中山大學 SUN YAT-SEN UNIVERSITY

计算机学院（软件学院）
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Compiler Design
编 译 器 构 造 实 验

Lab 0：实验准备、概述

教师：张献伟

助教：吴坎、单招文

xianweiz.github.io

DCS292, 2/21/2023

Linux Environment

- 所有的实验任务预期Linux环境下完成
  - 实现语言为C/C++
  - 需要熟悉Terminal和基本的commands，以及Vim、Emacs或其他编辑工具
  - 也可以在windows环境下完成，在提交前通过Linux环境（如WSL、docker）下的测试
  - 其他：cmake, git等
- 哪些Linux环境可以使用？
  - 虚拟机
  - 本地：Debian(推荐)/Ubuntu, RedHat/CentOS
  - 远程：通过Putty, MobaXterm等连接Linux服务器

计算机教育中缺失的一课
https://missing.csail.mit.edu/
https://www.bilibili.com/video/BV14E411J7n2/

Environment Setup

- Install docker
  - Docker Desktop: https://docs.docker.com/desktop/
  - 启动docker
- Launch container

```
docker pull wukan0621/sysu-lang
docker run \
  --name sysu-lang \
  -v "$PWD/workspace:/workspace" \
  -it wukan0621/sysu-lang \
  bash
```

将提前准备好的镜像下载到本机，镜像中已经配置好实验所需要的环境，无需另行环境配置。

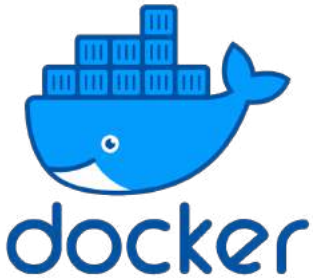
  - docker run --name sysu-lang:从镜像创建一个新的容器，名为sysu-lang
  - -v “$PWD/workspace:/workspace”: 宿主机目录映射至本地
  - -it wukan0621/sysu-lang bash: 以终端交互方式打开容器，默认shell为bash

https://docs.docker.com/engine/reference/commandline, https://github.com/arcsysu/SYsU-lang

Environment Setup (cont.)

手动配置（可跳过）

- Launch container
  - $docker pull debian
  - $docker run -it debian /bin/bash
- Configure
  - $sudo apt-get install -y --no-install-recommends \
    clang llvm-dev zlib1g-dev lld flex bison \
    cmake python3 ninja-build git
- Check versions
  - $cat /opt/os-release
  - $clang --version



https://docs.docker.com/engine/reference/commandline, https://github.com/arcsysu/SYsU-lang

