HZ BOOKS
华章IT

华章原创精品

**编译原理领域的鸿篇巨著和里程碑作品，它不仅能指导你设计出自己的编译器，更能帮助你写出更高质量的代码**

60余万行源代码、1140余幅运行时结构图，详尽阐述和展示应用程序的编译原理和GCC编译器的工作机制

# 编译系统透视

## 图解编译原理

SEE COMPILING SYSTEM RUN

新设计团队 著

机械工业出版社
China Machine Press

华章原创精品

# 编译系统透视

## 图解编译原理

新设计团队 著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

编译系统透视：图解编译原理 / 新设计团队著 . —北京：机械工业出版社，2015.4
（华章原创精品）

ISBN 978-7-111-49858-2

I. 编… II. 新… III. 编译器 – 图解 IV. TP314-64

中国版本图书馆 CIP 数据核字（2015）第 067988 号

本书是编译原理领域的鸿篇巨著，中文版尚未出版，英文版权已经输出到了美国。本书的出版将在世界范围内产生重要影响。从以下多个角度讲，本书都具有重要的里程碑意义：

- 它第一次让编译原理不再像是一门高深晦涩的“数学课”，而是一个可以调试、可以接触、可以真切感受的理论体系。本书用1140 余幅信息量巨大的运行时结构图和视频动画取代了同类书中复杂枯燥的数学公式，更加立体和直观，生动地将编译后的执行程序在内存中的运行时结构图展现了出来。
- 它第一次将 GCC 源代码、编译原理、运行时结构、编译系统原理（包含汇编与链接）的内在关系、逻辑与原理梳理清楚了，并将它们结合成一个整体，真正能够让读者透彻掌握编译器如何运行、如何设计，以及为什么要这么设计。
- 它是第一本系统解读著名商用编译器 GCC 核心源代码的著作。GCC 源代码一共有 600 万行，为了便于讲解和阅读，本书进行了取舍和裁剪，讲解了与编译本质相关的、最核心的 60 万行代码。

全书一共 8 章，具体内容和逻辑如下：

第 1 章以一个 C 程序（先简单，后复杂）的运行时结构为依托，对程序编译的整体过程做了宏观讲述，让读者对编译有一个整体认识，这样更容易理解后面的内容。

第 2 ~ 6 章通过实际的程序案例、结合 GCC 的源代码，根据程序编译的顺序和流程，依次讲解了词法分析、语法分析、中间结构和目标代码的生成，遵循了由易到难的原则，先是通过简单程序讲解清楚原理，然后通过复杂程序强化理解。

第 7 章讲解了与编译器紧密关联的汇编器和链接器，让读者对可执行程序的最终生成有一个完整的了解。

第 8 章讲解了预处理，就编译器的执行顺序而言，预处理器的执行比较靠前，之所以放在最后讲，是因为它比较独立，在读者已经了解整个编译过程之后再讲解，读者会更容易理解。

# 编译系统透视：图解编译原理

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）

责任编辑：姜 影 责任校对：董纪丽

印 刷：北京诚信伟业印刷有限公司 版 次：2016 年 3 月第 1 版第 1 次印刷

开 本：214mm × 275mm 1/16 印 张：65.75

书 号：ISBN 978-7-111-49858-2 定 价：169.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88379426 88361066 投稿热线：（010）88379604

购书热线：（010）68326294 88379649 68995259 读者信箱：hzit@hzbook.com

版权所有 • 侵权必究

封底无防伪标均为盗版

本书法律顾问：北京大成律师事务所 韩光 / 邹晓东

新设计团队由中国科学院大学的教师杨力祥发起，成立于世纪之交，团队成员全部都是杨力祥老师的得意弟子，现在他们是很多企业核心和支柱。新设计团队不断发展、优胜劣汰、适者生存、自然形成。团队在计算机领域中始终只对最基础的、有体系的事情感兴趣，喜欢从根节点解决问题，目前已经在编译器和操作系统等领域取得了突破性的成果，具体如下：

### 1. 图示化的编译器

成功研发出基于图形、图像（而非基于字符、语句）的图示化集成开发环境，直接由图形编译为可执行程序，中间不再转成一般的计算机语言。已经能够成功编译扫雷等界面应用程序，也可以成功编译 Linux0.11 这样的简单操作系统，编译结果可以正确 boot 运行。

### 2. 安全操作系统

研发出全新的安全操作系统。使用现有 CPU、内核依据新的原理设计而成，不需要安装任何防火墙和杀毒软件，就可以抵御一切已知、未知的网络入侵。此操作系统支持 FTP 的基础功能，兼容 Linux。此操作系统曾经于 2014 年 4 月 1 日至 2014 年 9 月 30 日在互联网上悬赏 1 万美金进行入侵攻击测试，至今未有人攻破。

### 3. 基于安全 CPU 的安全操作系统

根据新的操作系统原理，团队还设计了包含全新安全指令的 CPU，以及由安全 CPU 支持的安全操作系统。目前安全 CPU 的指令设计已经完成，开发了安全 CPU 仿真平台及基于其上的全新安全操作系统。操作系统可以全面支持 Linux（技术上也可以做到全面支持 Windows 或其他操作系统）。一旦 CPU 硬件设计制造完成，就可以直接运行安全操作系统，实现真正的安全（EAL7）。

新设计团队还将其对编译器和操作系统的研究理论成果集结成出版物，除本书外，还出版了《Linux 内核设计的艺术》（2011 年，机械工业出版社）一书，版权输出到美国、韩国，实现了国内计算机著作向美国输出的零突破。英文版被美国的 MIT、Stanford、Cornell、UMD 等 100 多所大学图书馆及 Library of Congress（美国国会图书馆）收藏。

# 前　言 Preface

掌握程序在内存中的运行时结构对提高程序设计水平的重要性再怎么强调都不过分，将程序员编写的源代码转化为可执行程序是由编译器完成的，编译器对运行时结构的形成起着非常重要的作用。如果你想提高自己的编程水平，了解编译器怎么将你编写的源代码转换为可执行程序的，那么本书就是为你而写的！如果你对编译原理很感兴趣，也很愿意阅读编译器的源代码，却苦于代码量庞大，不知从何下手，那么你必将从本书中得到巨大的收获。

对程序员来说，提高编程水平最关键的因素之一就是了解程序的运行时结构，只有了解了自己编写的源代码运行的时候在内存中是什么样的（运行时结构），才能真正写出高质量的代码。编译器是将源代码转化为最终运行时结构的工具，如何实现运行时结构正是本书最重要的一条主线。编译器是一个非常经典的程序，其中包含的很多技术已广泛应用于其他软件（如文字处理软件、数据库、Web 开发程序等）。读懂编译器的源代码，对计算机软件的很多方面来说都会有借鉴作用。

一般介绍编译原理的书籍通常都是空泛地讲一些抽象的概念，甚至夹杂不少晦涩的数学公式，脱离了具体的编译器，基本上没有编译器的源代码，初学者很难理解。

而本书则是以一个真实、具体、商用 GCC 编译器的源代码为蓝本，以几个案例程序的实际编译为线索，详细讲解编译案例程序的源代码的具体过程。

本书先对读者最难理解的复杂过程、关系和数据结构以动画视频的方式进行直观、形象的讲解。看过这些视频，读者就会对编译原理有一个概略、直观、整体的理解，从而很容易掌握更深的内容。纸质内容再将编译原理与 GCC 编译器的源代码有机联系起来，用了大量直观的图示、源代码、文字做详细讲解。

本书没有用一个数学公式，力争用最简单易懂的语言把深奥的理论讲明白。读者在看完本书后会真正了解一个编译器是如何运行的，以及为什么要这么设计，更重要的是知道编译完的程序执行时在内存中的运行时结构是什么样的。

