第 1 章
Assembly Language for x86 Processors, Seventh Edition

基本概念

本章将建立汇编语言编程的一些核心概念。比如，汇编语言是如何适应各种语言和应用程序的。本章还将介绍虚拟机概念，它在理解软件与硬件层之间的关系时非常重要。本章还用大量的篇幅说明二进制和十六进制的数制系统，展示如何执行转换和基本的算术运算。本章的最后将介绍基础逻辑操作（AND、OR 和 NOT），后续章节将证明这些操作是很重要的。

1.1 欢迎来到汇编语言的世界

本书主要介绍与运行 Microsoft Windows 32 位和 64 位系统的 Intel 和 AMD 处理器相兼容的微处理器编程。

配合本书应使用 Microsoft 宏汇编器（称为 MASM）的最新版本。Microsoft Visual Studio 的大多数版本（专业版，旗舰版，精简版……）都包含 MASM。请访问我们的网站（asmirvine.com），以便了解 Visual Studio 对 MASM 支持的最新详细信息。同时，网站中还包括很多关于如何设置软件并开始使用的有用信息。

在运行 Microsoft Windows 的 x86 系统中，其他一些有名的汇编器包括：TASM（Turbo 汇编器），NASM（Netwide 汇编器）和 MASM32（MASM 的一种变体）。GAS（GNU 汇编器）和 NASM 是两种基于 Linux 的汇编器。在这些汇编器中，NASM 的语法与 MASM 的最相似。

汇编语言是最古老的编程语言，在所有的语言中，它与原生机器语言最为接近。它能直接访问计算机硬件，要求用户了解计算机架构和操作系统。

教育价值 为什么要读这本书？也许读者正在学的大学课程的名称与下列课程之一相似，而这些课程经常使用这本书：

- 微计算机汇编语言
- 汇编语言编程
- 计算机体系结构导论
- 计算机系统基础
- 嵌入式系统编程

本书有助于学习计算机体系结构、机器语言和底层编程的基本原理。读者可以学到足够的汇编语言，来测试其掌握的当今使用最广泛的微处理器系列的知识。读者不会学到用模拟汇编器来编写一个“玩具”计算机；MASM 是一个由业界专业人士使用的工业级汇编器。读者将从程序员的角度来了解 Intel 处理器系列的体系结构。

如果读者计划成为 C 或 C++ 开发者，就需要理解内存、地址和指令是如何在底层工作的。在高级语言层次上，很多编程错误不容易被识别。因此，程序员经常会发现需要“深入”到程序内部，才能找出程序不工作的原因。

如果读者对底层编程和学习计算机软硬件细节的价值有所怀疑，请注意以下描述，它引用自首席计算机科学家 Donald Knuth 对其著名丛书《计算机程序设计艺术》的讨论：

有人说使用机器语言，从根本上来说，是我所犯的极大错误。但是我真的认为，只有有能力讨论底层细节，才可以为严肃的计算机程序员写书[1]。

2 登录本书网站 www.asmirvine.com，可获取大量的补充信息、教程和练习。

1.1.1 读者可能会问的问题

需要怎样的背景知识？ 在阅读本书之前，读者至少使用过一种结构化高级语言进行编程，如 Java、C、Python 或 C++。需要了解如何使用 IF 语句、数组和函数来解决编程问题。

什么是汇编器和链接器？ 汇编器（assembler）是一种工具程序，用于将汇编语言源程序转换为机器语言。链接器（linker）也是一种工具程序，它把汇编器生成的单个文件组合为一个可执行程序。还有一个相关的工具，称为调试器（debugger），使程序员可以在程序运行时，单步执行程序并检查寄存器和内存状态。

需要哪些硬件和软件？ 一台运行 32 位或 64 位 Microsoft Windows 系统的计算机，并已安装了近期版本的 Microsoft Visual Studio。

MASM 能创建哪些类型的程序？

- 32 位保护模式（32-Bit Protected Mode）：32 位保护模式程序运行于所有的 32 位和 64 位版本的 Microsoft Windows 系统。它们通常比实模式程序更容易编写和理解。从现在开始，将其简称为 32 位模式。
- 64 位模式（64-Bit Mode）：64 位程序运行于所有的 64 位版本 Microsoft Windows 系统。
- 16 位实地址模式（16-Bit Real-Address Mode）：16 位程序运行于 32 位版本 Windows 和嵌入式系统。由于 64 位 Windows 不支持这类程序，本书只限在第 14 章～第 17 章讨论这种模式。这些章节是电子版的，可以在出版社网站上获得。

