图标

描述已自动生成

编译原理实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| **专业** | **：** | **计算机科学与技术** |  |
| **班级** | **：** | **CS2008** |  |
| **学号** | **：** | **U202015533** |  |
| **姓名** | **：** | **徐瑞达** |  |
| **电话** | **：** | **17837353795** |  |
| **邮箱** | **：** | **2014027378@qq.com** |  |

**独创性声明**

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

特此声明！

作者签名：

日期：2023年 6 月 15 日

|  |  |
| --- | --- |
| 综合成绩 |  |
| 教师签名 |  |

目 录

[1 编译工具链的使用 1](#_Toc138333199)

[1.1 实验任务 1](#_Toc138333200)

[1.2 实验实现 1](#_Toc138333201)

[2 词法分析 4](#_Toc138333202)

[2.1 实验任务 4](#_Toc138333203)

[2.2 词法分析器的实现 4](#_Toc138333204)

[3 语法分析 6](#_Toc138333205)

[3.1 实验任务 6](#_Toc138333206)

[3.2 语法分析器的实现 6](#_Toc138333207)

[4 中间代码生成 8](#_Toc138333208)

[4.1 实验任务 8](#_Toc138333209)

[4.2 中间代码生成器的实现 8](#_Toc138333210)

[5 目标代码生成 9](#_Toc138333211)

[5.1 实验任务 9](#_Toc138333212)

[5.2 目标代码生成器的实现 9](#_Toc138333213)

[6 总结 11](#_Toc138333214)

[6.1 实验感想 11](#_Toc138333215)

[6.2 实验总结与展望 11](#_Toc138333216)

# 编译工具链的使用

## 实验任务

1. 编译工具链的使用；
2. Sysy语言及运行时库；
3. 目标平台arm的汇编语言；
4. 目标平台riscv64的汇编语言；

以上任务中(1)(2)为必做任务，(3)(4)中任选一个完成即可。

## 实验实现

1. 编译工具链使用

在该部分实验，学习使用了GCC、CLANG、交叉编译器、make等编译工具。在前三关中，根据实验提示，编写命令行即可。在编写makefile文件时，要注意严格的缩进，同时使用makefile的自动推导特性，能够大大简化makefile文件的编写。

1. Sysy语言及运行时库

Sysy语言中，没有for语句，也没有--、++、+=、-=等运算符，因此需要在C语言代码的基础上进行修改。同时，注意运行时库getint()用来读取一个整型变量、putint()用来打印一个整型变量、putch()用来打印一个字符（参数为ASCII码）。

1. arm汇编

在本关卡中，通过编写冒泡排序，了解并掌握arm汇编语言。在编写时，需要注意参数如何传递（数组arr的首地址保存在寄存器r0中，数组元素的个数n保存在寄存器r1中），返回值如何传递（返回值为0，由r0传递），立即数需要加前缀#，指令、伪指令、伪操作、寄存器名等不可以大小写混用，在进入bubblesort函数时保存现场（push {r0,r1,r2,r3,r4,r5,r6,r7,r8}），在结束bubblesort函数前恢复现场（pop {r0,r1,r2,r3,r4,r5,r6,r7,r8}）。在熟悉arm的指令集和寄存器结构后，即可正确编写代码。代码如下所示：

bubblesort:

push {r0,r1,r2,r3,r4,r5,r6,r7,r8}

mov r2,#0

sub r1,r1,1

loop1:

mov r3,#0

mov r8,r0

sub r7,r1,r2

loop2:

add r4,r3,1

ldr r5,[r8]

ldr r6,[r8,#4]

cmp r5,r6

ble addj

str r6,[r8]

str r5,[r8,#4]

addj:

add r3,r3,#1

add r8,r8,#4

cmp r7,r3

bgt loop2

add r2,r2,#1

cmp r1,r2

bgt loop1

out:

pop {r0,r1,r2,r3,r4,r5,r6,r7,r8}

mov r0,#0

bx lr

1. riscv64汇编

在本关卡中，通过编写冒泡排序，了解并掌握riscv64汇编语言。riscv64编程逻辑与arm类似，只需注意指令集、寄存器结构即可。代码如下所示：

bubblesort:

addi t0,a1,-1

loop1:

andi t1,t1,0

ori t2,a0,0

loop2:

lw t3,0(t2)

lw t4,4(t2)

ble t3,t4,addj

sw t3,4(t2)

sw t4,0(t2)

addj:

addi t1,t1,1

addi t2,t2,4

blt t1,t0,loop2

addi t0,t0,-1

blt zero,t0,loop1

# 词法分析

## 实验任务

分别在给出的语法分析器框架的基础上，实现一个Sysy语言的语法分析器：

1. 基于flex的Sysy词法分析器(C语言实现)
2. 基于flex的Sysy词法分析器(C++实现)
3. 基于antlr4的Sysy词法分析器(C++实现)