我们还为读者提供了一个缩减版的 GCC 源代码。原版的 GCC 源代码大约有 600 万行，是一个适用于多种计算机语言的编译器，体量过于庞大，几乎无法在短时间内阅读、理解，甚至很难记忆。我们只保留了 C 语言的相关部分，并去掉了错误分析、处理和优化的相关部分，大约只有 130 万行，其中约 50 万行是为了与具体指令集相关，由机器生成的代码，仅涉及后端；在剩下的 80 万行代码中，与编译本质相关的核心代码大约有 60 万行。此外，我们还提供了与之相对应的汇编器和链接器的源代码，这些代码虽然不是编译器的一部分，但却是生成完整的可执行程序必不可少的。我们还提供了一整套的开发调试环境，既有适用于 Linux 的，也有适用于 Windows 的。读者可以在一个比较小的范围内随着本书的讲解跟踪调试，这样效率更高。读者在阅读的时候始终都能与真实的编译过程、真实的编译器源代码紧密相连。本书的编译原理不再像一门“数学课”，而是一个可以调试、可以接触、可以真切感受的理论体系。

试读结束：需要全本请在线购买： www.ertongbook.com

读者只要了解 C 语言的语法规则，会使用 C 语言编写一些简单的程序，就能看懂本书。

## 本书内容安排

第 1 章的前半部分先讲解程序的运行时结构。如我们一再强调，运行时是程序执行的关键，编译器正是将源代码转化为可执行程序并形成运行时结构的工具。对于只是想提高开发能力的程序员，这部分几乎起到了 90% 的作用。

第 1 章的后半部分对整体的编译过程做了一个综述。当读者对整体有概念的时候，再去看每个章节的具体内容，会更容易理解。

第 2 章用一个简单案例讲解词法分析。词法分析是把源文件中的内容读出并识别出符号的过程。

第 3 章继续用词法分析时的简单案例讲解语法分析。语法分析是在词法分析的结果中识别出语句的过程。

第 4 章仍用前面的简单案例生成中间结构及目标代码。

第 5 章和第 6 章用几个更为复杂的案例来分析语法和生成中间结构及目标代码的过程。

至此，严格意义上的编译过程已经讲解完毕。为了让读者对最终生成的可执行程序有一个完整的了解，我们专门安排了汇编器、链接器的内容，这就是第 7 章。

第 7 章详细讲解如何将目标代码转变为可执行程序，包括文件格式、汇编器和链接器的内容。

第 8 章讲预处理。从编译器的执行顺序来看，预处理器的执行是比较靠前的，之所以把预处理放在最后讲，是因为预处理比较独立，在介绍完整个编译过程之后再讲解，读者更容易理解。

其中第 1 ~ 3 章都配了相应的视频，建议在看纸质内容之前先看视频。

## 致谢

首先，依然要感谢机械工业出版社华章公司的副总经理温莉芳女士以及其他领导，他们数年来一如既往的支持，是本书能够顺利出版的前提。

其次，特别感谢机械工业出版社华章公司的副总编辑杨福川，他对事业的追求、对工作认真负责的态度以及与作者团队的密切配合，使得本书能够以常规条件下难以置信的速度走上出版流程，与读者见面。

还要感谢机械工业出版社华章公司的版权输出团队和 CRC Press 的贺瑞君先生，他们的不懈努力、扎实工作和高效沟通，使得本书能够成功版权输出。

最后，感谢我们的家人和朋友，是他们的坚定支持才使得团队能够拒绝方方面面、形形色色的诱惑，放弃普遍追求的短期利益，踏踏实实地做一点实在、深入的工作。这是本书的基础。

# 目　录 *Contents*


作者简介

前　言

**第1章　运行时结构及编译过程概述** ······ 1

1.1　一个简单C程序的运行时结构 ······ 1

1.2　更为复杂C程序的运行时结构 ······ 16

1.3　编译过程概述 ······ 25

1.3.1　词法分析 ······ 25

1.3.2　语法分析 ······ 26

1.3.3　从语法树到中间代码再到目标代码 ······ 26

**第2章　词法分析** ······ 28

2.1　词法分析概要说明 ······ 28

2.2　词法分析过程 ······ 31

2.3　状态转换图 ······ 36

2.3.1　状态转换图总体介绍 ······ 36

2.3.2　依托状态转换图展现词法分析过程 ······ 42

2.4　GCC实现词法分析的源代码 ······ 55

2.4.1　词法分析源代码总览 ······ 55

2.4.2　结合GCC源代码讲解词法分析过程 ······ 55

2.4.3　标识符、数字、字符和字符串的详细分析过程 ······ 65

**第3章　语法分析** ······ 74

3.1　语法分析综述 ······ 74

3.2　语法分析思路 ······ 74

3.3　产生式 ······ 78

3.3.1　什么是产生式 ······ 78

3.3.2　产生式的具体示例 ······ 80

3.4　匹配产生式，消除左递归 ······ 89

3.4.1　用标准产生式做匹配，出现左递归 ······ 89

3.4.2　消除左递归 ······ 93

3.4.3　产生式的工作效率 ······ 97

3.5　提取左公因子，消除回溯 ······ 100

3.5.1　对“直接声明符”的产生式提取左公因子 ······ 100

3.5.2　用提取过左公因子的产生式再去匹配 ······ 102

3.5.3　对其他产生式都提取左公因子 ······ 103

3.5.4　函数声明和定义两部分产生式的合并 ······ 105

3.6　语法分析结果：语法树 ······ 107

3.7　GCC关于语法分析的源代码解析 ······ 112

3.7.1　GCC语法分析函数调用图 ······ 112

3.7.2　全部语句的语法分析 ······ 115

**第4章　语法树到目标代码** ······ 217

4.1　总述语法树到中间代码的转化过程 ······ 217

4.2　目标代码到运行时结构的映射 ······ 224

4.3　语法树转高端gimple ······ 232

4.3.1　语法树到高端gimple的总体步骤及运行时 ······ 236

4.3.2　高端gimple的实际数据结构 ······ 241

4.3.3　语法树转高端gimple的GCC源代码解析 ······ 246

4.4　高端gimple到低端gimple ······ 286

4.4.1 高端 gimple 转低端 gimple 概述 ………… 286
4.4.2 高端 gimple 转化低端 gimple 的 GCC 代码解析 ………… 293
4.5 低端 gimple 到 cfg ………… 297
4.5.1 低端 gimple 到 cfg 的转化概述 ………… 297
4.5.2 低端 gimple 转 cfg 的实际过程 ………… 300
4.6 cfg 转 ssa ………… 301
4.7 生成 RTL ………… 305
4.7.1 为何要有 RTL ………… 305
4.7.2 转化 RTL 阶段的主要步骤 ………… 306
4.7.3 确定初始 RTL 中的运行时信息 ………… 320
4.8 RTL 生成目标代码（汇编）………… 332
4.8.1 汇编文件介绍 ………… 332
4.8.2 创建汇编文件 ………… 334
4.8.3 输出汇编文件总入口 ………… 334
4.8.4 全局变量写入汇编文件 ………… 335
4.8.5 函数写入汇编文件 ………… 340