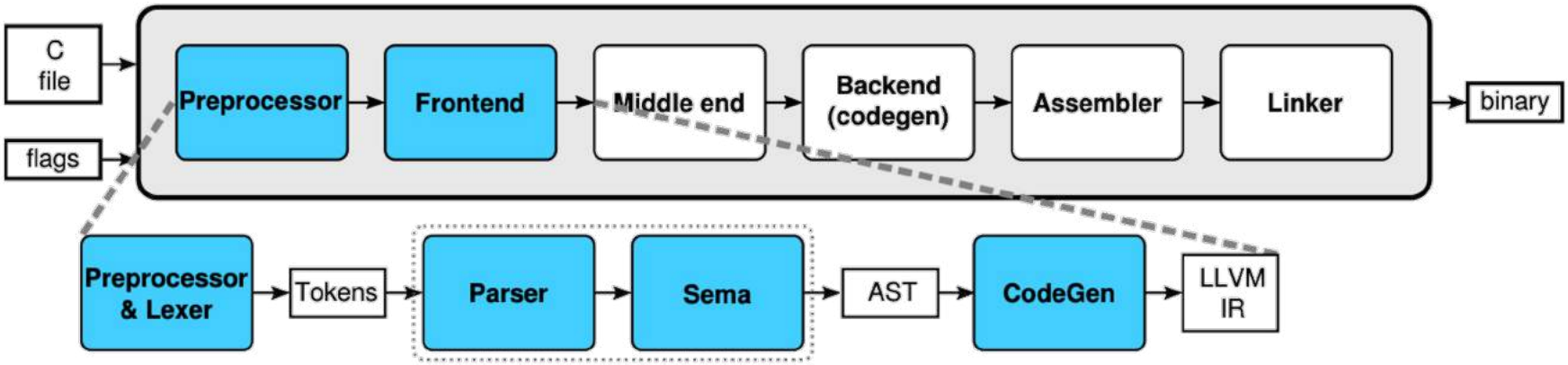
Familiarize Clang/LLVM ...[熟悉编译] clang

- Write the source code
  - $vim test0.c
- Compile the code
  - $clang -ccc-print-phases test0.c -o test

```
1  #define VAL 1
2
3  void main()
4  {
5    int arr[10], i, x = VAL;
6    float y = 0;
7
8    for (i = 0; i < 10; i++)
9      arr[i] = x * 5;
10 }
```

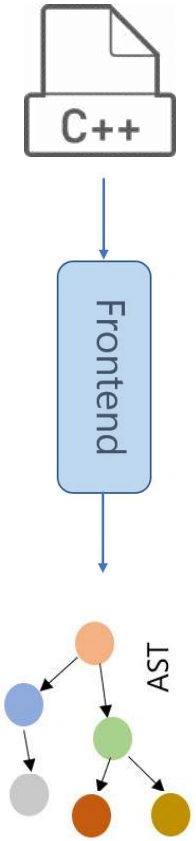
```
                +- 0: input, "test0.c", c
             +- 1: preprocessor, {0}, cpp-output
          +- 2: compiler, {1}, ir
       +- 3: backend, {2}, assembler
    +- 4: assembler, {3}, object
5: linker, {4}, image
```

  - $clang -### test0.c -o test



Clang/LLVM: Frontend

- Preprocess
  - $clang -E test0.c -o test.c

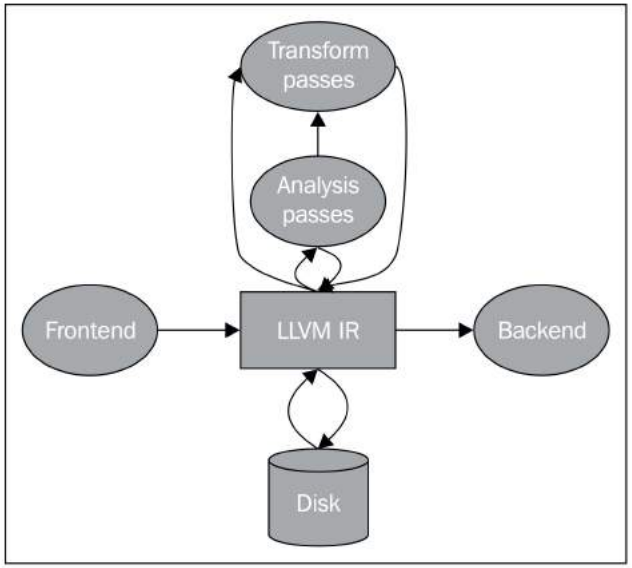


- ①Lexical Analysis
  - $clang -cc1 -dump-tokens test.c

- ②Syntax Analysis
  - $clang -Xclang -ast-dump -fsyntax-only test.c
    - $clang -Xclang -ast-dump=json -fsyntax-only test.c
- ②Semantic Analysis
  - $clang -Xclang -ast-dump -fsyntax-only test.c | \
    grep “Decl”

