具体命令为**python test.py s2**。通过阅读这些python脚本，我们能发现test.py其实就是根据参数分别执行build.py、run.py 和 score.py。  Build.py使用CMakelist中配置的参数和make和cmake指令执行编译compiler；    Run.py执行编译好的二进制compiler程序，调用其完成相应的实验任务，并把每个输入对应的输出结果输出到./test/output文件夹下，每条命令如下：**../bin/compiler.exe ./testcase/basic/11\_add2.sy -e -o ./output/basic/11\_add2.out**  score.py则是将./output文件夹下的输出与标准输出./test/ref或./test/testcase的内容对比，调用diff程序对比，相同则该测试点通过，不同则没有通过。**diff .\ref\s2\basic\15\_mul.out .\output\basic\15\_mul.out -wB**  对于实验三，还会先调用gcc汇编器将生成的汇编文件转为二进制，然后调用qemu模拟riscv cpu执行二进制文件，最后将输出结果与标准输出做对比。  调用交叉汇编器riscv32-unknown-linux-gnu-gcc，将生成的汇编文本文件与库函数静态库链接，生成riscv32二进制文件，并调用qemu-riscv32，使用-L参数指定系统库位置，模拟riscv32执行二进制文件，生成结果。  评分标准：不回答扣2分 | | | | | |
| 五、实验总结   1. 实验过程中所遇到的问题及解决办法  * cmake -G "MinGW Makefiles" ..\命令执行失败         检查很久后发现是.h声明函数和.cpp实现函数不对应导致，很愚蠢   * Output tk与ref tk一致但是score为0，而且单独拿两个出来测试diff命令，返回值都是0，不知道为什么score.py执行时返回值为1   解决方法：当时不知原因，但是第二天电脑重启后就正常了，现在想来应该是环境变量配置后没有生效导致的   * 实验二make命令报错，根据群内答疑发现如果要自测需要取消注释，取消注释后make 命令执行成功      * Docker运行python脚本，显示没有python语法     解决方法：     * 双运算符未判断成功，且16进制数匹配错误     调试发现，由于||中的|并不属于单预算符，导致判断为Empty状态而丢失了||，需要在set中添加|和&。    对于16进制匹配错误，调试发现0x都是对的，但在x后接收b导致提前输出。是因为源代码只考虑了0后x和b的情况，而x后的字母abcdef没有考虑到，添加后该点通过，至此词法分析调试结束     * ConstDef中，最后一步如果是直接用的children的name会导致找不到ste，因为生成ste时用的时初始名字 * ConstDef中，数组声明问题，顺序不一致导致8号测试点过不去 * ConstInitVal中，如果判断ste的维度，会导致一个测试点过不去，但是判断下一个token就可以 * ConstInitVal中，改变ste的level和len后，需要将其复原，否则八号测试点过不去 * Lval需要考虑取值和存值两种情况 * Number需要分辨进制 51测试点  1. 对实验的建议   其实对于大部分同学来说，编译原理的理论知识是不足的，根据现有的文档可能很难完成当前实验。尽管看了源码，完整阅读文档，了解了每个实验该做啥，也很难下手完成实验（没有具体的理论知识框架，也不知道该从哪一步下手，没有具体的思路）。文档应该更加详细的引导实验流程，框架应用和框架搭建，最好有一定的示例代码。 | | | | | |