华中科技大学

编译原理实验报告

专　　业：　计算机科学与技术

班　　级：　CS2005

学　　号：　U202090063

姓　　名：　董玲晶

电　　话：　13067217235

邮　　箱：　1355532189@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 06 月 22 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

(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。

1.2 实验实现

1.2.1 GCC 编译器的使用

1. 实验目标

用 gcc 编译器，指定合适的命令行选项，编译出符合要求的二进制可执行代码。有源程序和.h 文件： def-test.c alibaba.c alibaba.h，用 gcc 编译器编译 def-test.c 和 alibaba.c, 并指定合适的编译选项，生成二进制可执行代码 def-test。执行的结果应当包括 Bilibili 的自我介绍以及 Alibaba 对 BiliBili 的喊话。

2. 实验解析

代码如下。-o def-test：将生成的可执行文件命名为 def-test；def-test.c：编译 def-test.c 源代码文件；alibaba.c：编译 alibaba.c 源代码文件；-DBILIBILI：定义预处理宏 BILIBILI，在编译过程中启用相关的条件编译指令。

```
gcc -o def-test def-test.c alibaba.c -DBILIBILI
```

1.2.2 CLANG 编译器的使用

1. 实验目标

用 clang 编译器，并指定合适的编译选项，将一个 SysY2022 语言的源程序“翻译”成一个优化的 armv7 的汇编代码。

2. 实验解析

在给定的编译指令中，源代码文件名为 bar.c，输出文件名为 bar.clang.arm.s；指定-S 选项来生成汇编代码；同时，使用-O2 选项来启用优化级别 2，以进行更高级的优化。这将对源代码进行各种优化，以提高生成的汇编代码的效率和性能。此外，使用-target armv7-linux-gnueabihf 选项来指定目标架构为 armv7，并使用 Linux GNU 工具链进行交叉编译。

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

1.2.3 交叉编译器 arm-linux-gnueabihf-gcc 和 qemu-arm 虚拟机的使用

1. 实验目标

用交叉编译器 arm-linux-gnueabihf-gcc 将源程序“翻译”成 arm 汇编代码，再将汇编代码汇编并与 Sysy2022 运行时库连接，生成 arm 可执行代码，然后用 qemu-arm 虚拟机运行 arm 可执行程序。

2. 实验解析

使用 arm-linux-gnueabihf-gcc 交叉编译器将源程序 iplusf.c 编译成 ARM 汇编代码 iplusf.arm.s；再使用 arm-linux-gnueabihf-gcc 将汇编代码 iplusf.arm.s 与 SysY2022 运行时库 sylib.a 连接，并生成 ARM 的可执行代码 iplusf.arm；最后使用 qemu-arm 虚拟机运行 ARM 可执行程序 iplusf.arm。

```
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
```

1.2.4 make 的使用

1. 实验目标

编写一个 Makefile，使用 make 完成项目的构建，为 helloworld 目标编写一条生成一个名为 helloworld 的可执行文件的规则。

2. 实验解析

首先进行变量定义，如下列代码前五行所示，分别定义了编译器的名称；编译选项，包括-c（目标文件）、-Wall 显示警告信息以及 linclude（指定头文件搜索路径）；源文件名称；源文件扩展名；目标文件名称。然后进行规则定义：all 表示构建所有的源文件和可执行文件；$(EXECUTABLE)指定生成可执行文件的规则，将目标文件列表$(OBJECTS)，通过$(CC)和$(LDFLAGS)将目标文件链接在一起生成可执行文件；.cc.o 通过$(CC)和$(CFLAGS)，将源文件编译为目标文件；clean 规则执行 rm -rf $(OBJECTS) $(EXECUTABLE)命令将删除目标文件和可执行文件。

```
CC = g++
CFLAGS = -c -Wall -Iinclude
SOURCES = main.cc helloworld.cc
OBJECTS = $(SOURCES:.cc=.o)
EXECUTABLE = helloworld

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
    $(CC) $(LDFLAGS) $(OBJECTS) -o $@

.cc.o:
    $(CC) $(CFLAGS) $< -o $@

clean:
    rm -rf $(OBJECTS) $(EXECUTABLE)
```