以上任务任选一个完成即可。

## 词法分析器的实现

本关卡选用基于flex的Sysy词法分析器(C语言实现)。flex源文件分为辅助定义、正则式、用户子程序三部分。按照题目要求，只需在正则式部分填写模式和动作即可。

1. 标识符ID

标识符应以大小写字母和下划线开头，并接以若干个大小写字母、下划线和数字。其对应语义动作中，打印单词和类型并返回类型ID即可。具体代码如下：

[a-z\_A-Z][a-z\_A-Z0-9]\* {printf("%s : ID\n", yytext); return ID; }

1. int型字面量INT\_LIT

int型字面量包含十进制、八进制、十六进制三种，不同进制的正则式之间以|分隔。十进制字面量以数字1-9开头，接以数字0-9；八进制字面量以0开头，接以数字0-7；十六进制字面量以0x或0X开头，接以十六进制字符（注意是正闭包）。其对应语义动作中，打印单词和类型并返回类型INT\_LIT即可。具体代码如下：

[1-9][0-9]\*|0[0-7]\*|0[xX][0-9a-fA-F]+ {printf("%s : INT\_LIT\n", yytext); return INT\_LIT; }

1. float型字面量FLOAT\_LIT

float型字面量包含大于1的浮点数、小于1的浮点数和科学计数法三种，不同表示法之间以|分隔。具体代码如下：

[0-9]+\.[0-9]+f|[0-9]+[\.]\*[0-9]\*[Ee][+-][0-9]+f {printf("%s : FLOAT\_LIT\n", yytext); return FLOAT\_LIT; }

1. 词法错误

当变量以数字开头、十进制数字含有前导0时即发生词法错误。

[0-9][a-z\_A-Z][a-z\_A-Z0-9]\*|0[8-9][0-9]\* {printf("Lexical error - line %d : %s\n",yylineno,yytext);return LEX\_ERR;}

# 语法分析

## 实验任务

分别在给出的语法分析器框架的基础上，实现一个Sysy语言的语法分析器：

1. 基于flex/bison的语法分析器(C语言实现)
2. 基于flex/bison的语法分析器(C++实现)
3. 基于antlr4的语法分析器(C++实现)

以上任务任选一个完成即可。

## 语法分析器的实现

本关卡选用基于flex/bison的语法分析器(C语言实现)。在实验中，需要将产生式转换为bison的语法规则，并为每个产生式写明语义动作以构建抽象语法树。在实现时，需要注意含有Exp的产生式需要改写为含有Exp和不含有Exp的产生式，终结符';'用相应的Token代号(SEMICOLON)取代。

在书写语义动作时，根据AST数据结构的字段，使用new\_node函数创建抽象语法树。具体而言，type属性填写为左部非终结符Stmt，int\_val填写为语句类别，float\_val填写为0，symbol填写为NULL，d\_type填写为NonType。而left、mid和right属性需要根据不同产生式填写，当有1个孩子节点时使用right，有两个孩子节点时使用left,right，当有三个孩子节点时使用left,mid,right。

具体代码如下：

Stmt: LVal ASSIGN Exp SEMICOLON { $$ = new\_node(Stmt, $1, NULL, $3, AssignStmt, 0, NULL, NonType); }

| Exp SEMICOLON { $$ = new\_node(Stmt, NULL, NULL, $1, ExpStmt, 0, NULL, NonType); }

| SEMICOLON { $$ = new\_node(Stmt, NULL, NULL, $1, BlankStmt, 0, NULL, NonType); }

| Block { $$ = new\_node(Stmt, NULL, NULL, $1, Block, 0, NULL, NonType); }

| IF LP Cond RP Stmt ELSE Stmt { $$ = new\_node(Stmt, $3, $5, $7, IfElseStmt, 0, NULL, NonType); }

| IF LP Cond RP Stmt %prec THEN { $$ = new\_node(Stmt, $3, NULL, $5, IfStmt, 0, NULL, NonType); }

| WHILE LP Cond RP Stmt { $$ = new\_node(Stmt, $3, NULL, $5, WhileStmt, 0, NULL, NonType); }

| BREAK SEMICOLON { $$ = new\_node(Stmt, NULL, NULL, NULL, BreakStmt, 0, NULL, NonType); }

| CONTINUE SEMICOLON { $$ = new\_node(Stmt, NULL, NULL, NULL, ContinueStmt, 0, NULL, NonType); }

| RETURN Exp SEMICOLON { $$ = new\_node(Stmt, NULL, NULL, $2, ReturnStmt, 0, NULL, NonType); }

| RETURN SEMICOLON { $$ = new\_node(Stmt, NULL, NULL, NULL, BlankReturnStmt, 0, NULL, NonType); }