Clang/LLVM: IR

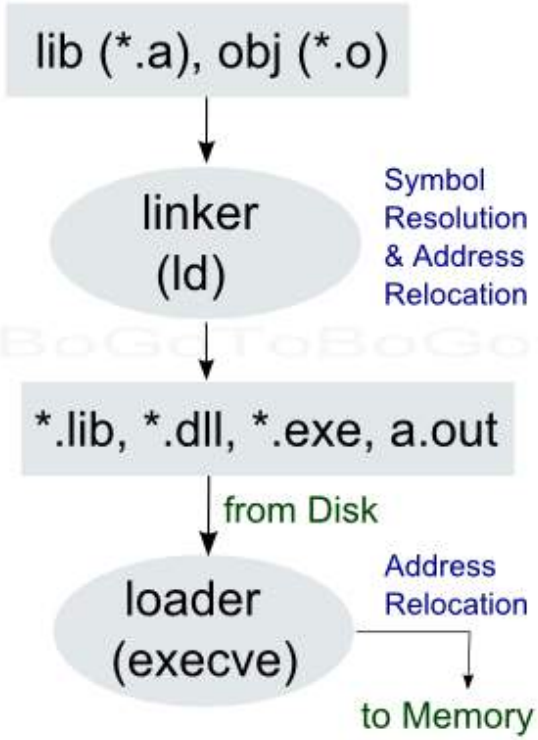
- ③IR Generation
  - $clang test.c -emit-llvm -c -o test.bc
    - $llvm-dis test.bc -o test.ll
    - $clang test.c -emit-llvm -S -c -o test.ll
    - $llvm-as test.ll -o test.bc
  - $clang -O1 test.c -emit-llvm -c -o test-O1.bc
    - $llvm-dis test-O1.bc -o test-O1.ll
    - $vimdiff test.ll test-O1.ll



- ④IR Optimization
  - $opt -O1 test.bc -o test-O1.bc --enable-new-pm
    - $opt -O1 test.ll -S -o test-O1.ll --enable-new-pm
  - $llvm-dis test-O1.bc -o test-O1.ll && vimdiff test.ll test-O1.ll
  - $opt test.ll -time-passes -mem2reg -S -o test-Om.ll --enable-new-pm
  - $opt test.ll -debug-pass=Structure -mem2reg -S -o test-Om.ll

Clang/LLVM: Code Generation

- Code Generation
  - $llc -filetype=obj test.bc -o test.o
    - -march=aarch64 : $uname -m
  - $clang test.o -o test
    - $clang -ccc-print-phases test.o -o test
    - $clang -### test.o -o test
- Extra
  - $objdump -d test.o
  - $objdump -d test
  - 👉 *What is the output difference?*