1.2.5 Sysy 语言与运行时库

1. 实验目标

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。

2. 实验解析

核心代码如下。在函数内部，定义了三个变量：min 用于记录遍历过程中的最低股价，max 用于记录当前的最大利润，i 用于循环遍历股价数组。使用 while 循环遍历股价数组，逐个比较更新最低股价和最大利润的值：如果当前股价 prices[i]小于 min，则更新 min 为 prices[i]；而如果当前股价减去最低股价 prices[i] - min 大于最大利润 max，则更新 max 为 prices[i] - min。

```
int maxProfit(int prices[]) {
  int min = 10001, max = 0, i = 0;
  while (i < N) {
    if (prices[i] < min) {
      min = prices[i];
    }
    if (prices[i] - min > max) {
      max = prices[i] - min;
    }
    i = i + 1;
  }
  return max;
}
```

1.2.6 RISC-V 汇编

1. 实验目标

用 RISCV 汇编编写一个按升序对数组进行排序的 arm 汇编程序。

2. 实验解析

使用两层循环，外层循环 loop1 控制比较的起始位置，内层循环 loop2 遍历当前位置后面的元素，并进行比较和交换操作。其它细节见代码中的注释部分。

```
        .text
        .align   1
        .globl   bubblesort
        .type    bubblesort, @function
bubblesort:
        addi sp, sp, -20
        sw ra, 16(sp)
        sw s3, 12(sp)
        sw s4, 8(sp)
        li s4, 0          # i = 0
loop1:
        bge s4, a1, exit1
        addi s3, s4, 1
loop2:
        slli t0, s4, 2   # 左移 2 位，计算偏移（i）
        add t0, a0, t0   # 加上 arr 首地址
        lw t1, 0(t0)     # 取出 arr[i]内容
        bge s3, a1, exit3         # j>n
        slli t2, s3, 2   # 计算偏移（j）
        add t2, a0, t2   # 加上 arr 首地址
        lw t3, 0(t2)     # 取出 arr[j]内容
        ble t1, t3, exit2         #a[i]<a[j]
        # 交换
        sw t3, 0(t0)
        sw t1, 0(t2)
exit2:
        addi s3, s3, 1   # j++
        j loop2
exit3:
        addi s4, s4, 1   # i++
        j loop1
exit1:
        lw s4, 8(sp)
        lw s3, 12(sp)
        lw ra, 16(sp)
        addi sp, sp, 20
.L2:
        li       a0,0
        ret
        .size    bubblesort, .-bubblesort
```

2 词法分析

2.1 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。

2.2 词法分析器的实现

1. 实验目标

选了 C 语言实现。利用 flex 工具生成 SysY2022 语言的词法分析器，要求输入一个 SysY2022 语言源程序文件,比如 test.c，词法分析器能输出该程序的 token 以及 token 的种别。

2. 实验解析

标识符 ID 以字母或者下划线开头[a-zA-Z_]，后面可以跟着零或数个字母、数字、下划线[a-zA-Z0-9_]*。

```
[a-zA-Z_][a-zA-Z0-9_]*   { printf("%s : ID\n", yytext); }
```

Int 型字面量 INT_LIT 分为十进制、八进制、十六进制类型。对于十进制，可以匹配单个 0 或一个非 0 开头的数字序列[1-9][0-9]*；对于八进制，以 0 开头，后可以选择性跟着字符 o 或者 0，最后是一个或多个八进制数字 0~7；对于十六进制，以 0x 字符开头，后面跟着一个或多个十六进制数字 0~9 以及大小写 A~F

和 a~f。

```
[0]|[1-9][0-9]* | 0[oO]?[0-7]+ | 0x[0-9a-fA-F]+          { printf("%s : INT_LIT\n", yytext); }
```

对于以 0 开头的数字序列和以非零数字开头、后跟任意数量的数字、字母和下划线的标识符输出词法错误 Lexical error – line。

```
0[0-9]+ | [1-9][0-9]*[a-zA-Z]+[0-9a-zA-Z_]*
{ printf("Lexical error - line %d : %s\n", yylineno, yytext); }
```

