

四、中间代码生成

(10. LLVM IR 简介)

魏恒峰

hfwei@nju.edu.cn

2024 年 04 月 26 日



Chris Lattner @ Homepage



lattner @ GitHub

<https://llvm.org/>

The LLVM Compiler Infrastructure



p:

W
S
at
ion
Gu
ide
ons

LLVM Overview

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Despite its name, LLVM has little to do with traditional virtual machines. The name "LLVM" itself is not an acronym; it is the full name of the project.

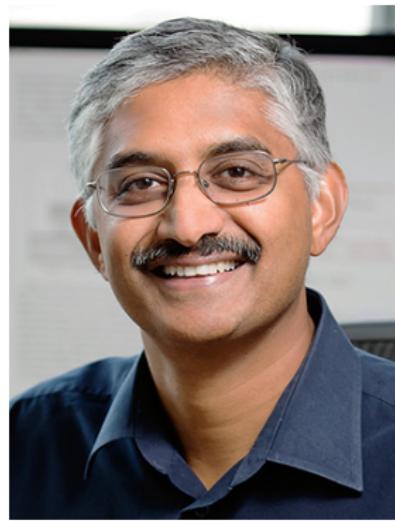
Latest LLVM Release!

17 Apr 2024: LLVM 18.1.4 is now available for download! LLVM is publicly available under an open

“Low Level Virtual Machine”



Chris Lattner (1978); UIUC 2000



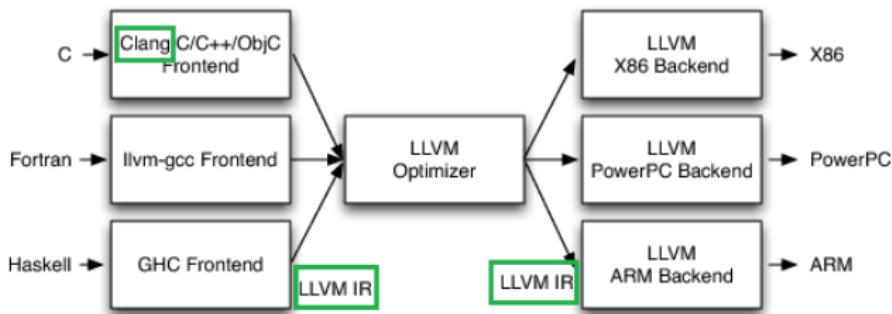
Vikram Adve (1966)



ACM Software System Award (历年获奖名单)



LLVM IR (Intermediate Representation)

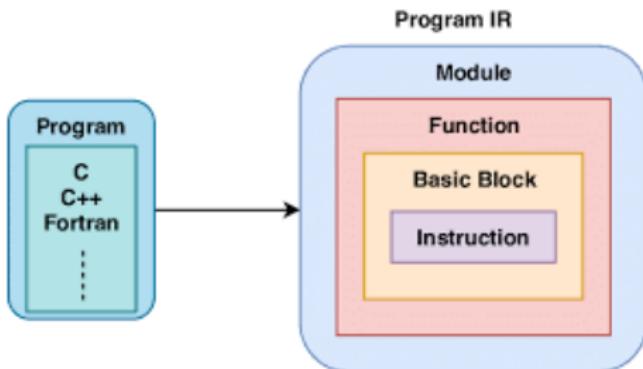


“IR 设计的优秀与否决定着整个编译器的好坏”



8 章技术内容, 其中 4 章介绍 Maple IR, 另外 4 章基于 Maple IR

LLVM Language Reference Manual



IR: Intermediate Representation

LLVM IR: 带类型的、介于高级程序设计语言与汇编语言之间
(LLVM Assembly Language)

"TALK IS
CHEAP.
SHOW ME THE
CODE."
-LINUS TORVALDS



顺序语句 (函数调用)、选择语句、循环语句

factorial0.c @ Compiler Explorer

```
int factorial(int val);

int main(int argc, char **argv) {
    return factorial(val: 2) * 7 == 42;
}
```

```
6 ; Function Attrs: noinline nounwind optnone uwtable
7 define dso_local i32 @main(i32 noundef %0, i8** noundef %1) #0 {
8     %3 = alloca i32, align 4
9     %4 = alloca i32, align 4
10    %5 = alloca i8**, align 8
11    store i32 0, i32* %3, align 4
12    store i32 %0, i32* %4, align 4
13    store i8** %1, i8*** %5, align 8
14    %6 = call i32 @factorial(i32 noundef 2)
15    %7 = mul nsw i32 %6, 7
16    %8 = icmp eq i32 %7, 42
17    %9 = zext i1 %8 to i32
18    ret i32 %9
19 }
```

```
clang -S -emit-llvm -fno-discard-value-names
factorial0.c -o f0-opt0.ll
```