本书有哪些补充资料？ 本书网站（www.asmirvine.com）上有如下资料：

- 汇编语言工作手册：一系列的教程。
- 64 位、32 位和 16 位编程的 Irvine 64、Irvine 32 和 Irvine 16 子程序库，及其完整源代码。
- 本书示例程序的所有源代码。
- 勘误表。
- 入门，帮助建立 Visual Studio 以使用 Microsoft 汇编器的详细教程。
- 关于高级主题的文章，限于篇幅，它们没有包含在本书的印刷版内。
- 在线论坛的链接，从论坛上可以获得其他使用本书的专家的帮助。

能学到什么？ 本书将使读者更好地了解数据表示、调试、编程和硬件控制。读者将学到：

- x86 处理器应用的计算机体系结构的基本原理。
- 基本布尔逻辑，以及它是如何应用于编程和计算机硬件的。
- 使用保护模式和虚模式时，x86 处理器如何管理内存。

3

- 高级语言编译器（如 C++）如何将其语句转换为汇编语言和原生机器代码。
- 高级语言如何在机器级实现算术表达式、循环和逻辑结构。
- 数据表示，包括有符号和无符号整数、实数以及字符数据。
- 如何在机器级调试程序。使用 C 和 C++ 语言时，它们生成的是原生机器代码，这个技术显得至关重要。

- 应用程序如何通过中断处理程序和系统调用与计算机操作系统进行通信。
- 如何连接汇编语言代码与 C++ 程序。
- 如何创建汇编语言应用程序。

汇编语言与机器语言有什么关系？ 机器语言（machine language）是一种数字语言，专门设计成能被计算机处理器（CPU）理解。所有 x86 处理器都理解共同的机器语言。汇编语言（assembly language）包含用短助记符如 ADD、MOV、SUB 和 CALL 书写的语句。汇编语言与机器语言是一对一（one-to-one）的关系：每一条汇编语言指令对应一条机器语言指令。

C++ 和 Java 与汇编语言有什么关系？ 高级语言如 Python、C++ 和 Java 与汇编语言和机器语言的关系是一对多（one-to-many）。比如，C++ 的一条语句就会扩展为多条汇编指令或机器指令。大多数人无法阅读原始机器代码，因此，本书探讨的是与之最接近的汇编语言。例如，下面的 C++ 代码进行了两个算术操作，并将结果赋给一个变量。假设 X 和 Y 是整数：

```
int   Y;
int   X = (Y + 4) * 3;
```

与之等价的汇编语言程序如下所示。这种转换需要多条语句，因为每条汇编语句只对应一条机器指令：

```
mov   eax,Y   ;Y 送入 EAX 寄存器
add   eax,4   ;EAX 寄存器内容加 4
mov   ebx,3   ;3 送入 EBX 寄存器
imul  ebx     ;EAX 与 EBX 相乘
mov   X,eax   ;EAX 的值送入 X
```

（寄存器（register）是 CPU 中被命名的存储位置，用于保存操作的中间结果。）这个例子的重点不是说明 C++ 与汇编语言哪个更好，而是展示它们的关系。

汇编语言可移植吗？ 一种语言，如果它的源程序能够在各种各样的计算机系统中进行编译和运行，那么这种语言被称为是可移植的（portable）。例如，一个 C++ 程序，除非需要特别引用某种操作系统的库函数，否则它就几乎可以在任何一台计算机上编译和运行。Java 语言的一大特点就是，其编译好的程序几乎能在所有计算机系统中运行。

汇编语言不是可移植的，因为它是为特定处理器系列设计的。目前广泛使用的有多种不同的汇编语言，每一种都基于一个处理器系列。对于一些广为人知的处理器系列如 Motorola 68x00、x86、SUN Sparc、Vax 和 IBM-370，汇编语言指令会直接与该计算机体系结构相匹配，或者在执行时用一种被称为微代码解释器（microcode interpreter）的处理器内置程序来进行转换。

4

为什么要学习汇编语言？ 如果对学习汇编语言还心存疑虑，考虑一下这些观点：

- 如果是学习计算机工程，那么很可能会被要求写嵌入式（embedded）程序。嵌入式程序是指一些存放在专用设备中小容量存储器内的短程序，这些专用设备包括：电话、汽车燃油和点火系统、空调控制系统、安全系统、数据采集仪器、显卡、声卡、硬盘驱动器、调制解调器和打印机。由于汇编语言占用内存少，因此它是编写嵌入式程序的理想工具。
- 处理仿真和硬件监控的实时应用程序要求精确定时和响应。高级语言不会让程序员对编译器生成的机器代码进行精确控制。汇编语言则允许程序员精确指定程序的可

