華中科技大學

编译原理实验报告

院　　系　计算机科学与技术学院

专业班级　本硕博 2001 班

姓　　名　李茗畦

学　　号　U202015630

指导教师　杨茂林

2023 年 6 月 24 日

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

1.1 实验任务

本节实验需要完成以下任务：

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

本次实验选择的是目标平台 riscv64 的汇编语言。

1.2 实验实现

1.2.1 编译工具链的使用

使用如下命令使用 gcc 编译器将 alibaba.c 和 def-test.c 编译生成命名为 def-test 的可执行文件，使用-D BILIBILI 添加预处理语句 #define BILIBILI

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

使用以下命令使用 clang 编译器将 bar.c 编译成 armv7 汇编代码。其中-O2 表示编译器对程序的优化级别，-S 表示用于生成汇编代码而不是可执行文件，-target armv7-linux-gnueabihf 设置目标平台为 ARMv7 指令集的 Linux 操作系统。

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

使用以下命令使用 arm-linux-gnueabihf-gcc 将 iplusf.c 编译成 arm 汇编代码 iplusf.arm.s，之后使用 arm-linux-gnueabihf-gcc 将 iplusf.arm.s 连接 SysY2022 的运行时库 sylib.a 生成可执行文件 iplusf.arm，并在 qemu-arm 上运行该可执行文件。

```
arm-linux-gnueabihf-gcc -S iplusf.c -o iplusf.arm.s
arm-linux-gnueabihf-gcc iplusf.arm.s sylib.a -o iplusf.arm
qemu-arm -L /usr/arm-linux-gnueabihf/ ./iplusf.arm
```

最后要求编写一个 Makefile, 将 main.cc 和 helloworld.cc 编译为文件名为 helloworld 的可执行文件，其中头文件目录为./include，Makefile 如下所示，其中-I 用于指定头文件目录。

```
objects:=main.o helloworld.o
CC:=clang++
CFLAGS:=-O2
INCLUDE:=./include
helloworld:$(objects)
	$(CC) $(CFLAGS) $(objects) -o helloworld
main.o:main.cc
	$(CC) -c -I $(INCLUDE) $(CFLAGS) main.cc -o main.o
helloworld.o:helloworld.cc
	$(CC) -c -I $(INCLUDE) $(CFLAGS) helloworld.cc -o
	   helloworld.o
```

1.2.2 Sysy 语言及其运行时库

本实验较简单，需要注意的就是 Sysy 是 c 语言的一个子集，不能使用 Sysy 不支持的语法。

1.2.3 目标平台-RISCV

本实验也较简单，因此不再详细阐述实现过程。

2 词法分析

2.1 实验任务

本实验的任务为实现 Sysy 词法分析器，我实现的是基于 flex 的 Sysy 词法分析器 (C 语言实现)。需要完成识别标识符 ID、int 型字面量 INT_LIT 和 float 型字面量 FLOAT_LIT。

2.2 词法分析器的实现

首先声明以下一系列规则，分别用于匹配单个数字、单个符号、单个非零十进制数、单个非零八进制数、单个十六进制数和十六进制的前缀”0x” 和”0X”。

```
DIGIT [0-9]
LETTER [A-Za-z_]
NOZERO_DIGIT [1-9]
OCTAL_DIGIT [0-7]
HEXADECIMAL_DIGIT [0-9a-fA-F]
HEXADECIMAL_PREFIX "0X"|"0x"
```

以下规则分别用于匹配十进制常数、八进制常数、十六进制常数和 int 型常量。其中十进制常数需要保证第一个数字是非零的十进制数，八进制常数需要保证第一个数是 0，十六进制的常数需要保证最开始是十六进制的前缀。int 型常量就是十进制、八进制或者十六进制常数。

```
DECIMAL_CONST {NOZERO_DIGIT}{DIGIT}*
OCTAL_CONST 0{OCTAL_DIGIT}*
HEXADECIMAL_CONST {HEXADECIMAL_PREFIX}{HEXADECIMAL_DIGIT}*
INTEGER_CONST {DECIMAL_CONST}|{OCTAL_CONST}|{
    HEXADECIMAL_CONST}
```

使用以下规则匹配 float 型常量。其中 EXPONENT 用于匹配浮点数中的指数部分。

```
EXPONENT [eE][-+]?{DIGIT}+
FLOAT_CONST ({DIGIT}+"."{DIGIT}*{EXPONENT}?|{DIGIT}*"."{
    DIGIT}+{EXPONENT}?|{DIGIT}+{EXPONENT})[Ff]?
```

使用以下规则匹配标识符。标识符由字母或者符号组成，第一个字符需要保证是符号。

```
1 IDENTIFIER {LETTER}({LETTER}|{DIGIT})*
```

3 语法分析

3.1 实验任务

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。我实现的是一个基于 flex/bison 的语法分析器 (C 语言实现)。需要完成的语义规则包括赋值语句、if-else 语句、while 语句、break 语句、continue 语句和 return 语句等。