Three Address Code (TAC)

```
6 ; Function Attrs: noinline nounwind optnone uwtable
7 define dso_local i32 @main(i32 noundef %0, i8** noundef %1) #0 {
8     %3 = alloca i32, align 4
9     %4 = alloca i32, align 4
10    %5 = alloca i8**, align 8
11    store i32 0, i32* %3, align 4
12    store i32 %0, i32* %4, align 4
13    store i8** %1, i8*** %5, align 8
14    %6 = call i32 @factorial(i32 noundef 2)
15    %7 = mul nsw i32 %6, 7
16    %8 = icmp eq i32 %7, 42
17    %9 = zext i1 %8 to i32
18    ret i32 %9
19 }
```

```
clang -S -emit-llvm -fno-discard-value-names
factorial0.c -o f0-opt0.ll
```

Three Address Code (TAC)

Static Single Assignment (SSA)

```
6 ; Function Attrs: noinline nounwind optnone uwtable
7 define dso_local i32 @main(i32 noundef %0, i8** noundef %1) #0 {
8     %3 = alloca i32, align 4
9     %4 = alloca i32, align 4
10    %5 = alloca i8**, align 8
11    store i32 0, i32* %3, align 4
12    store i32 %0, i32* %4, align 4
13    store i8** %1, i8*** %5, align 8
14    %6 = call i32 @factorial(i32 noundef 2)
15    %7 = mul nsw i32 %6, 7
16    %8 = icmp eq i32 %7, 42
17    %9 = zext i1 %8 to i32
18    ret i32 %9
19 }
```

```
clang -S -emit-llvm -fno-discard-value-names
factorial0.c -o f0-opt0.ll
```

mem2reg

```
6 ; Function Attrs: nounwind uwtable
7 define dso_local i32 @main(i32 %0, i8** nocapture readnone %1)
8     %3 = call i32 @factorial(i32 2) #2
9     %4 = mul nsw i32 %3, 7
10    %5 = icmp eq i32 %4, 42
11    %6 = zext i1 %5 to i32
12    ret i32 %6
13 }
```

```
clang -S -emit-llvm factorial0.c -o f0-opt1.ll -O1 -g0
```

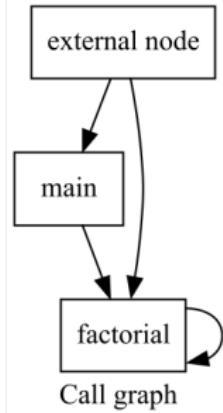
```
int factorial(int val);

int main(int argc, char **argv) {
    return factorial(val: 2) * 7 == 42;
}

// precondition: val is non-negative
int factorial(int val) {
    if (val == 0) {
        return 1;
    }

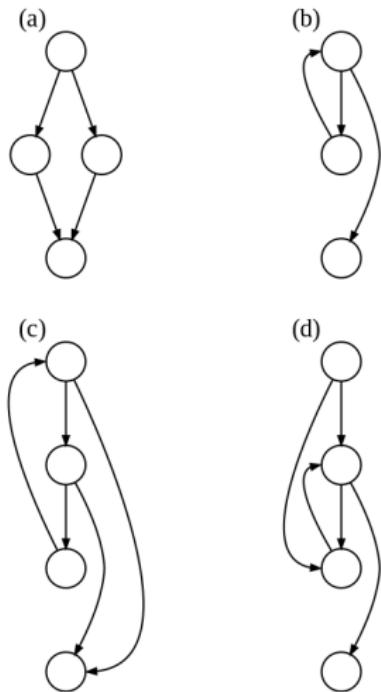
    return val * factorial(val: val - 1);
}
```

factorial1.c





Frances Elizabeth Allen
(1932 ~ 2020; 2006 Turing Award)



(Intra-procedure) Control Flow Graph (CFG)

Control Flow Graph (CFG)

Definition (CFG)

Each **node** represents a *basic block*, i.e. a straight-line code sequence with no **branches/jumps** in except to the **entry point** and no **branches/jumps** out except at the **exit point**.

Control Flow Graph (CFG)

Definition (CFG)

Each **node** represents a *basic block*, i.e. a straight-line code sequence with no **branches/jumps** in except to the **entry point** and no **branches/jumps** out except at the **exit point**.

Jump targets start a block, and jumps end a block.