**第 5 章 语句拓展案例的编译过程 ………… 353**

5.1 总述各个语句拓展案例的编译过程 ………… 353
5.2 if 语句的语法分析 ………… 376
5.2.1 多个变量的声明语句语法分析 ………… 376
5.2.2 if 语句的语法分析过程 ………… 381
5.2.3 if...else if 语句的语法分析过程 ………… 387
5.3 带标号语句的语法分析 ………… 395
5.4 switch...case、goto、break 语句的语法分析过程 ………… 399
5.4.1 switch...case 语句 ………… 399
5.4.2 goto 语句 ………… 407
5.4.3 分析 break 语句 ………… 409
5.5 do...while、while、for 语句的语法分析过程 ………… 420
5.5.1 do...while 语句的语法分析 ………… 424
5.5.2 while 语句的语法分析过程 ………… 433
5.5.3 for 语句的语法分析过程 ………… 444
5.6 各种语句嵌套组合的语法分析过程 ………… 472
5.6.1 两条变量声明语句分析的结果 ………… 477
5.6.2 分析 while 循环语句 ………… 477
5.6.3 进入 if 进行分析 ………… 480
5.6.4 进入 else 进行分析 ………… 485
5.7 所有案例语法树转中间结构的过程 ………… 516
5.7.1 案例 1 的语法树转高端 gimple 的总体介绍 ………… 516
5.7.2 案例 1 的语法树转高端 gimple 的代码分析 ………… 528
5.7.3 案例 1 的高端 gimple 转低端 gimple ………… 552
5.7.4 案例 1 的低端 gimple 到 cfg ………… 552
5.7.5 转化 RTL 阶段的主要步骤 ………… 562
5.7.6 案例 2 的语法树转高端 gimple ………… 587
5.7.7 案例 3 的语法树转高端 gimple ………… 596

**第 6 章 数据拓展案例的编译过程 ………… 612**

6.1 数据拓展案例的编译过程总述 ………… 612
6.1.1 基础类型数据总述 ………… 612
6.1.2 用户自定义类型数据总述 ………… 617
6.1.3 指针类型数据总述 ………… 626
6.1.4 作用域和生存期总述 ………… 640
6.1.5 表达式总述 ………… 645
6.2 基础类型数据的语法分析过程 ………… 652
6.2.1 非浮点型数据的语法分析 ………… 653
6.2.2 浮点型数据的语法分析 ………… 662
6.3 复合类型数据的语法分析过程 ………… 670
6.3.1 数组的语法分析 ………… 670
6.3.2 枚举类型数据的语法分析 ………… 675
6.3.3 struct 类型数据的语法分析 ………… 678
6.3.4 union 类型数据的语法分析 ………… 683
6.3.5 自定义数据声明和使用的语法分析 ………… 684
6.4 指针类型数据的语法分析过程 ………… 693
6.4.1 对 swap_point 函数中指针的语法分析 ………… 693
6.4.2 对指针使用的语法分析 ………… 696
6.5 关于作用域和生存期的语法分析过程 ………… 705
6.5.1 C 语言作用域和生存期概述 ………… 705
6.5.2 全局变量 data 语法分析中作用域相关处理过程 ………… 706

6.5.3 fun 函数定义的语法分析中作用域相关处理 …… 709

6.5.4 main 函数定义中局部变量声明 data 作用域处理过程 …… 716

6.5.5 main 函数内部语句块中变量 nCount 作用域处理过程 …… 719

6.5.6 main 函数中引用变量 data 时选择相应声明节点的过程分析 …… 719

6.5.7 main 函数中引用变量 nCount 时选择相应声明节点的过程分析 …… 720

6.5.8 main 函数中退出内部语句块时更新变量作用域过程分析 …… 721

6.5.9 fun 函数中静态变量 temp 生存期信息的语法分析 …… 726

6.6 表达式的语法分析过程 …… 728

6.6.1 if 条件中的表达式语法分析 …… 728

6.6.2 if 条件下面“语句”部分的表达式语法分析 …… 740

6.7 所有案例语法树转中间结构（RTL）的过程 …… 754

6.7.1 基础类型数据语法树转高端 gimple 的过程 …… 754

6.7.2 用户自定义数据语法树转高端 gimple 的过程 …… 794

6.7.3 指针类型数据语法树转高端 gimple 的过程 …… 838

6.7.4 作用域和生存期案例语法树转高端 gimple 的过程 …… 878

6.7.5 复杂表达式案例的语法树转高端 gimple 的过程 …… 887

**第 7 章 汇编与链接** …… 934

7.1 汇编器 …… 934

7.1.1 详细介绍汇编指令到机器指令的转化 …… 934

7.1.2 .o 文件格式总体情况介绍 …… 953

7.1.3 代码段、数据段以及其他各个表项间的关系 …… 962

7.1.4 从汇编文件到目标文件的实现 …… 967

7.1.5 汇编器处理的源代码分析 …… 973

7.2 链接器 …… 985

7.2.1 .o 文件链接总体介绍 …… 985

7.2.2 多个 .o 文件链接时通过符号表建立关系 …… 989

7.2.3 链接时统一计算地址并回填 …… 997

7.2.4 链接器源代码介绍 …… 999

7.2.5 库函数的链接 …… 1002

7.2.6 动态链接 …… 1002

**第 8 章 预处理** …… 1012

8.1 文件包含 …… 1012

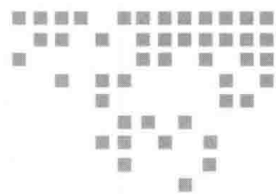
8.2 宏定义 …… 1017

8.3 条件编译 …… 1019

8.4 带参数的宏定义 …… 1022

**附录 RTX 定义** …… 1031

**作者的话** …… 1039



第 1 章 Chapter 1

# 运行时结构及编译过程概述

为了让读者能更直观和更容易地理解本章的内容，我们针对本章内容精心制作了视频内容，感兴趣的读者可以扫描二维码观看和学习。

## 1.1 一个简单 C 程序的运行时结构

解决编程过程中的实际问题，需要透彻了解程序在内存中的运行时结构，而透彻的程度自然成为衡量计算机语言学习水平的重要标准，也成为衡量软件项目开发水平的重要标准。

C 程序运行的核心是函数的执行和调用，它构成了整个 C 程序运行时结构的基础框架。这一运行过程主要是在程序指令的驱动以及数据压栈、清栈的支持下实现的。为了介绍这一过程，我们设计了一个简单 C 程序，如下所示：

```
int fun(int a,int b);
int m=10;
int main()
{
    inti=4;
    int j=5;
    m = fun(i,j);
    return 0;
}

int fun(int a,int b)
{
    int c=0;
    c=a+b;
    return c;
}
```