Float 型字面量 FLOAT_LIT 也分为几种情况：整数部分后跟着小数点和至少一个数字[0-9]+"."[0-9]+；小数点前没有整数部分，只有小数部分[0-9]+；整数部分[0-9]+，指数部分包括 e 或 E 后面跟着选择性的正负号和一个或多个数字[eE][+-]?[0-9]+，可选的后缀 f 或 F；整数部分后跟着小数点和可选的小数部分[0-9]+"."[0-9]*，可选的指数部分，包括 e 或 E 后面跟着可选的正负号和一个或多个数字[eE][+-]?[0-9]+，可选的后缀 f 或 F 表示浮点数[fF]?。

```
[0-9]+"."[0-9]*[eE][+-]?[0-9]+[fF]? | [0-9]+"."[0-9]+ | "."[0-9]+ | [0-9]+[eE][+-]?[0-9]+[fF]?
{ printf("%s : FLOAT_LIT\n", yytext); }
```

3 语法分析

3.1 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。

3.2 语法分析器的实现

3.2.1 Sysy2022 语法检查（C 语言实现）

根据给定的语法规则、语义计算规则和 node_type 的定义把相应符号换成对应的名称，并配上简单的语义动作过测试避免出错：{$$ = NULL;}。

Stmt → LVal '=' Exp ';' | [Exp] ';' | Block

| 'if' '(' Cond ')' Stmt ['else' Stmt] | 'while' '(' Cond ')' Stmt

| 'break' ';' | 'continue' ';' | 'return' [Exp] ';'

如实验指导部分所说，对于含有类似[Exp]等可选部分的产生式可以拆分成两个产生式进行转换。特别注意，为了解决移进-规约冲突，对于 if-else 结构，通过指定%prec NOELSE 来设置优先级，这里 NOELSE 是定义的虚拟 token（但是其实在 educoder 平台上不写这个也没关系哈哈）。

代码如下。

```
Stmt: LVal ASSIGN Exp SEMICOLON {$$ = NULL;}
    | Block {$$ = NULL;}
    | SEMICOLON {$$ = NULL;}
    | Exp SEMICOLON {$$ = NULL;}
```

```
    | IF LP Cond RP Stmt ELSE Stmt {$$ = NULL;}
    | IF LP Cond RP Stmt %prec NOELSE {$$ = NULL;}
    | WHILE LP Cond RP Stmt {$$ = NULL;}
    | BREAK SEMICOLON {$$ = NULL;}
    | CONTINUE SEMICOLON {$$ = NULL;}
    | RETURN Exp SEMICOLON {$$ = NULL;}
    | RETURN SEMICOLON {$$ = NULL;};
```

3.2.2 SysY2022 语法分析（C 语言实现）

完善上一关没有完成的语法或语义计算规则，调用 new_node()函数创建抽象语法树，根据实验指导给出的 ASTNode 数据结构设置对应的参数。

赋值语句中，left 是 Lval 即右部的第一个部分$1，right 是 Exp 即右部的第三个部分$3，中间设置为 NULL，种别为 AssignStmt。只有一个孩子根据指导书就设置 right，其它设置空。具体情况不再展开赘述，代码如下。

```
Stmt: LVal ASSIGN Exp SEMICOLON {$$ = new_node(Stmt, NULL, $1, $3, AssignStmt, 0, NULL, NonType);}
    | Block {$$ = new_node(Stmt, NULL, NULL, $1, Block, 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);}
    | 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);};
```

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。

4.2 中间代码生成器的实现

1. 实验目标

选了 C++语言实现。在 genIR.cpp 文件中，实现 genIR.h 中说明的 visit()方法 void visit(StmtAST &ast) override; 中，赋值语句的翻译。

2. 实验解析

首先，根据实验文档，为了表明这是赋值语句的左值，将 requireLVal 设置为 true；设置完成后，调用 ast.lVal 指针中的 accept()方法来处理该赋值语句的左值部分，处理完后保存结果 recentVal 到临时变量。Exp 部分的处理方式也是一样的。

上述操作完成后，检查赋值语句左值和右值的类型。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；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。