Control Flow Graph (CFG)

Definition (CFG)

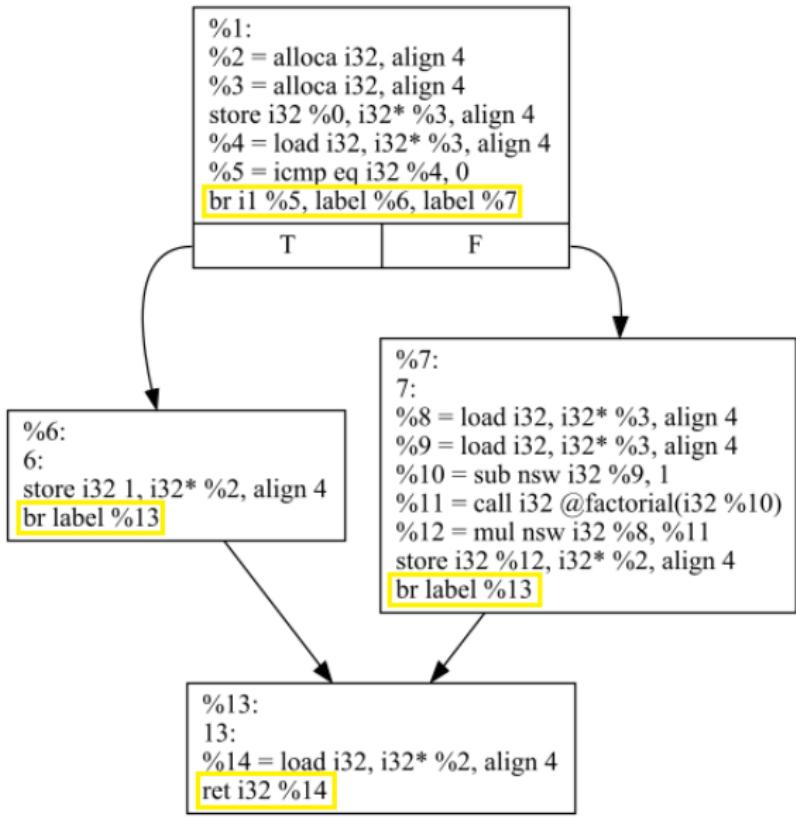
Each **node** represents a *basic block*, i.e. a straight-line code sequence with no **branches/jumps** in except to the **entry point** and no **branches/jumps** out except at the **exit point**.

Jump targets start a block, and jumps end a block.

Directed **edges** are used to represent jumps in the control flow.

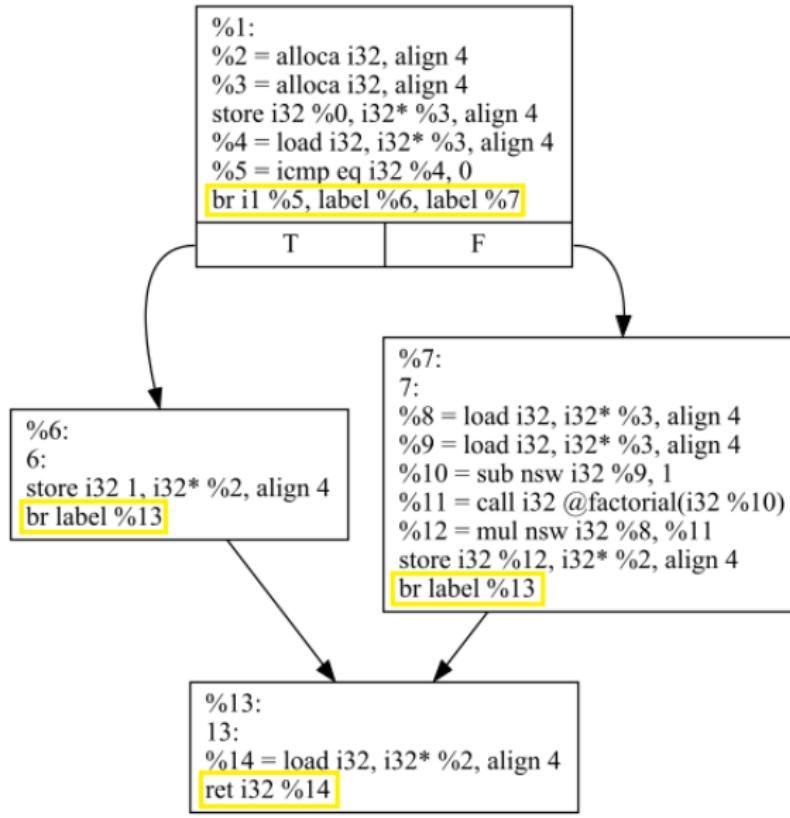
factorial1.c @ Compiler Explorer

```
int factorial(int val) {  
    if (val == 0) {  
        return 1;  
    }  
  
    return val * factorial(val: val - 1);  
}
```



CFG for 'factorial' function

%2: store the return value (in different branches)