程序很简单，却凸现了函数调用和执行的最基本情况。我们把此情景展现在内存中，共有三个区域，分别是代码区、静态数据区和动态数据区。情景如图 1-1 所示。

代码区装载了这个程序所对应的机器指令，main 函数和 fun 函数的机器指令装载位置如图 1-2 所示。

全局变量 m 的数值装载在静态数据区中，情景如图 1-3 所示。

程序开始执行前，动态数据区中没有数据，情景如图 1-4 所示。

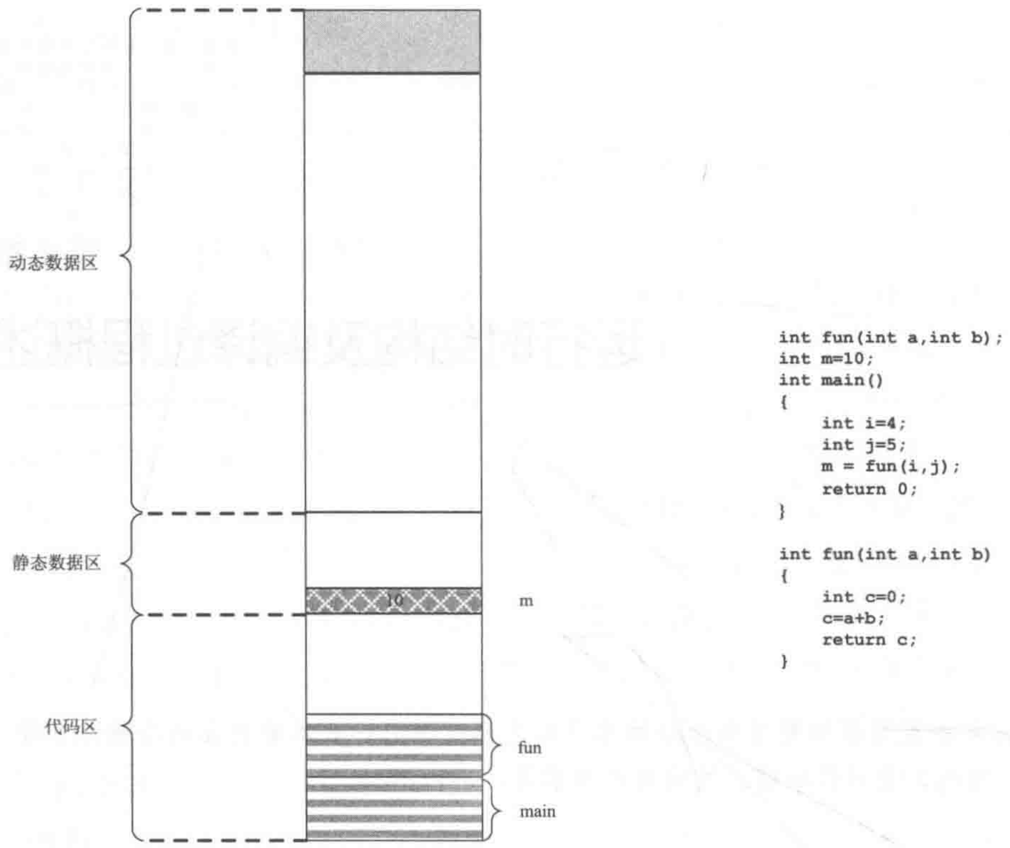



图 1-1　内存区域特性的总体介绍

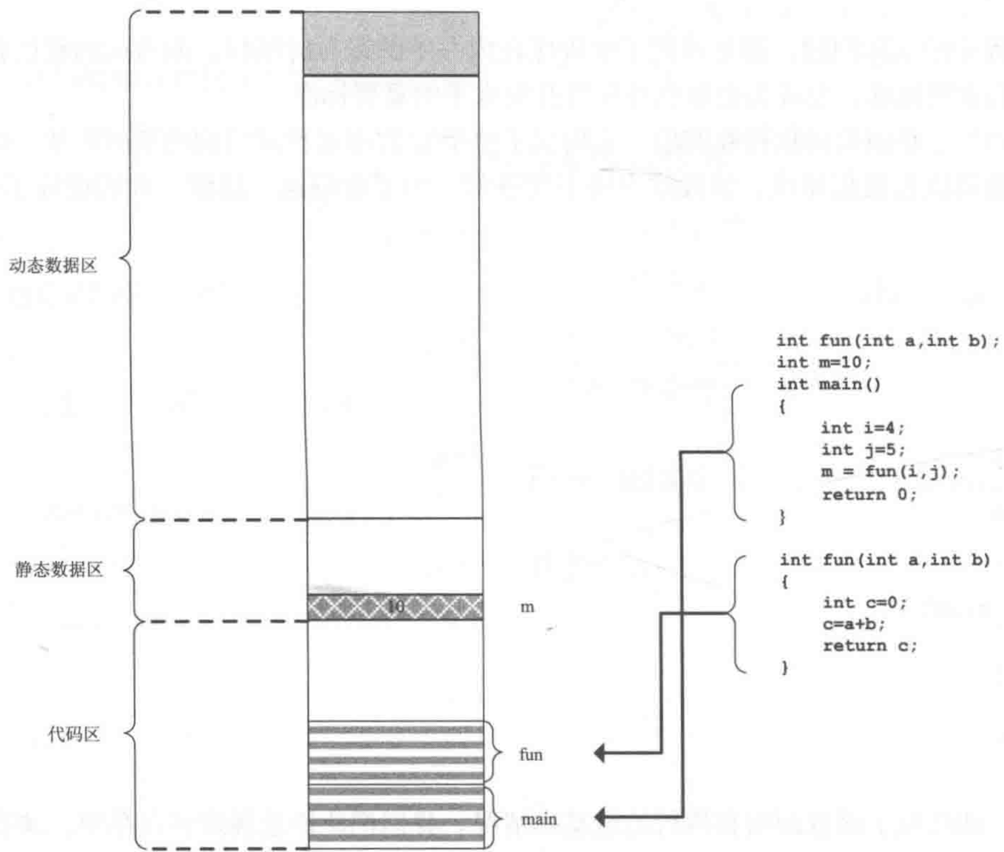



图 1-2　main 函数和 fun 函数在代码区的位置

这是因为，只有程序开始执行后，在指令的驱动下，这一区域才会产生数据，压栈和清栈的工作就是在这一区域完成的，情景如图 1-5 所示。

试读结束：需要全本请在线购买： www.ertongbook.com

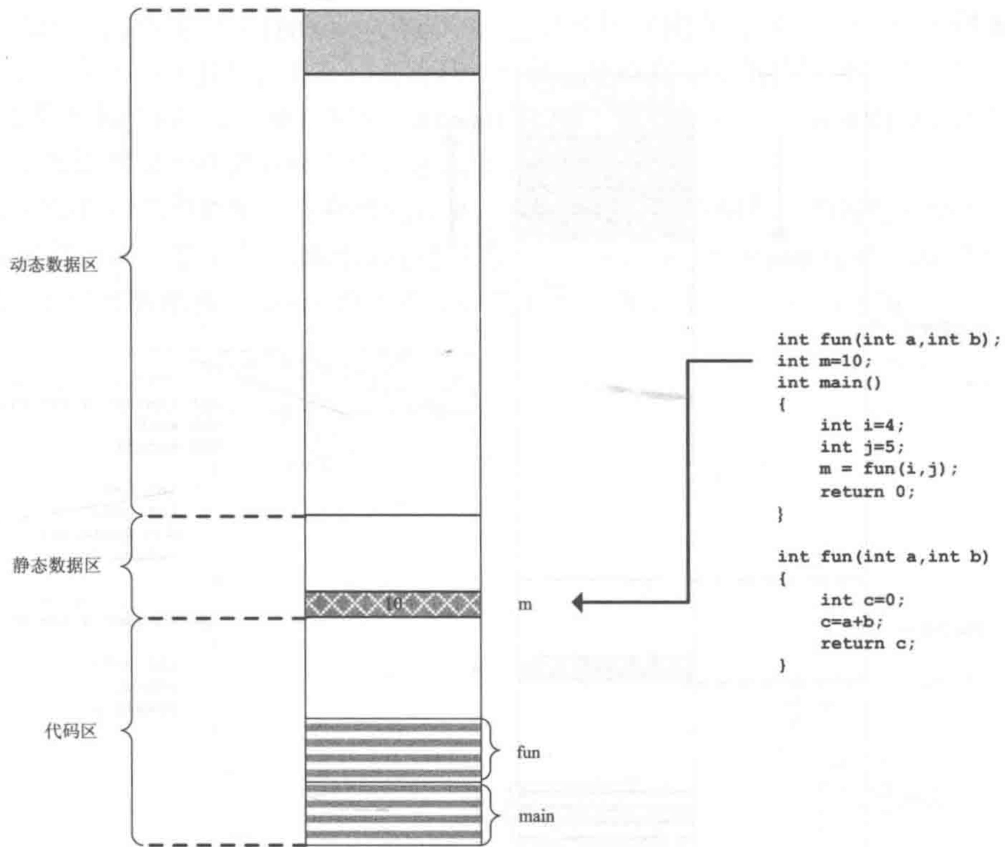



图 1-3　全局变量 m 在静态数据区的位置

