图标

描述已自动生成

编译原理实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| **专业** | **：** | **XXXXX** |  |
| **班级** | **：** | **CS200X** |  |
| **学号** | **：** | **U2020XXXXX** |  |
| **姓名** | **：** |  |  |
| **电话** | **：** | **88888888888** |  |
| **邮箱** | **：** | **XXXXXXX@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 月 日

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

目 录

[1 编译工具链的使用 1](#_Toc137396202)

[1.1 实验任务 1](#_Toc137396203)

[1.2 实验实现 1](#_Toc137396204)

[2 词法分析 6](#_Toc137396205)

[2.1 实验任务 6](#_Toc137396206)

[2.2 词法分析器的实现 6](#_Toc137396207)

[3 语法分析 8](#_Toc137396208)

[3.1 实验任务 8](#_Toc137396209)

[3.2 语法分析器的实现 8](#_Toc137396210)

[4 中间代码生成 10](#_Toc137396211)

[4.1 实验任务 10](#_Toc137396212)

[4.2 中间代码生成器的实现 10](#_Toc137396213)

[5 目标代码生成 12](#_Toc137396214)

[5.1 实验任务 12](#_Toc137396215)

[5.2 目标代码生成器的实现 12](#_Toc137396216)

[6 总结 15](#_Toc137396217)

[6.1 实验感想 15](#_Toc137396218)

[6.2 实验总结与展望 15](#_Toc137396219)

# 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

### 编译工具链的使用

1. GCC的使用

GCC可以通过使用一些参数来对编译进行一些要求，使用gcc编译器，可以通过命令将def-test.c和alibaba.c两个源代码文件进行编译链接，并生成名为def-test的二进制可执行文件。通过在编译命令行中添加"-D BILIBILI"参数，可以定义预处理宏BILIBILI，从而使生成的程序能够形成Alibaba和Bili两个人的对话。

gcc def-test.c alibaba.c -D BILIBILI -o def-test

1. CLANG的使用

Clang可以使用"-target"参数将C源程序翻译成其他平台下的汇编代码或二进制代码，因此可以将bar.c源文件翻译成优化的armv7架构、Linux系统下符合gnueabihf嵌入式二进制接口规则，并支持arm硬浮点的汇编代码。通过指定编译选项"-S"和"-target armv7-linux-gnueabihf"，以及优化级别"-O2"，就可以生成名为bar.clang.arm.s的汇编代码文件。

       clang -S -target armv7-linux-gnueabihf bar.c -O2 -o bar.clang.arm.s

1. 交叉编译器arm-linux-gnueabihf-gcc的使用

使用交叉编译器arm-linux-gnueabihf-gcc，可以将iplusf.c源文件编译成ARM汇编代码iplusf.arm.s。然后，再次使用arm-linux-gnueabihf-gcc汇编iplusf.arm.s，并同时连接SysY2022的运行时库sylib.a，生成ARM的可执行代码iplusf.arm。最后，可以使用qemu-arm来运行iplusf.arm。

# 用arm-linux-gnueabihf-gcc 将iplusf.c编译成arm汇编代码iplusf.arm.s

arm-linux-gnueabihf-gcc -S iplusf.c -o iplusf.arm.s

# 再次用arm-linux-gnueabihf-gcc 汇编iplusf.arm.s，同时连接SysY2022的运行时库sylib.a，生成arm的可执行代码iplusf.arm

arm-linux-gnueabihf-gcc iplusf.arm.s sylib.a -o iplusf.arm

# 用qemu-arm运行iplusf.arm,iplusf.arm前有空格

qemu-arm -L /usr/arm-linux-gnueabihf/ iplusf.arm

1. make的使用

"helloworld"依赖于main.cc文件，可以使用clang++编译器将main.cc和helloworld.cc两个源文件进行编译链接，生成可执行文件"helloworld"。在编译过程中，需要指定了一些编译选项，如"-I include"用于指定头文件的搜索路径，"-Wall"用于启用警告信息，"-g"用于生成调试信息，"-O2"用于进行优化，"-std=c++17"用于指定C++标准。

objects:= main.cc helloworld.cc

helloworld: main.cc

clang++ -o helloworld -I include main.cc helloworld.cc -Wall -g -O2 -std=c++17

helloworld.cc: helloworld.hh

clang++ -c helloworld.cc

### 采用sys语言编写一个程序

1. 题目要求：

给定一个数组 prices ，它的第 i 个元素 prices[i] 表示一支给定股票第 i 个交易日的价格(假定股价是整数)。你只能选择某个交易日买入这只股票，并选择在未来的另一个交易日卖出该股票。设计一个算法来计算你所能获取的最大利润，并返回这个最大利润值。如果你不能获取任何利润，返回 0 。

1. 实现：

采用贪心算法。在模拟买股票的过程中，按照时间顺序对交易值进行遍历，在遍历的时候需要记录当前的较小交易值是多少，因为如果后面有较大的交易值时，在较小的交易值买入较大的交易值卖出的收益最大。在记录之后如果遇到大的交易值就更新收益。代码如下所示：

int prices[N];