CFG for 'factorial' function

Instruction Reference

- Terminator Instructions

- ‘ret’ Instruction
- ‘br’ Instruction
- ‘switch’ Instruction
- ‘indirectbr’ Instruction
- ‘invoke’ Instruction
- ‘callbr’ Instruction
- ‘resume’ Instruction
- ‘catchswitch’ Instruction
- ‘catchret’ Instruction
- ‘cleanupret’ Instruction
- ‘unreachable’ Instruction



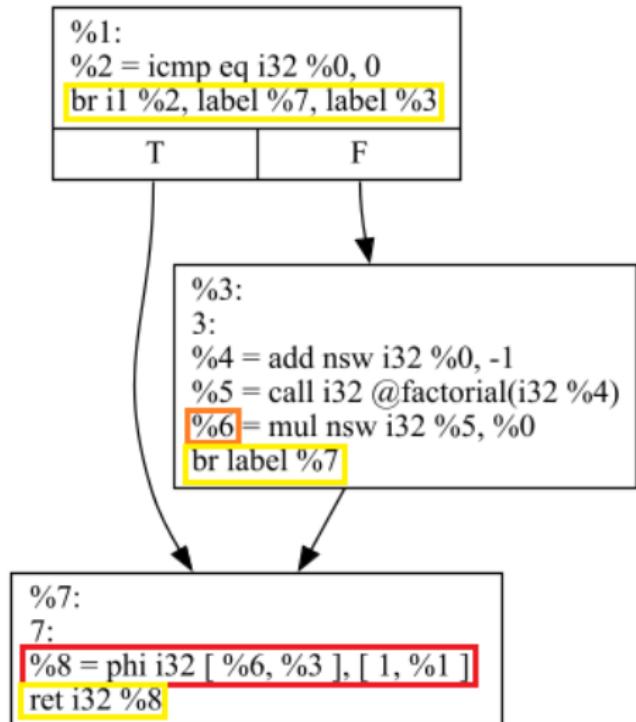
为什么基本块的中间某条指令可以是 call 指令？



为什么基本块的中间某条指令可以是 call 指令？

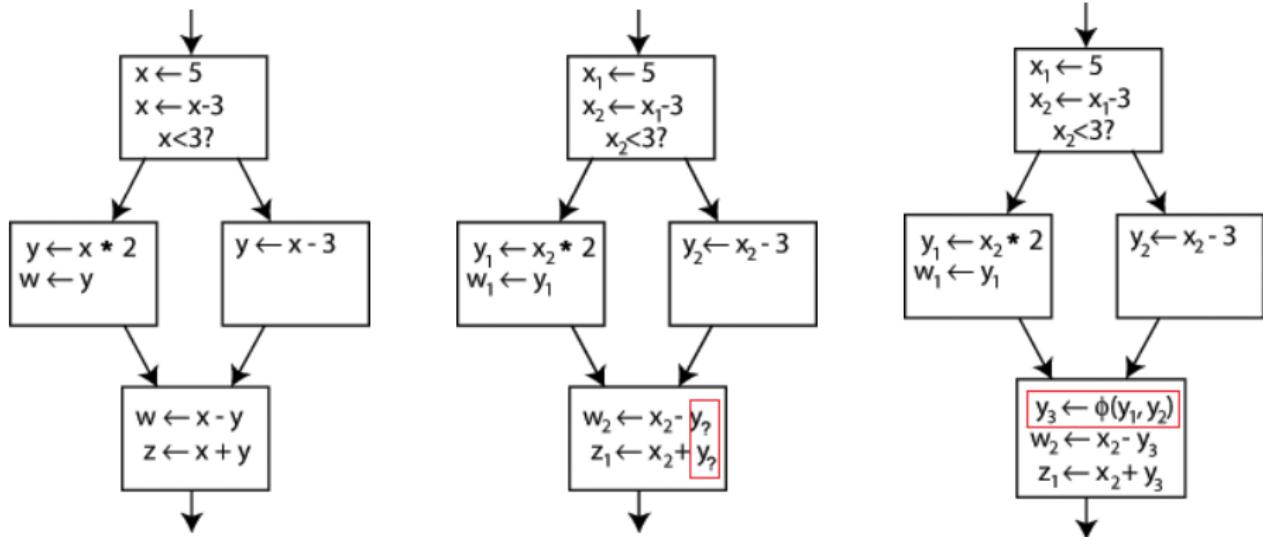
Terminator Instructions

As mentioned [previously](#), every basic block in a program ends with a “Terminator” instruction, which indicates which block should be executed after the current block is finished. These [terminator instructions](#) typically yield a ‘void’ value: they produce control flow, not values (the one exception being the [‘invoke’](#) instruction).