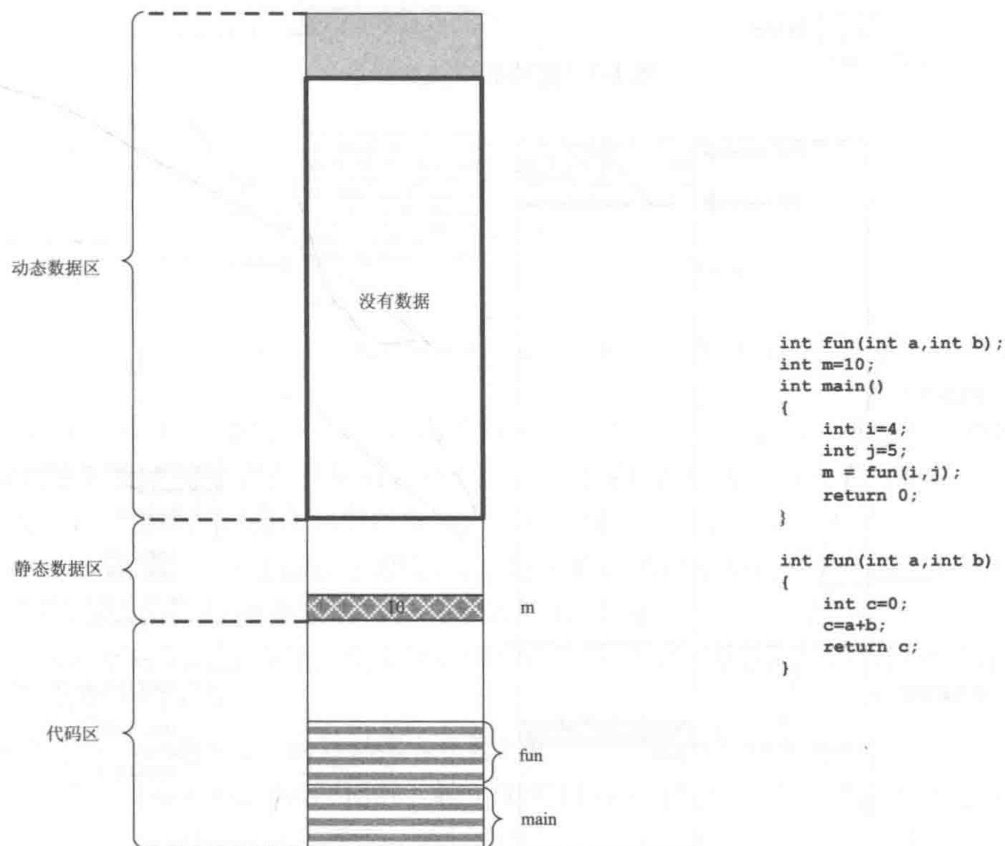



图 1-4　动态数据区没有数据

程序执行的本质就是代码区的指令不断执行，驱使动态数据区和静态数据区产生数据变化。这一过程需要计算机的管控。下面我们着重介绍对代码区和动态数据区的管控。CPU 中有三个寄存器，分别是 eip、ebp 和

esp，情景如图 1-6 所示。

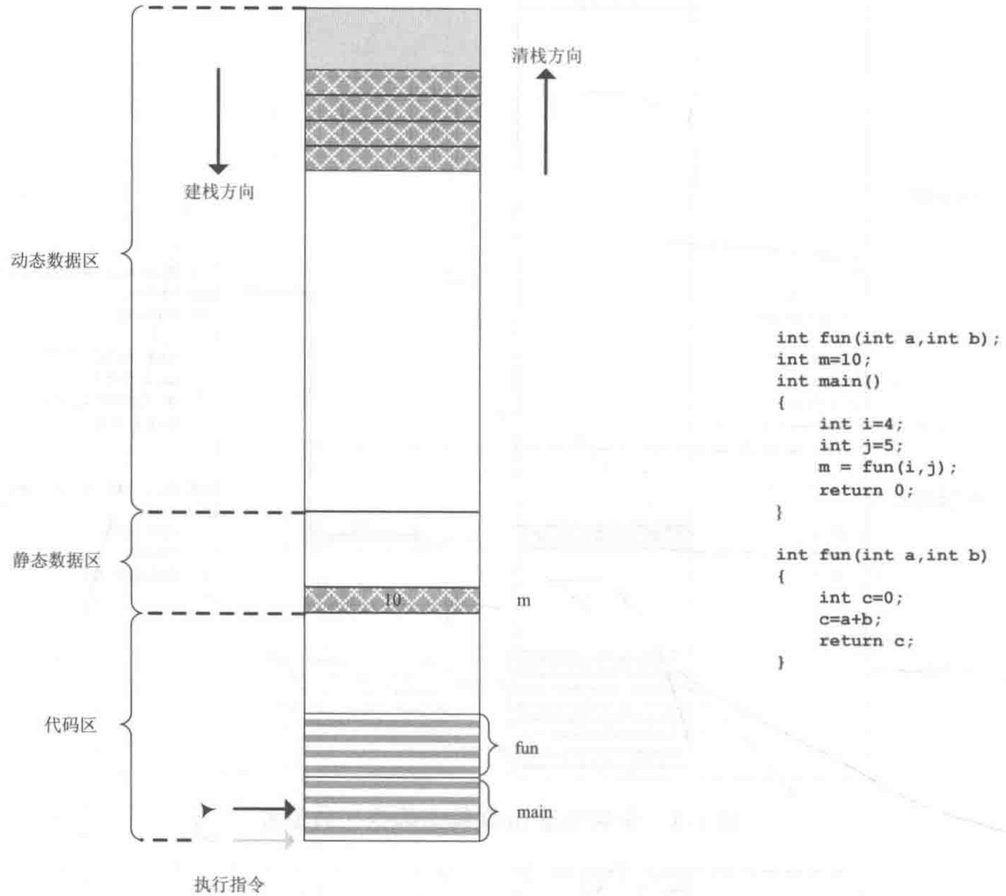



图 1-5　建栈和清栈的情景

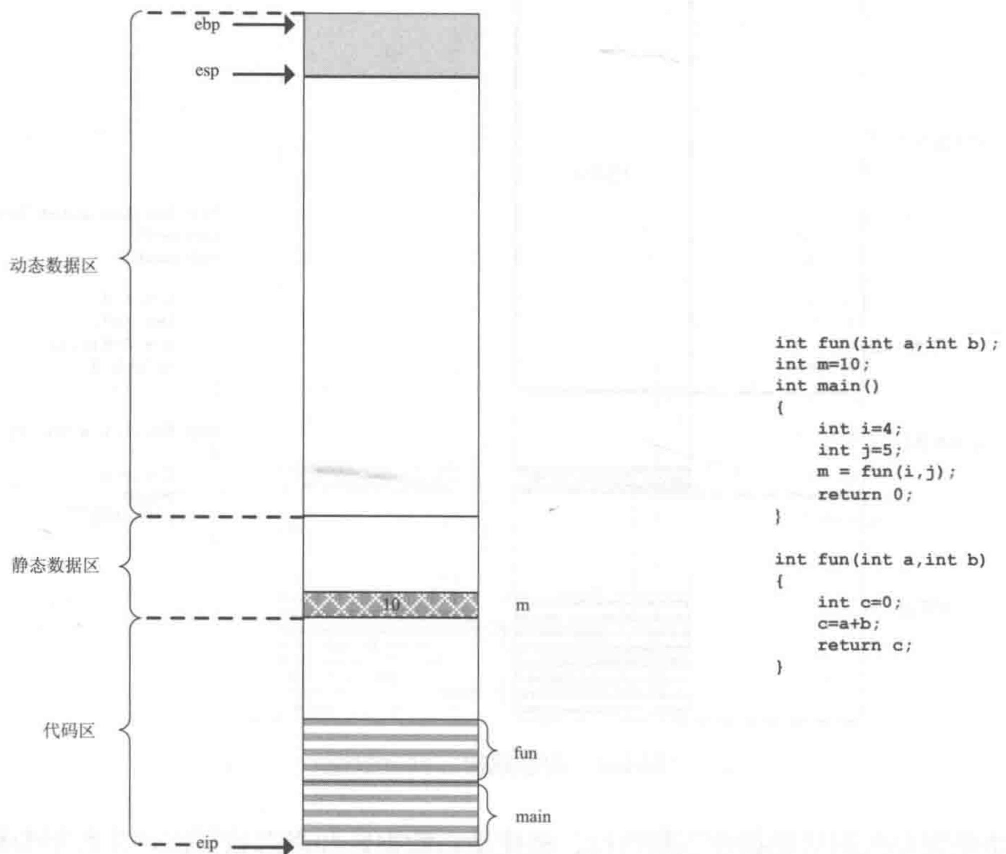



图 1-6　对代码区和动态数据区的管控

其中 eip 永远指向代码区将要执行的下一条指令，它的管控方式有两种，一种是“顺序执行”，即程序执行完一条指令后自动指向下一条执行；另一种是跳转，也就是执行完一条跳转指令后跳转到指定的位置。

ebp 和 esp 用来管控栈空间，ebp 指向栈底，esp 指向栈顶，在代码区中，函数调用、返回和执行伴随着不断压栈和清栈，栈中数据存储和释放的原则是后进先出。