// 请完成函数maxProfit(),其输入为股价数组，输出为可获得的最大利润

int maxProfit(int prices[]){

// ---------- 开始

int res=0;

int min=prices[0];

int i=1;

while(i<N){

if(prices[i]<min){

min=prices[i];

}

else{

if(prices[i]-min>res) res=prices[i]-min;

}

i=i+1;

}

return res;

// ----------- 结束

}

### arm汇编实现冒泡排序

1. 冒泡排序的思想：
2. 从数组的第一个元素开始，依次比较相邻的两个元素。
3. 如果前一个元素大于后一个元素，则交换这两个元素的位置，使得较大的元素向后移动。
4. 继续向后比较相邻的元素，重复执行步骤2。
5. 完成一轮比较后，最大的元素将移动到数组的末尾。
6. 重复执行步骤1至步骤4，但每次比较的元素数量减少1，直到所有元素都排好序。
7. 汇编实现

在使用汇编时，main调用bubblesort之前，将数组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。bubblesort返回值为0，由r0传递（调用者借返回值 给循环变量赋初值 ）。这个过程中会用到一些寄存器，因此需要在开始的时候就将这些值保存到栈中。之后再进行代码实现，实现过程就是上述算法的描述，在下面代码中，r2就是外层循环的计数，r3就是内层循环的计数，其他的寄存器保存地址或者值。代码如下所示：

.arch armv7-a

.text

.global bubblesort

.syntax unified

.thumb

.thumb\_func

bubblesort:

push {r0,r1,r2,r3,r4,r5,r6,r7} @保护寄存器

sub r2,r1,#1

extern\_loop: @外层循环

mov r3,#0

mov r4,r0

inner\_loop: @内层循环

@add r5,r4,#4

ldr r6,[r4]

ldr r7,[r4,#4]

cmp r6,r7

ble skip\_swap

@进行交换

str r6,[r4,#4]

str r7,[r4]

skip\_swap:

add r3,r3,#1

add r4,r4,#4

cmp r3,r2

blt inner\_loop

sub r2,r2,#1

cmp r2,#0

bgt extern\_loop

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

mov r0,#0

bx lr

### riscv汇编实现冒泡排序

冒泡排序的思想在1.2.3已经阐述过了，riscv与arm的主要区别就是使用的指令不一样，整体还是一样的。

.text

.align 1

.globl bubblesort

.type bubblesort, @function

bubblesort:

addi t0,a1,-1

extern\_loop:

andi t1,t1,0

ori t2,a0,0

inner\_loop:

lw t3,0(t2)

lw t4,4(t2)

ble t3,t4,skip\_swap

sw t3,4(t2)

sw t4,0(t2)

skip\_swap:

addi t1,t1,1

addi t2,t2,4

blt t1,t0,inner\_loop

addi t0,t0,-1

blt zero,t0,extern\_loop

.L2:

li a0,0

ret

.size bubblesort, .-bubblesort

# 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。这里我选择的是任务（1）。

## 词法分析器的实现

对①标识符ID②int型字面量INT\_LIT③float型字面量FLOAT\_LIT⑤词法错误这四种规则进行描述。

1. 标识符

标识符以字母或下划线开头，后跟字母、数字或下划线的字符序列。因此其正则表达式是开头是字母/下划线，后面是合法字符的闭包。当识别到这样的字符序列时，将其输出为ID类型的token，并返回对应的种别码。

1. Int

整型数字是以非零数字开头的数字序列、以0开头的八进制数字序列或以0x或0X开头的十六进制数字序列。十进制第一位是数字1-9的一个，后面是数字的闭包；八进制第一位是0，后面是数字0-7的闭包；十六进制在前两位后面是数字与前六位大小写字母的正闭包；其中十进制中的0包含在八进制中。当识别到这样的数字序列时，将其输出为INT\_LIT类型的token，并返回对应的种别码。

1. Float

浮点数包括小数点形式和科学计数法形式。小数点形式中小数点左右都是数字的正闭包；科学计数法形式E/e左边是小数点形式或者整数形式，右边是正负整数的正闭包。当识别到这样的浮点数时，将其输出为FLOAT\_LIT类型的token，并返回对应的种别码。

1. 词法错误

词法错误的情况包括数字开头做标识符的情况，或者以0开头的超出八进制范围的数字。当识别到词法错误时，输出相应的错误信息，并返回词法错误类型的种别码。

代码如下：

/\*标识符、整数、浮点数\*/

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

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

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

/\*出错处理\*/

[0-9][a-z\_A-Z][a-z\_A-Z0-9]\*|0[0-9]\*[8-9a-fA-F][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。这里我选择的是任务（1）。

## 语法分析器的实现

1. 要求：

第一关要求任给一个SysY2022语言的源程序，能识别并定位源程序中的语法错误。第二关要求输出语法树

1. 实现

完善parser.y的语法规则和语义计算规则，实现语法检查和语法分析。只要正确写出文法的产生式(配上语义动作：{$$ = NULL;})即可通过语法检查，正确写出创建AST的语义动作才能完成语法分析，并生成AST。

