前注

2、3、4、5、6、8中选三个，C63 = 20种组合

实验项目内容

1、 实现哪些内容

实验一到实验三的所有内容，但浮点数未处理。

2、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。

采用源代码构建库，那么修改CMakeList.txt如下：

```
# link_directories(./lib)
# ---------------------- from src ----------------------
aux_source_directory(./src/ir IR_SRC)
add_library(IR ${IR_SRC})
aux_source_directory(./src/tools TOOLS_SRC)
add_library(Tools ${TOOLS_SRC})
```

此时src下的ir和tools下面的源文件将会参与整个构建过程；
如果使用静态库链接，则修改如下：

```
link_directories(./lib)
# ---------------------- from src ----------------------
# aux_source_directory(./src/ir IR_SRC)
# add_library(IR ${IR_SRC})
# aux_source_directory(./src/tools TOOLS_SRC)
# add_library(Tools ${TOOLS_SRC})
```

此时，由预先编译好后生成的静态库libIR.a和libTools.a在整个项目构建的链接阶段参与构建。

3.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？

使用一个叫做global()的函数进行初始化，并用GVT维护全局变量。这样做的好处，一是简化了实现，二是为后面后端的优化提供了可能的扩展。后端中，如果不做优化的话，可以简单地扫描GVT，在.data段中进行声明，然后使用global函数并调用，进行初始化。

4.在函数调用的过程中，IR测评机发生了什么？

不会，参考：IR 测评机 · GitBook

> 发生函数调用时，会发生**上下文的切换**，原来的上下文将压入函数调用栈中，函数返回时，函数调用栈会将当前上下文弹出。**上下文切换**的具体含义是指：发生函数调用时，当前 IR 执行位置和存储中的操作数将被保存下来，执行流切换到下一个函数。新的函数在执行时 IR 将从第一条开始执行、存储中没有任何操作数。当从该函数返回时，将回到当初发生函数的调用的位置继续执行，存

> 储中的操作数恢复为发生调用前保存的操作数

5.如何处理数组作为参数的情况，为什么可以这样做?

首先明确：数组本身可以视为指针，下标的访问实质是指针的运算。如果是数组作为参数需要注意这样一串文法： FuncRParams -> Exp { ',' Exp }
Exp -> AddExp AddExp -> MulExp { ('+' | '-') MulExp }
MulExp -> UnaryExp { ('*' | '/' | '%') UnaryExp }
UnaryExp -> PrimaryExp
PrimaryExp -> LVal
LVal -> Ident {'[' Exp ']'}
其中FuncRParams表示传入的一系列参数，数组作为参数会从LVal这个节点一路传递上去。因此在LVal节点使用getptr的ir获取指针，然后一路往上传就可以了。在后端中需要考虑数组是全局变量还是局部变量了。如果是全局变量，直接使用伪指令la即可；否则需要借助stackVarMap存储的偏移量和sp算出地址，然后传递数组的地址。

6.如何支持短路运算?

一共有两处文法有短路运算，分别介绍。
①LOrExp -> LAndExp ['||' LOrExp] 这个文法的短路是为有一项为真则跳过剩余运算。
②LAndExp -> EqExp ['&&' LAndExp] 这个文法的短路是为有一项为假则跳过剩余运算。

7.是否进行了寄存器分配，使用了什么方法？（实验四的内容，一般不选择）

- 没有
- 有，参见编译优化技术 · GitBook

8、 在函数调用的过程中，汇编需要如何实现，汇编层次下是怎么控制参数传递的？是怎么操作栈指针的?

整数寄存器a0-a7以及浮点数寄存器fa0-fa7，按照传参列表的顺序，根据变量类型选择寄存器，然后按照先后顺序从0到7的顺序依次放。如果参数过多，寄存器放不下的情况下，就需要放在栈空间中。此时栈指针的地址大小要减少放不下的操作数的空间大小之和，然后按照放不下的参数，从栈指针开始，按照地址大小由低到高的顺序存放。

9、是否进行IR或者后端的优化，是如何实现的?

- 没有
- 有，参见编译优化技术 · GitBook

实验测试

①测试程序如何运行

一共有四个python脚本：build.py, run.py, score.py, test.py.其中：
build.py：一键编译整个工程
run.py：命令行接收四个选项：s0,s1,s2,S，分别用来跑词法分析，语法分析，IR以及生成汇编，生成的结果被放在output下面
score.py:将run.py生成的结果与ref相比较（利用diff工具），并给出最后得分。
test.py: 前三个脚本用途之和。

②执行了什么命令

build.py:

```
os.system("cd ../build && cmake .. && make")
```

调用CMake工具和makefile工具来构建整个项目

run.py:

```
cmd = ' '.join([compiler_path, testcase_dir + src, step, "-o", output_dir +
fname + "." + oftype])
```

遍历/test/testcase下的所有用例，并调用compiler跑结果，放在output下。

score.py:

```
cmd = ' '.join(["diff", ref_dir + file, output_dir + file, '-wB']
```

调用diff工具比对结果并打分。

比较特殊的是S（汇编）。首先执行了这条命令：

```
cmd = ' '.join(["riscv32-unknown-linux-gnu-gcc", output_dir + file, "sylib-
riscv-linux.a", '-o', exec_file])
```

这条命令的用处是将生成的汇编与静态库链接在一起，生成可执行文件。下一步则是运行二进制可执行文件：

```
if os.path.exists(input_file):
    cmd = ' '.join([cmd, "<", input_file])
    cmd = ' '.join([cmd, ">", output_file])
```

这段代码重定向输入输出，生成最后的结果。

```
cmd = ' '.join(["diff", ref_file, output_file, '-wB'])
```

最后是调用diff工具进行结果比对。

③汇编如何变成rv

首先用自己的编译器生成risc-v汇编，然后和sylib-riscv-linux.a静态库链接起来，最后生成二进制可执行文件并运行。

实验总结

1、实验过程中所遇到的问题及解决办法

遇到的问题：部分样例不能通过，在实验三中有的汇编程序能够正常生成，但执行会报错，有的是段错误有的是编译器报的错。

解决办法：通过打印，gdb等调试工具逐步排查问题。曾经遇到过以下问题：

①ir_executor.cpp提示找不到源操作数，这种情况大多数是由于某处程序的类型声明错误。

②部分测试点的数组的初始化不涵盖所有情况（如int a[10] = {1,2}）

③临时变量的重复导致bug

④最常见的段错误，一般都是非法访存导致的结果。

2、对实验的建议

实验二降低一点难度，如设置前导作业。