;

# 中间代码生成

## 实验任务

在给出的中间代码生成器框架基础上完成LLVM IR中间代码的生成，将Sysy语言程序翻译成LLVM IR中间代码。

## 中间代码生成器的实现

本关卡需要实现visit()方法，用于生成StmtAST类抽象语法树的IR。

当sType=ASS时，结点类型为赋值语句，它有两个子节点：lVal和exp。其语义是将exp的值store到代表左值的变量地址。

在代码中，首先需要告诉LVal节点当前是赋值语句的左值，而不是表达式。

requireLVal = true;

然后visit(lVal)，并从recentVal取左值的Value。

ast.lVal->accept(\*this);

auto lval = recentVal;

接着visit(exp)，并从recentVal取右值的Value。

ast.exp->accept(\*this);

auto rval = recentVal;

然后需要检查赋值语句左值和右值的类型，如果必要需要进行类型转换。

if(lval->type\_->tid\_==Type::FloatTyID && rval->type\_->tid\_ ==Type::IntegerTyID){

rval=builder->create\_sitofp(rval,FLOAT\_T);

}else if(lval->type\_->tid\_==Type::IntegerTyID && rval->type\_->tid\_ ==Type::FloatTyID){

rval=builder->create\_sitofp(rval,INT32\_T);

}

最后调用create\_store生成store指令即可。

builder->create\_store(lval,rval);

# 目标代码生成

## 实验任务

在给出的代码框架基础上，将LLVM IR中间代码翻译成指定平台的目标代码：

1. 基于LLVM的目标代码生成(ARM)
2. 基于LLVM的目标代码生成(RISCV64)

以上任务任选一个完成即可。

## 目标代码生成器的实现

本关卡选用基于LLVM的目标代码生成(RISCV64)。需要完善的代码分为三部分，分别是初始化目标、指定目标平台、初始化addPassesToEmitFile()的参数。

// 补充代码1 - 初始化目标

InitializeAllTargetInfos();

InitializeAllTargets();

InitializeAllTargetMCs();

InitializeAllAsmParsers();

InitializeAllAsmPrinters();

// 补充代码2 - 指定目标平台

auto target\_triple = "riscv64-unknown-elf";

module->setTargetTriple(target\_triple);

// 补充代码3 - 初始化addPassesToEmitFile()的参数，请按以下顺序

// (1) 调用getGenFilename()函数，获得要写入的目标代码文件名filename

std::string filename = getGenFilename(ir\_filename, gen\_filetype);

// (2) 实例化raw\_fd\_ostream类的对象dest。

// Flags置sys::fs::OF\_None

// 注意EC是一个std::error\_code类型的对象，你需要事先声明EC，

std::error\_code EC;

auto Flags = sys::fs::OF\_None;

raw\_fd\_ostream dest(filename, EC, Flags);

// 通常还应在调用函数后检查EC，if (EC) 则表明有错误发生(无法创建目标文件)，此时应该输出提示信息后return 1

if (EC)

{

errs() << "Could not open file: ";

return 1;

}

// (3) 实例化legacy::PassManager类的对象pass

legacy::PassManager pass;

// (4) 为file\_type赋初值。

auto file\_type = gen\_filetype == CGFT\_AssemblyFile ? CGFT\_AssemblyFile : CGFT\_ObjectFile;

# 总结

## 实验感想

本次实验紧扣理论知识，从认识基本编译工具到词法分析，从语法分析到语义计算，从中间代码生成到目标代码生成，无不加深了我对课堂知识的理解与体会。同时，本次实验的文档尤为详尽，有助于快速掌握各种编译器、机器指令架构、词法分析工具等使用方法。

## 实验总结与展望

本次实验中，我掌握了常见的编译工具，例如gcc、clang、make；了解了两种机器平台架构——arm和riscv；学会了如何使用词法分析工具flex和语法分析工具bison；掌握了语义计算、抽象语法树、中间代码生成等代码实现。

本次实验中，主要通过补全代码的形式完成关卡，难度较低，可以尝试提高关卡难度，以对编译系统有更好的认识。同时，可以设计一些关卡实现对抽象语法树的简单可视化工具加深理解。