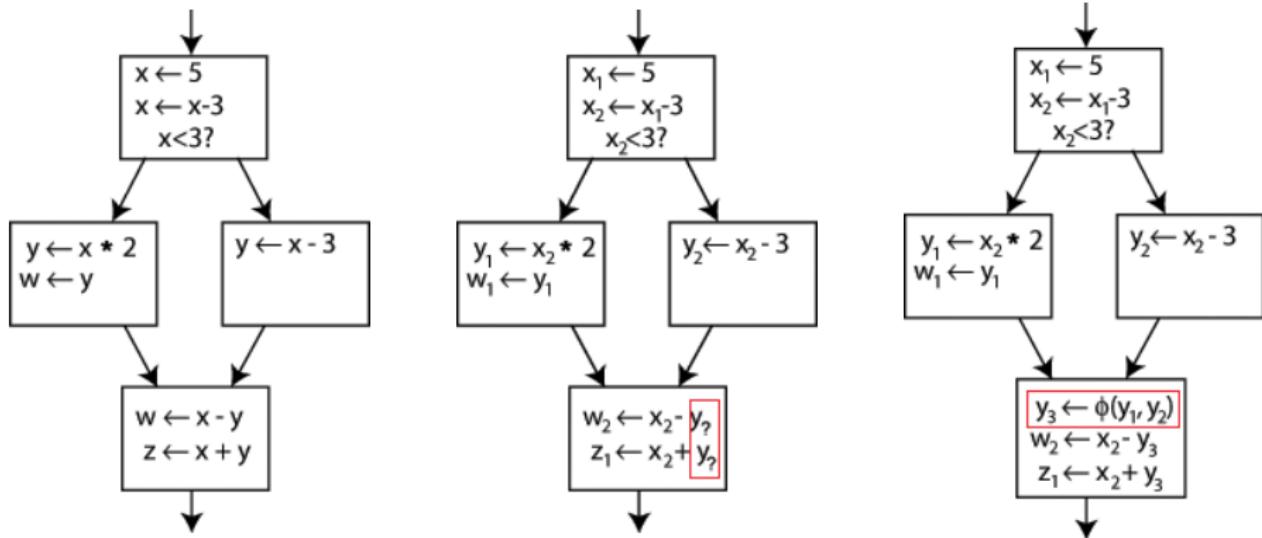


CFG for 'factorial' function

ϕ 根据控制流决定选择 y_1 还是 y_2

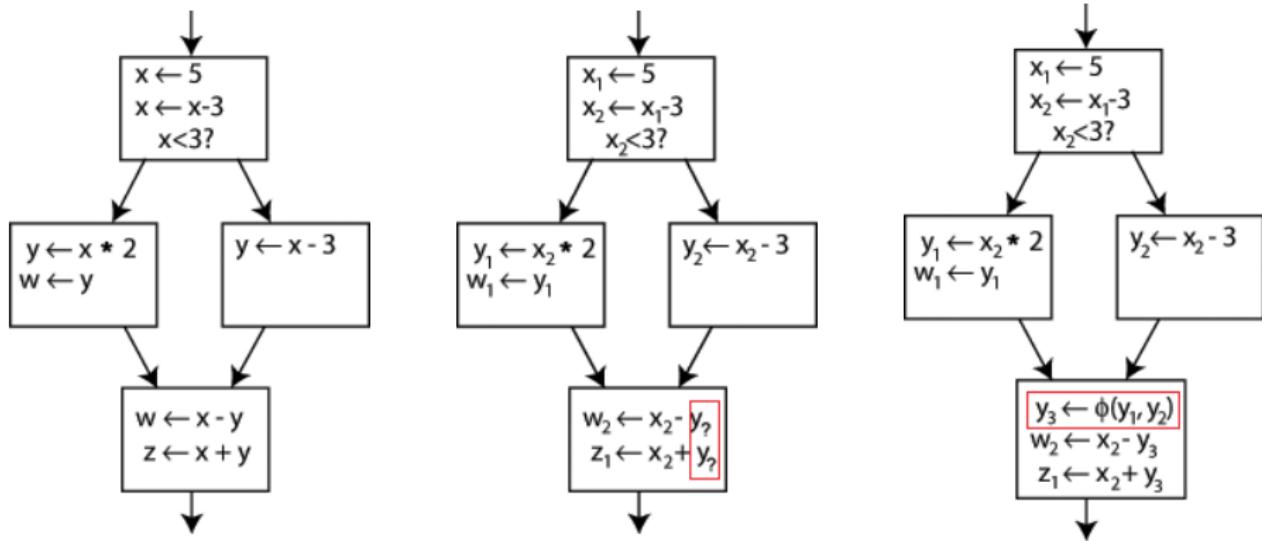


ϕ 根据控制流决定选择 y_1 还是 y_2



How to implement ϕ instruction?

ϕ 根据控制流决定选择 y_1 还是 y_2



How to implement ϕ instruction?