3.2 语法分析器的实现

本实验中我们只需要调用 new_node 构造节点即可。实现代码如下。需要注意的是 new_node 函数的 left、mid 和 right 参数，当只有两个子结点时，需要使用 left 和 right 节点，只有一个子结点时使用的是 right 节点。

```
1  Stmt : LVal ASSIGN Exp SEMICOLON { $$ = new_node(Stmt,$1,
       NULL,$3,AssignStmt,0,NULL,NonType); }
2      | SEMICOLON { $$=NULL; }
3      | Exp SEMICOLON { $$ = new_node(Stmt,NULL,NULL,$1,
         ExpStmt,0,NULL,NonType); }
4      | Block { $$ = new_node(Stmt,NULL,NULL,$1,Block,0,NULL,
         NonType); }
5      | IF LP Cond RP Stmt { $$ = new_node(Stmt,$3,NULL,$5,
         IfStmt,0,NULL,NonType); }
6      | IF LP Cond RP Stmt ELSE Stmt { $$ = new_node(Stmt,$3,
         $5,$7,IfElseStmt,0,NULL,NonType); }
7      | WHILE LP Cond RP Stmt { $$ = new_node(Stmt,$3,NULL,$5
         ,WhileStmt,0,NULL,NonType); }
8      | BREAK SEMICOLON { $$ = new_node(Stmt,NULL,NULL,NULL,
         BreakStmt,0,NULL,NonType); }
9      | CONTINUE SEMICOLON { $$ = new_node(Stmt,NULL,NULL,
         NULL,ContinueStmt,0,NULL,NonType); }
10     | RETURN SEMICOLON { $$ = new_node(Stmt,NULL,NULL,NULL,
         BlankReturnStmt,0,NULL,NonType); }
11     | RETURN Exp SEMICOLON { $$ = new_node(Stmt,NULL,NULL,
         $2,ReturnStmt,0,NULL,NonType); };
```

4 中间代码生成

4.1 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。我们需要完成的是 GenIR 的 visit 方法中的 ASS 分支。

4.2 中间代码生成器的实现

ASS 类型的结点是赋值语句，它有两个子节点：lVal 和 exp。其语义是将 exp 的值存储到代表左值的变量地址。首先将 requireLVal 赋值为 true，表示当前的 lVal 是赋值语句左值，不是表达式。接着调用 lVal 的 accept 方法取出表达式的左部，之后再用同样的方法取出表达式的右部。需要注意的是，我们需要检查赋值语句的左值和右值的类型是否匹配，两个操作数必须是同一类型。当左值是 int32 类型而右值是 float 类型时，需要将右值转换为 int32 类型；当左值是 float 类型而右值是 int32 类型时，需要将右值转换为 float 类型。最后调用 IRStmtBuilder::create_store(Value *val, Value *ptr) 方法生成 store 指令。该部分的实现代码如下所示。

```
1  requireLVal=true; // 当前lVal是赋值语句左部
2  ast.lVal->accept(*this); // 取出左部
3  auto left_val=recentVal;
4  ast.exp->accept(*this); // 取出右部
5  auto right_val=recentVal;
6  if(left_val->type_->tid_==Type::TypeID::IntegerTyID &&
      right_val->type_->tid_==Type::TypeID::FloatTyID)
7      right_val=builder->create_fptosi(right_val,left_val->
          type_);
8  else if(left_val->type_->tid_==Type::TypeID::IntegerTyID &&
       right_val->type_->tid_==Type::TypeID::FloatTyID)
9      right_val=builder->create_fptosi(right_val,left_val->
          type_);
10 builder->create_store(right_val,left_val);
```

5 目标代码生成

5.1 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。我实现的是生成平台为 RISCV64 的目标代码。

5.2 目标代码生成器的实现

首先需要指定目标三元组。在本实验中将目标平台设置为"riscv64-unknown-elf", 并初始化所有用来生成目标代码的目标。之后调用 TargetRegistry::lookupTarget 函数，通过刚刚得到的目标三元组得到一个对应的目标。

```
auto target_triple = "riscv64-unknown-elf";
module->setTargetTriple(target_triple);
InitializeAllTargetInfos();
InitializeAllTargets();
InitializeAllTargetMCs();
InitializeAllAsmParsers();
InitializeAllAsmPrinters();
```

接下来需要指定目标机器，我们只需要使用通用 CPU。

然后我们生成目标代码。调用 getGenFilename() 函数，获得要写入的目标代码文件名 filename。接下来实例化一个 raw_fd_ostream 类的对象 dest，其中 Flags 成员置 sys::fs::OF_None，实例化 legacy::PassManager 类的对象 pass。然后为 file_type 赋初值。该部分代码如下所示。

```
std::string filename=getGenFilename(ir_filename,
    gen_filetype);
std::error_code EC;
auto dest=raw_fd_ostream(filename, EC, sys::fs::OF_None);
if(EC){
    errs()<<"raw_fd_ostream error";
    return 1;
}
legacy::PassManager pass;
auto filetype=gen_filetype;
if (TheTargetMachine->addPassesToEmitFile(pass, dest,
    nullptr,filetype)) {
```

```
11      errs() << "TheTargetMachine can't emit a file of this type";
12      return 1;
13  }
14  pass.run(*module);
15  dest.flush();
```

6 总结

本次课程的实验主要包括词法分析、语法分析、语义分析、中间代码生成和目标代码生成等几部分。通过这次实验，我对编译原理中的词法分析器、语法分析器等有了一个基本的理解。对编译原理理论课程中学习到的知识进行了时间，加深了对理论课程知识的理解和掌握。