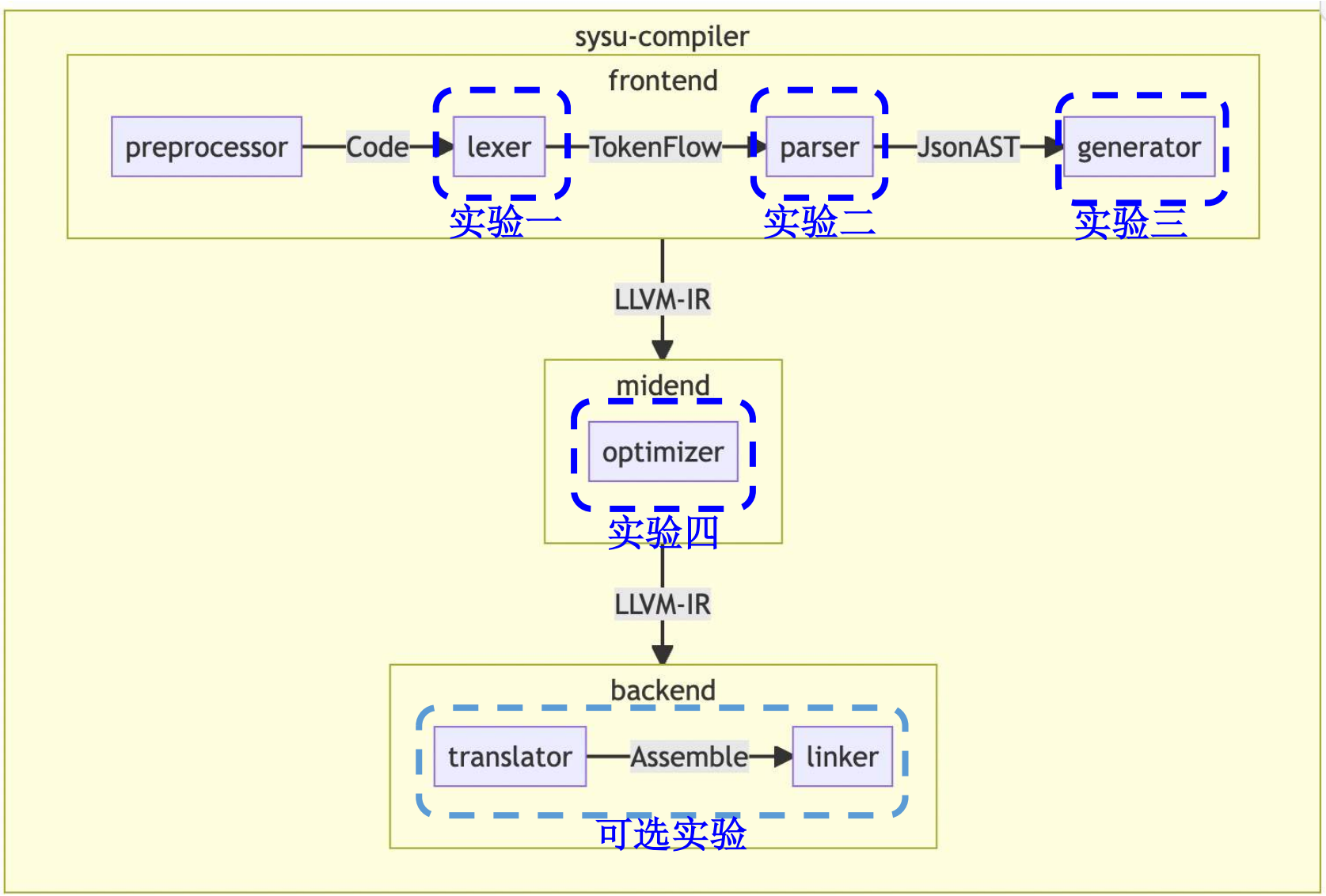


LLVM defaults to use your system linker because the LLVM linker project, lld, is currently in development and is not integrated into the core LLVM project. Therefore, if you do not have lld, you can finish the compilation by using your regular compiler driver, which will activate your system linker:

$gcc test.o -o test

LLD - The LLVM Linker , https://lld.llvm.org/

Schedule[实验安排]



Project Setup

- Clone code
  - $git clone https://github.com/arcsysu/SYsU-lang

- Build and install
  - $cd SYsU-lang/
  - $rm -rf $HOME/sysu
  - $cmake -G Ninja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \
    -DCMAKE_C_COMPILER=clang \
    -DCMAKE_CXX_COMPILER=clang++ \
    -DCMAKE_INSTALL_PREFIX=$HOME/sysu \
    -DCMAKE_PREFIX_PATH="$(llvm-config --cmakedir)" \
    -DCPACK_SOURCE_IGNORE_FILES=".git/;tester/third_party/" \
    -B $HOME/sysu/build
  - $cmake --build $HOME/sysu/build
  - $cmake --build $HOME/sysu/build -t install

Project Setup (cont.)

- Test
  - $(export PATH=$HOME/sysu/bin:$PATH \
    CPATH=$HOME/sysu/include:$CPATH \
    LIBRARY_PATH=$HOME/sysu/lib:$LIBRARY_PATH \
    LD_LIBRARY_PATH=$HOME/sysu/lib:$LD_LIBRARY_PATH &&
    sysu-compiler -S -o a.S tester/functional/000_main.sysu.c
    && **clang -O0 -lsysy -lsysu -o a.out a.S** && ./a.out)