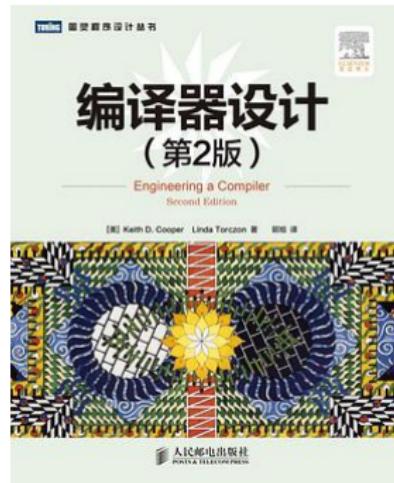
基本思想：将 ϕ 指令转换成若干赋值指令，上推至前驱基本块中

SSA 形式的构建 (Construction)、消去 (Destruction)、重建 (Reconstruction)



Section 4.3

SSA 形式的构建 (Construction)、消去 (Destruction)、重建 (Reconstruction)



Section 4.3

SSA 形式的构建 (Construction)、消去 (Destruction)、重建 (Reconstruction)



Efficiently Computing Static Single Assignment Form and the Control Dependence Graph

RON CYTRON, JEANNE FERRANTE, BARRY K. ROSEN, and
MARK N. WEGMAN
IBM Research Division
and
F. KENNETH ZADECK
Brown University

TOPLAS1991 @
compilers-papers-we-love

Section 4.3

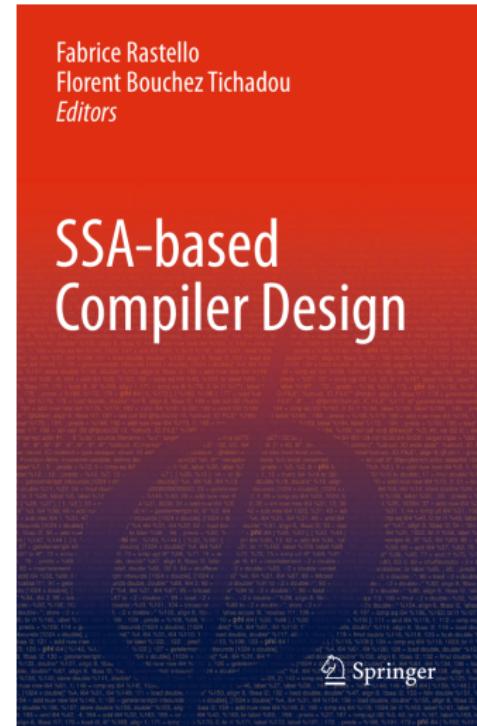
魏恒峰 (hfwei@nju.edu.cn)

中间代码生成



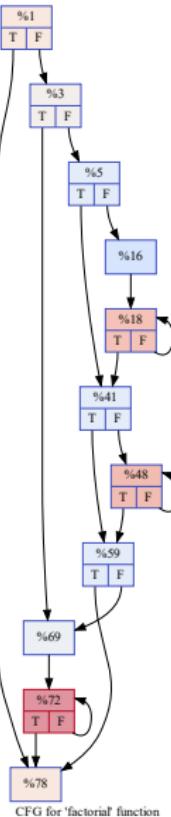
Section 9.3

“The SSA Book”



Springer

<https://github.com/courses-at-nju-by-hfwei/compilers-resources/tree/master/books/Classic%20Textbooks>



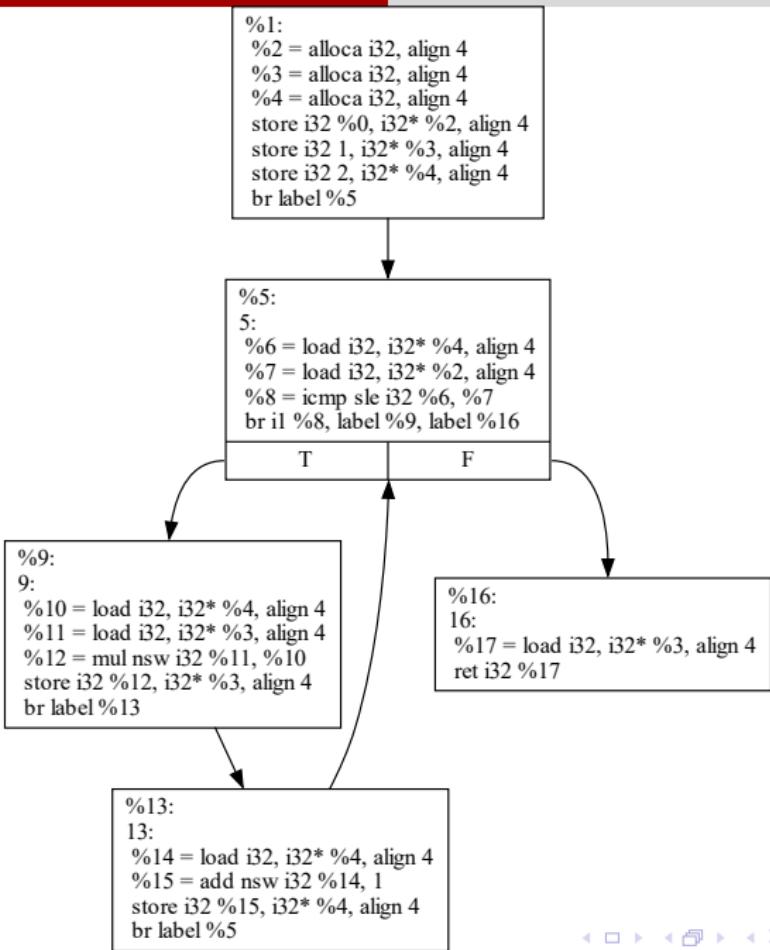
factorial1 (opt3)