内存的划分及程序执行的总体情况先介绍到这里。下面详细介绍案例程序的运行时结构。初始情景是这样的，eip 指向 main 函数的第一条指令，此时程序还没有运行，栈空间里还没有数据，ebp 和 esp 指向的位置是程序加载时内核设置的（详情请看《Linux 内核设计的艺术》一书），情景如图 1-7 所示。

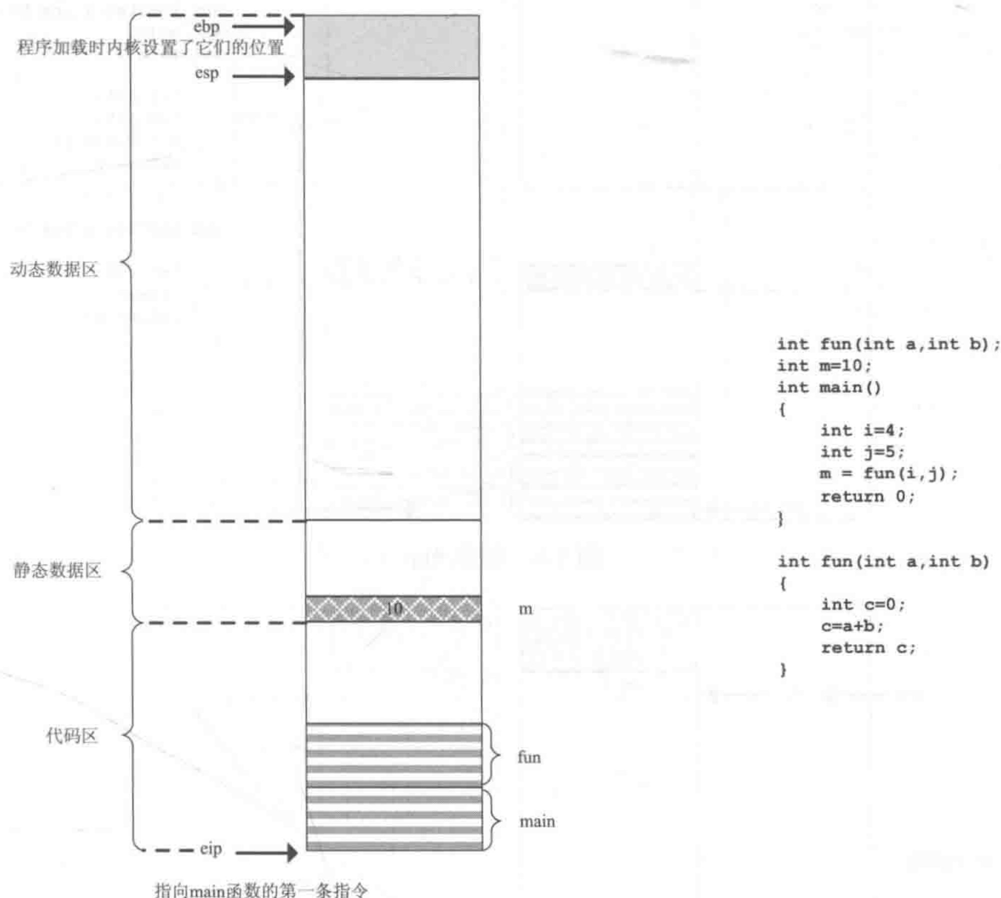



图 1-7　程序加载时 esp 和 ebp 的起始位置

程序开始执行 main 函数第一条指令，eip 自动指向下一条指令。第一条指令的执行，致使 ebp 的地址值被保存在栈中，保存的目的是本程序执行完毕后，ebp 还能返回现在的位置，复原现在的栈。随着 ebp 地址值的压栈，esp 自动向栈顶方向移动，它将永远指向栈顶，情景如图 1-8 所示。

程序继续执行，开始构建 main 函数自己的栈，ebp 原来指向的地址值已经被保存了，它被腾出来了，用来看管 main 函数的栈底，此时它和 esp 是重叠的，情景如图 1-9 所示。

程序继续执行，eip 指向下一条指令，此次执行的是局部变量 i 的初始化，初始值 4 被存储在栈中，esp 自动向栈顶方向移动，情景如图 1-10 所示。

继续执行下一条指令，局部变量 j 的初始值 5 也被压栈，情景如图 1-11 所示。

这两个局部数据都是供 main 函数自己用的，接下来调用 fun 函数时压栈的数据虽然也保存在 main 函数的栈中，但它们都是供 fun 函数用的。可以说 fun 函数的数据，一半在 fun 函数中，一半在主调函数中，下面来看函数调用时留在 main 函数中的那一半数据。

先执行传参的指令，此时参数入栈的顺序和代码中传参的书写顺序正好相反，参数 b 先入栈，数值是 main 函数中局部变量 j 的数值 5，情景如图 1-12 所示。