最后调用 create_store()方法生成 store 指令。（感谢实验文档 orz，再次磕一个）。

```
case ASS: {
    // ******************  代码填写处
    requireLVal = true;
    ast.lVal -> accept(*this);
```

```
    Value *a = recentVal;
    is_single_exp = true;
    ast.exp -> accept(*this);
    Value *b = recentVal;

   // 类型转换操作
    if (a -> type_ -> tid_ == Type::IntegerTyID && b -> type_ -> tid_ == Type::FloatTyID)
      b = builder -> create_fptosi(b, INT32_T);
    else if (a -> type_ -> tid_ == Type::FloatTyID && b -> type_ -> tid_ == Type::IntegerTyID)
      b = builder -> create_sitofp(b, FLOAT_T);

   // 生成 store 指令
    builder -> create_store(b, a);
    // ********************  代码结束
    break;
  }
```

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：

(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。

5.2 目标代码生成器的实现

初始化目标，设置 target_triple 为的值。

```
// Initialize the target registry etc.
  // ********************************************************************
  // 补充代码 1 - 初始化目标
  InitializeAllTargetInfos();
  InitializeAllTargets();
  InitializeAllTargetMCs();
  InitializeAllAsmParsers();
  InitializeAllAsmPrinters();

  //auto target_triple = module->getTargetTriple();
  //auto target_triple = getDefaultTargetTriple();
  auto target_triple = "riscv64-unknown-elf";
  //auto target_triple = "armv7-unknown-linux-gnueagihf";
```

设置 cpu 的值。

```
// ********************************************************************
  // 补充代码 2 - 指定目标平台

  module->setTargetTriple(target_triple);
  std::string error_string;
  auto target = TargetRegistry::lookupTarget(target_triple, error_string);

  // Print an error and exit if we couldn't find the requested target.
```

```
  // This generally occurs if we've forgotten to initialise the
  // TargetRegistry or we have a bogus target triple.
  if (!target) {
    errs() << error_string;
    return 1;
  }

  auto cpu = "generic-rv64";
  // auto cpu = "";
  auto features = "";

  … …
```

代码注释说明了一切 orz。

```
// ********************************************************************************
  // 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
  // (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
  // (3) 实例化 legacy::PassManager 类的对象 pass
  // (4) 为 file_type 赋初值。

  auto filename = getGenFilename(ir_filename, gen_filetype);
  std::error_code EC;
  raw_fd_ostream dest(filename, EC, sys::fs::OF_None);
  if (EC) return 1;
  legacy::PassManager pass;
  auto file_type = gen_filetype;
```

6 总结

6.1 实验感想

这绝对是我写过的实验文档最详细的实验了！课程开始前，想象中的编译原理实验应该是比操作系统实验更抽象的课程，但是本实验的实验文档的详细程度真的震惊到我了。整体实验配合课程是很有趣的，从词法分析一步步到目标代码生成（虽然是简略版），把晦涩的课本知识和代码实践结合起来，加深了我对这门课程的理解。虽然本实验大部分的时间都花在了阅读实验文档上了，但我也从中收获到了许多，同时这种方式也减轻了本人做实验的痛苦 orz。在这里特别特别感谢杨老师和赵老师在理论课程和实验课程上给予的帮助，尤其是杨老师发布在 bilibili 的网课，在前三章实验和期末复习都给予了我很大的帮助。

6.2 实验总结与展望

在本次实验中，我熟悉了各种编译工具的使用，同时了解了 Sysy 语言及运行时库的基础语法和基础编程方法；通过灵活运用正则表达式，实现将输入的源程序文件逐个字符地进行匹配和解析，并输出对应的 token 和种别信息；同时，利用 flex 和 bison 来对任意给定的源程序，对其进行语法检查和语法分析，最终输出它的抽象语法树；实验四中，在给出的中间代码生成器框架基础上我完成对赋值语句的翻译，初步实现了对于其的 LLVM IR 中间代码的生成；最后一个实验里，在给出的代码框架基础上，我实现了将 LLVM IR 中间代码翻译成 RISC-V 平台的目标代码。通过本课程的所有实验，我深入地理解了编译器前端的工作流程和具体实现方法，将理论和实践进行了有效结合。虽然不知道未来是否能够用上，但总体是次不错的学习体验。