1. 第一关：

使用$$表示当前产生式的结果，$1、$2等表示产生式中的相应符号。通过在每个产生式中添加适当的语义动作，我们可以进行语法检查，并在需要时生成相关的语法树节点。代码如下：

//第一关

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,NULL,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 { $$ = 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);}

;

第二关：

通过在语义动作中调用new\_node函数来创建AST节点，并将其作为产生式的结果。这样可以根据语法规则构建出一棵完整的语法树。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,NULL,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 { $$ = 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

生成中间代码需要做的工作是：

1. 在visit(lVal)之前通过全局临时变量，传递信息，表示当前的lVal是赋值语句左值，不是表达式。

2. 在visit(lVal)之后从recentVal取左值的Value(一个Value类的对象)，在visit(exp)后，从recentVal取右值的Value(也是一个Value类的对象)。

3. 检查赋值语句左值和右值的类型，必要时作类型转换。LLVM IR是强类型语言，所有指令的两个操作数必须是同一类型。因此需要①当左值为指向i32的指针而右值是一float类型的值时，右值应当转为i32。②当左值为指向float的指针而右值是一个int类型的值时，右值应当转为float。类型转换采用对应的方法：IRStmtBuilder::create\_fptosi(Value \*val, Type \*ty) 。

5. 调用IRStmtBuilder::create\_store(Value \*val, Value \*ptr)方法生成store指令。赋值语句的左值是一个指针(变量的地址)，右值是一个值，类型为i32或float。

case ASS: {

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 代码填写处

requireLVal = true; //表示当前的lVal是赋值语句左值，不是表达式。

ast.lVal->accept(\*this);//取左值的Value

auto var = recentVal;

is\_single\_exp = true;//作为单独的exp语句出现，形如 "exp;"

ast.exp->accept(\*this);//取右值的Value

auto expval = recentVal;

if (var->type\_->tid\_ == Type::FloatTyID && expval->type\_->tid\_ == Type::IntegerTyID)

{

//当左值为指向float的指针而右值是一个int类型的值时，右值应当转为float。

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

}

else if (var->type\_->tid\_ == Type::IntegerTyID && expval->type\_->tid\_ == Type::FloatTyID)

{

//当左值为指向i32的指针而右值是一float类型的值时，右值应当转为i32。

expval = builder->create\_fptosi(expval, INT32\_T);

}

builder->create\_store(expval, var);//创建语句

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 代码结束

break;

}

# 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。这里我完成了任务（1）和（2）。

## 目标代码生成器的实现

1. 要求：生成指定平台上可以运行的目标代码
2. 实现：

①使用parseIRFile函数将源文件解析为一个Module对象。如果解析失败，打印错误信息并返回false。

②初始化 LLVM 的目标模块、目标机器等相关组件，调用Initialize函数，用于初始化目标机器、目标汇编解析器、目标汇编打印器等。

③指定了目标平台，对于 ARM 平台，使用armv7-unknown-linux-gnueabihf作为目标三元组；对于 RISC-V 平台，使用riscv64-unknown-elf作为目标三元组。然后，通过setTargetTriple函数将目标三元组设置到Module对象中。

④创建一个TargetMachine对象，并设置相关的选项和参数。

⑤完成addPassesToEmitFile函数的初始化参数。首先，调用getGenFilename函数获得要写入的目标代码文件名。然后，使用raw\_fd\_ostream类创建了一个输出流对象dest，用于将目标代码写入文件。同时创建了一个legacy::PassManager对象pass，用于管理需要应用于模块的优化和转换的传递器。最后，为file\_type赋初值，即生成文件的类型。

⑥在完成了参数的初始化后，调用TheTargetMachine->addPassesToEmitFile函数，将相关的传递器添加到pass中。如果添加失败，打印错误信息并返回false。

⑦运行pass中的传递器，将优化后的模块写入目标文件，刷新并关闭目标代码文件，并返回true表示代码生成成功。

1. Arm平台代码：

// 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 = "armv7-unknown-linux-gnueabihf";

// 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

auto filename = getGenFilename(ir\_filename,gen\_filetype);

// (2) 实例化raw\_fd\_ostream类的对象dest。构造函数：

// raw\_fd\_ostream(StringRef Filename, std::error\_code &EC, sys::fs::OpenFlags Flags);

// 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，

// 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

std::error\_code EC;

raw\_fd\_ostream dest(filename, EC, sys::fs::OF\_None);

if(EC){

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;

1. riscv平台代码

// 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";

/\*修改处\*/

auto cpu = "generic-rv64";

// 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

auto filename = getGenFilename(ir\_filename,gen\_filetype);

// (2) 实例化raw\_fd\_ostream类的对象dest。构造函数：

// raw\_fd\_ostream(StringRef Filename, std::error\_code &EC, sys::fs::OpenFlags Flags);

// 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，

// 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

std::error\_code EC;

raw\_fd\_ostream dest(filename, EC, sys::fs::OF\_None);

if(EC){

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;

# 总结

## 实验感想

## 实验总结与展望