程序继续执行，参数 a 被压入栈中，数值是局部变量 i 的数值 4，情景如图 1-13 所示。

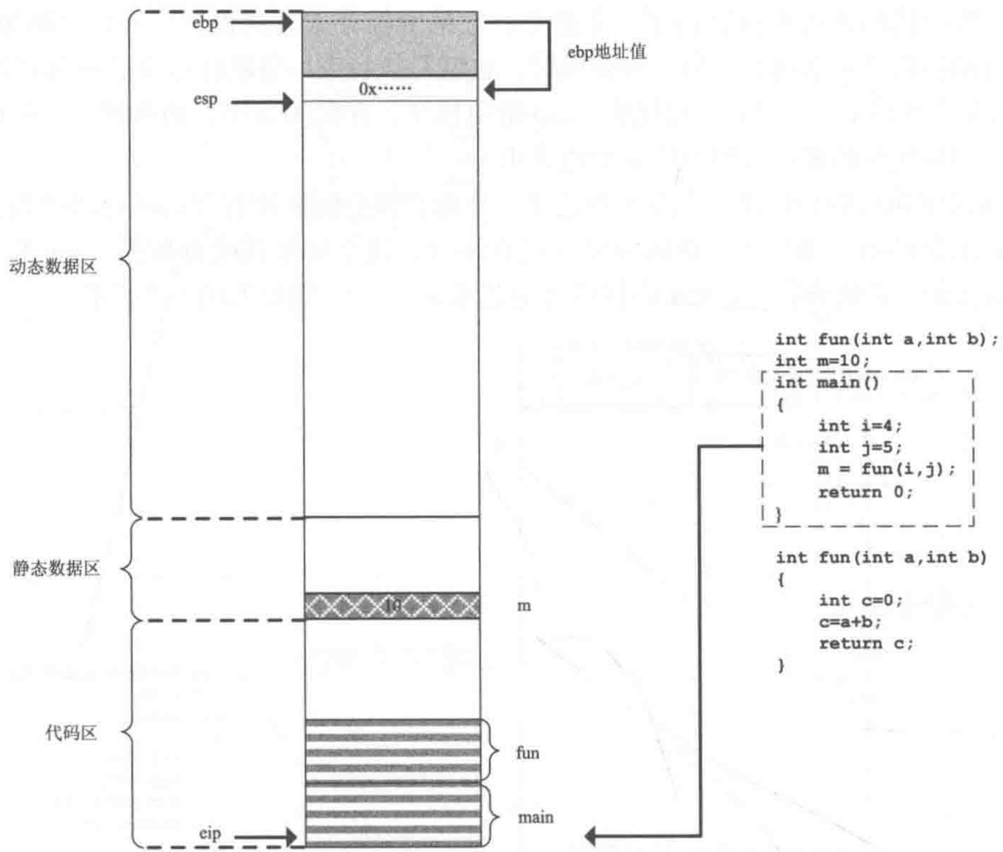



图 1-8　保存 ebp

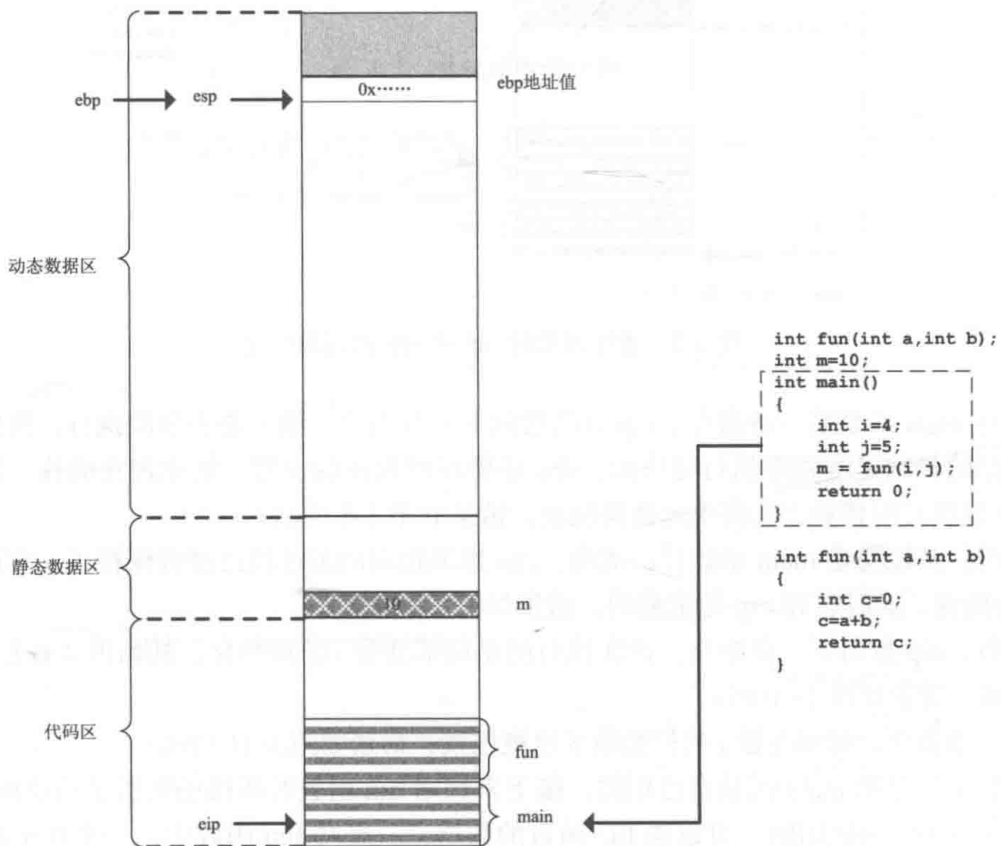



图 1-9　准备构建 main 函数的栈

程序继续执行，此次压入的是 fun 函数返回值，将来 fun 函数返回之后，这里的值会传递给 m，情景如图 1-14 所示。

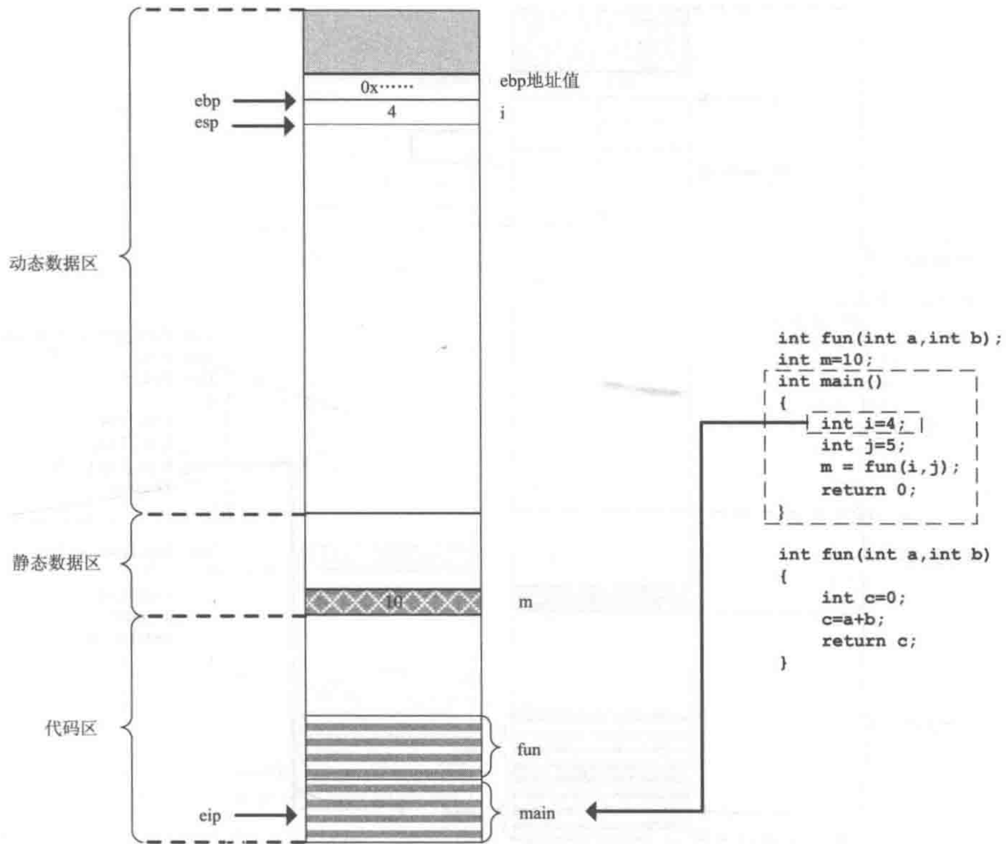



图 1-10 局部变量 i 压栈并初始化

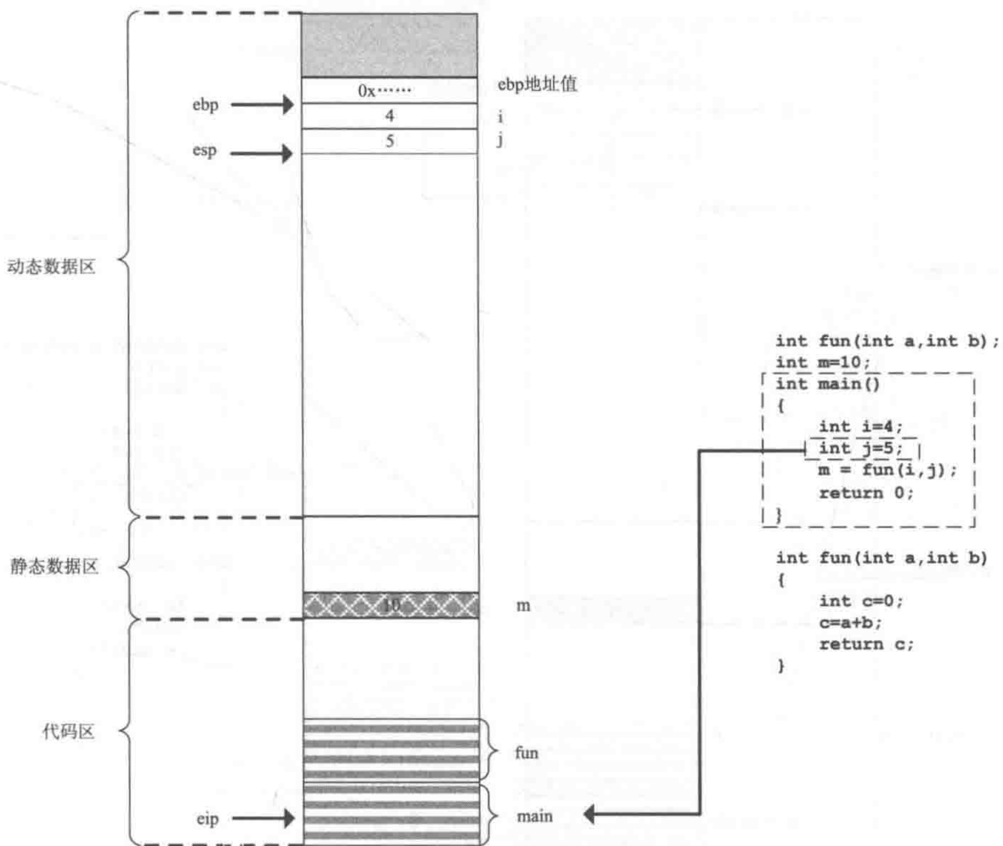