000_main.sysu.c

```
1 int main(){
2     return 3;
3 }
```

sysu-compiler

① lexer
② parser
③ generator
④ optimizer

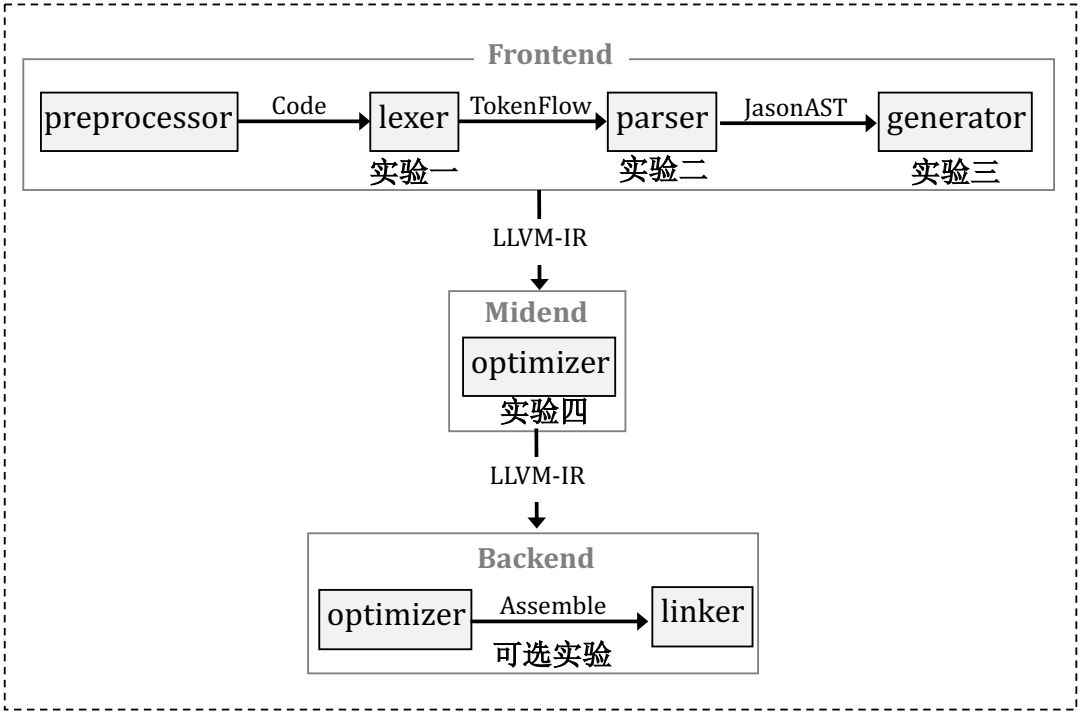
a.S

```
1         .text
2         .file   "-"
3         .globl  main                                    // -- Begin function main
4         .p2align        2
5         .type   main,@function
6 main:                                                   // @main
7         .cfi_startproc
8 // %bb.0:                                               // %entry
9         mov     w0, #3
10        ret
11 .Lfunc_end0:
12        .size   main, .Lfunc_end0-main
13        .cfi_endproc
14                                                        // -- End function
15 _      .section        ".note.GNU-stack","",@progbits
```

clang → executable

https://github.com/arcsysu/SYsU-lang

Project Overview



实验任务量及时间安排

实验	内容	代码量(行数/LOC)	用时(小时/h)	预留时间(天/d)
一	词法分析器/lexer	250	2 - 6	14
二	语法分析器/parser	1000	24 - 72	30
三	IR 生成器/generator	1500	36 - 108	30
四	IR 优化器/optimizer	开放性实验，推荐值为 500	24 - 108	30

References

- Getting Started: Building and Running Clang, https://clang.llvm.org/get_started.html
- SYsU-lang, https://github.com/arcsysu/SYsU-lang
- Getting Started with LLVM Core Libraries, https://faculty.sist.shanghaitech.edu.cn/faculty/songfu/course/spring2018/CS131/llvm.pdf
- LLVM Tutorial, https://llvm.org/docs/tutorial/
- Learn LLVM 12, https://github.com/xiaoweiChen/Learn-LLVM-12
- Tutorial: Creating an LLVM Backend for the Cpu0 Architecture ，http://jonathan2251.github.io/lbd/about.html