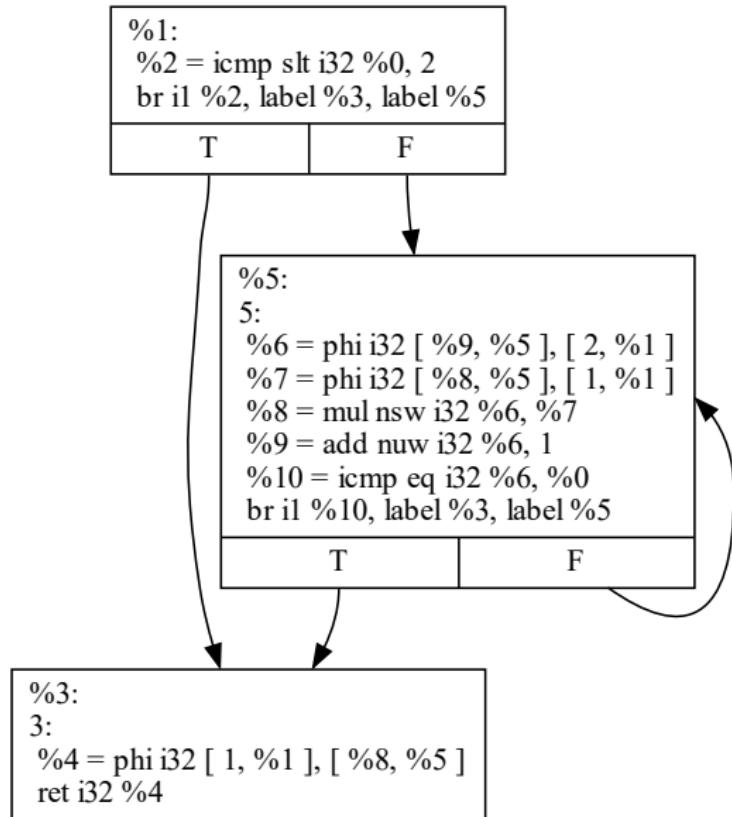
factorial2.c @ Compiler Explorer

```
int factorial(int val) {
    int temp = 1;

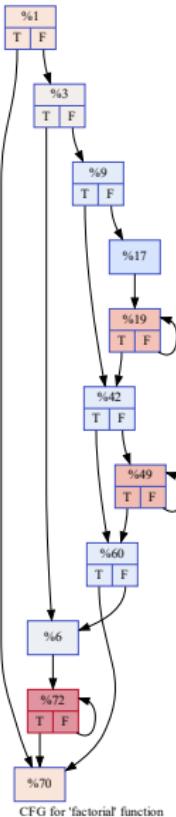
    for (int i = 2; i <= val; i++) {
        temp *= i;
    }

    return temp;
}
```





CFG for 'factorial' function



factorial2 (opt3)

LLVM IR Animation



LLVM IR Tutorial @ Bilibili

LLVM Language Reference Manual

https://llvm.org/docs/LangRef.html



[LLVM Home](#) | [Documentation](#) » [Reference](#) »

LLVM Language Reference Manual

Program Visualization using LLVM @ Bilibili



Program Visualization



LLVM Language Reference Manual

https://llvm.org/docs/LangRef.html



[LLVM Home](#) | [Documentation](#) » [Reference](#) »

LLVM Language Reference Manual

如何用编程的方式生成 LLVM IR?