图 1-11 局部变量 j 压栈并初始化

还剩最后一步，跳转到 fun 函数去执行，这一步分为两部分动作，一部分是把 fun 函数执行后的返回地址压入栈中，以便 fun 函数执行完毕后能返回到 main 函数中继续执行，情景如图 1-15 所示。

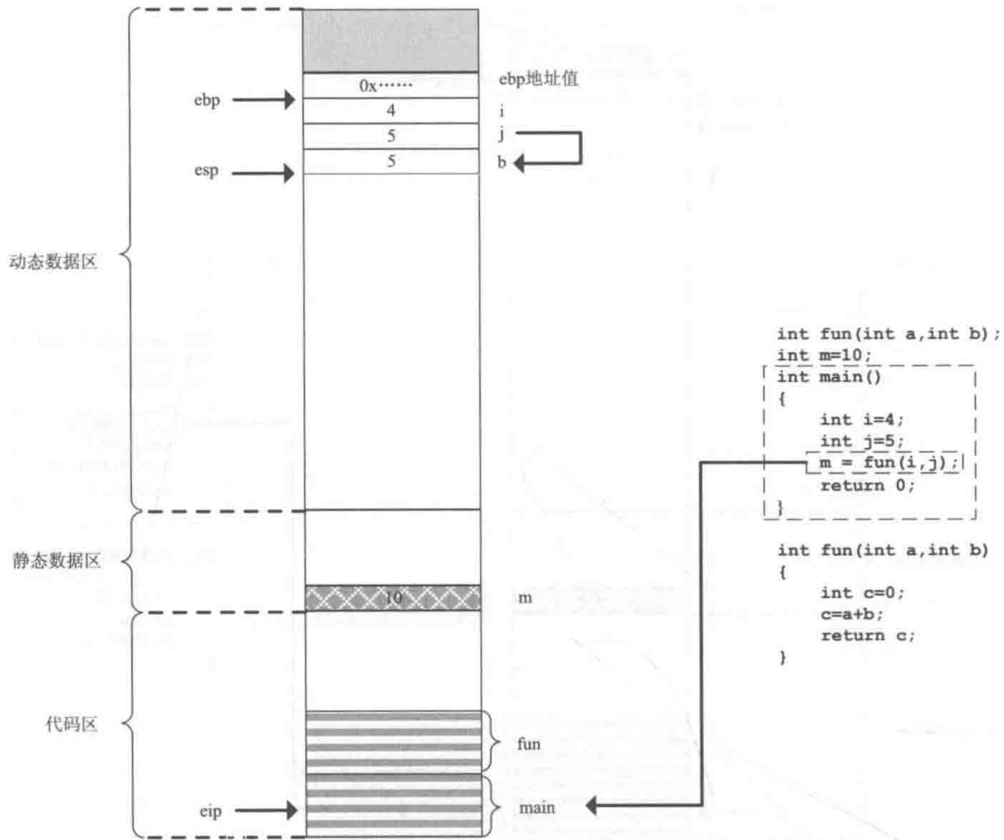



图 1-12 j 的数值作为参数被压栈

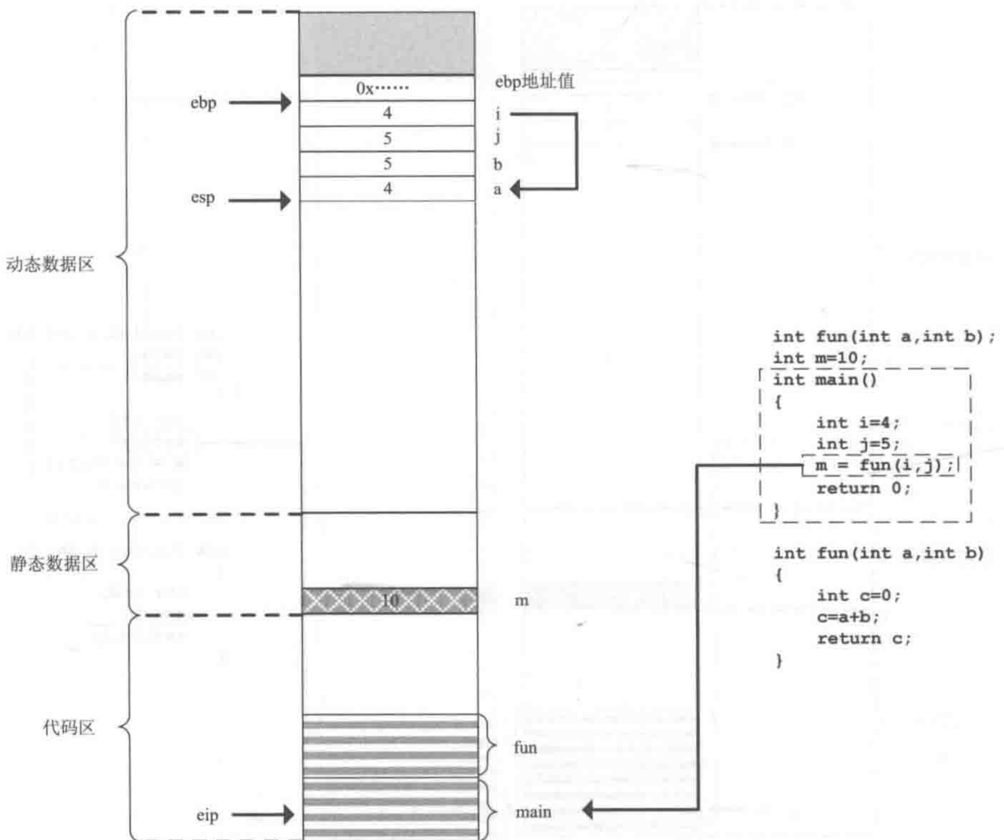



图 1-13 i 的数值作为参数被压栈

到这里，函数调用的数据准备工作就完成了。另一部分就是跳转到被调用的函数的第一条指令去执行，情景如图 1-16 所示。

试读结束：需要全本请在线购买： www.ertongbook.com