Bytedeco/javacpp @ github

JavaCPP Presets Platform For LLVM



LLVM JAVA API使用手册
准备工作

课程实验中具体如何生成 ϕ 指令？



主打一个“课上不讲，课后自学”吗？



LLVM Programmer's Manual

A screenshot of a web browser displaying the LLVM Programmer's Manual. The URL in the address bar is <https://llvm.org/docs/ProgrammersManual.html>. The page features the LLVM logo (a stylized 'L' composed of two interlocking shapes) and the text "LLVM COMPILER INFRASTRUCTURE". Below the logo, a navigation bar includes links to "LLVM Home", "Documentation", "Getting Started/Tutorials", and the current page, "LLVM Programmer's Manual". The main title "LLVM Programmer's Manual" is prominently displayed in a large, bold, dark blue font. The browser interface shows standard navigation buttons at the bottom.

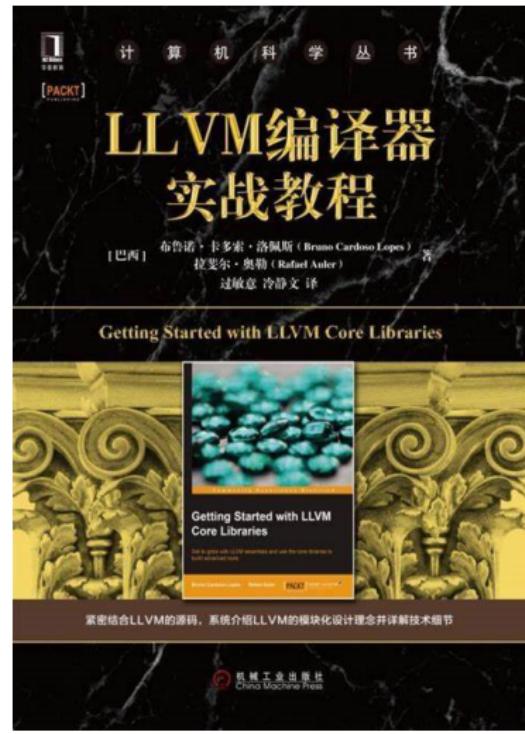
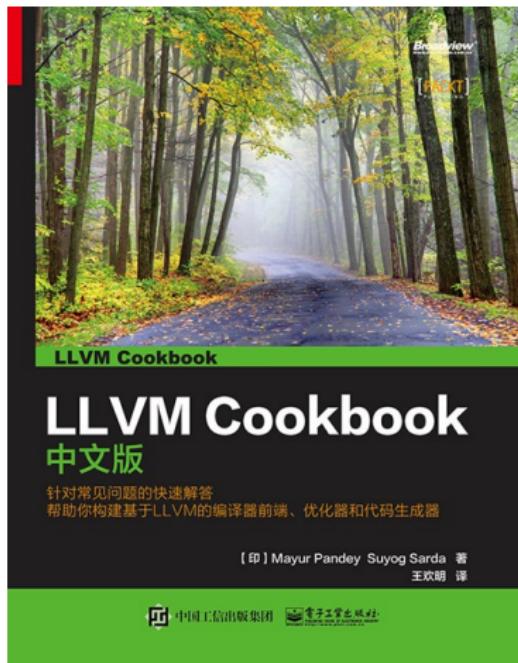
Kaleidoscope: Implementing a Language with LLVM

The screenshot shows the LLVM Documentation website at <https://llvm.org/docs/tutorial/>. The main navigation bar includes 'LLVM Home' and 'Documentation'. Below it, the 'Getting Started/Tutorials' section is selected. A large 'Table of Contents' heading is visible, followed by a link to 'Kaleidoscope: Implementing a Language with LLVM'.

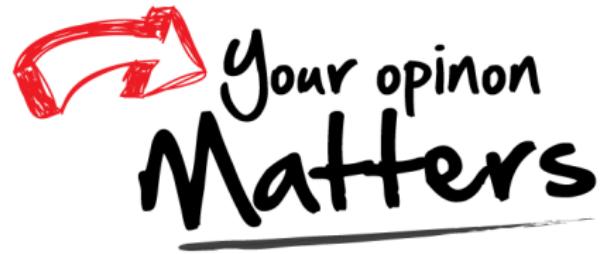
The screenshot shows a GitHub repository page for 'courses-at-nju-by-hfwei / kaleidoscope-in-java'. The repository is public and contains code for 'chapter3 / src / main / java / com / compiler / kaleidoscope / AST /'. A pull request titled 'finish chapter 3' has been merged by 'dracoooooo'. The commit message indicates the completion of chapter 3. The pull request details show eight files: BinaryExprAST.java, CallExprAST.java, ExprAST.java, FunctionAST.java, NumberExprAST.java, PrototypeAST.java, and VariableExprAST.java, all marked as 'finish chapter 3'.

kaleidoscope-in-java@github

强烈**不推荐**, 严重挫伤初学者的学习热情



Thank You!



Office 926

hfwei@nju.edu.cn