执行代码。

- 电脑游戏要求软件在减少代码大小和加快执行速度方面进行高度优化。就针对一个目标系统编写能够充分利用其硬件特性的代码而言，游戏程序员都是专家。他们经常选择汇编语言作为工具，因为汇编语言允许直接访问计算机硬件，所以，为了提高速度可以对代码进行手工优化。
- 汇编语言有助于形成对计算机硬件、操作系统和应用程序之间交互的全面理解。使用汇编语言，可以运用并检验从计算机体系结构和操作系统课程中获得的理论知识。
- 一些高级语言对其数据表示进行了抽象，这使得它们在执行底层任务时显得有些不方便，如位控制。在这种情况下，程序员常常会调用使用汇编语言编写的子程序来完成他们的任务。
- 硬件制造商为其销售的设备创建设备驱动程序。*设备驱动程序*（device driver）是一种程序，它把通用操作系统指令转换为对硬件细节的具体引用。比如，打印机制造商就为他们销售的每一种型号都创建了一种不同的 MS-Windows 设备驱动程序。通常，这些设备驱动程序包含了大量的汇编语言代码。

汇编语言有规则吗？ 大多数汇编语言规则都是以目标处理器及其机器语言的物理局限性为基础的。比如，CPU 要求两个指令操作数的大小相同。与 C++ 或 Java 相比，汇编语言的规则较少，因为，前者是用语法规则来减少意外的逻辑错误，而这是以限制底层数据访问为代价的。汇编语言程序员可以很容易地绕过高级语言的限制性特征。例如，Java 就不允许访问特定的内存地址。程序员可以使用 JNI（Java Native Interface）类来调用 C 函数绕过这个限制，可结果程序不容易维护。反之，汇编语言可以访问所有的内存地址。但这种自由的代价也很高：汇编语言程序员需要花费大量的时间进行调试！

5

1.1.2 汇编语言的应用

早期在编程时，大多数应用程序部分或全部用汇编语言编写。它们不得不适应小内存，并尽可能在慢速处理器上有效运行。随着内存容量越来越大，以及处理器速度急速提高，程序变得越来越复杂。程序员也转向高级语言如 C、FORTRAN 和 COBOL，这些语言具有很多结构化能力。最近，Python、C++、C# 和 Java 等面向对象语言已经能够编写含数百万行代码的复杂程序了。

很少能看到完全用汇编语言编写的大型应用程序，因为它们需要花费大量的时间进行编写和维护。不过，汇编语言可以用于优化应用程序的部分代码来提升速度，或用于访问计算机硬件。表 1-1 比较了汇编语言和高级语言对各种应用类型的适应性。

表 1-1 汇编语言与高级语言的比较

应用类型	高级语言	汇编语言
商业或科学应用程序，为单一的中型或大型平台编写	规范结构使其易于组织和维护大量代码	最小规范结构，因此必须由具有不同程度经验的程序员来维护结构。这导致对已有代码的维护困难
硬件设备驱动程序	语言不一定提供对硬件的直接访问。即使提供了，可能也需要难以控制的编码技术，这导致维护困难	对硬件的访问直接且简单。当程序较短且文档良好时易于维护
为多个平台（不同的操作系统）编写的商业或科学应用程序	通常可移植。在每个目标操作系统上，源程序只做少量修改就能重新编译	需要为每个平台单独重新编写代码，每个汇编器都使用不同的语法。维护困难

（续）

应用类型	高级语言	汇编语言
需要直接访问硬件的嵌入式系统和电脑游戏	可能生成很大的可执行文件，以至于超出设备的内存容量	理想，因为可执行代码小，运行速度快

C 和 C++ 语言具有一个独特的特性，能够在高级结构和底层细节之间进行平衡。直接访问硬件是可能的，但是完全不可移植。大多数 C 和 C++ 编译器都允许在其代码中嵌入汇编语句，以提供对硬件细节的访问。

1.1.3 本节回顾

1. 汇编器和链接器是如何一起工作的？
2. 学习汇编语言如何能提高你对操作系统的理解？ 6
3. 比较高级语言和机器语言时，一对多关系是什么意思？
4. 解释编程语言中的可移植性概念。
5. x86 处理器的汇编语言与 Vax 或 Motorola 68x00 等机器的汇编语言是一样的吗？
6. 举一个嵌入式系统应用程序的例子。
7. 什么是设备驱动程序？
8. 汇编语言和 C/C++ 语言中的指针变量类型检查，哪一个更强（更严格）？
9. 给出两种应用类型，与高级语言相比，它们更适合使用汇编语言。
10. 编写程序来直接访问打印机端口时，为什么高级语言不是理想工具？
11. 为什么汇编语言不常用于编写大型应用程序？
12. 挑战：参考本章前面给出的例子，将下述 C++ 表达式转换为汇编语言：X=（Y*4）+ 3。

1.2 虚拟机概念

虚拟机概念（virtual machine machine）是一种说明计算机硬件和软件关系的有效方法。在安德鲁·塔嫩鲍姆（Andrew Tanenbaum）的书《结构化计算机组织》（*Structured Computer Organization*）中可以找到对这个模型广为人知的解释。要说明这个概念，先从计算机的最基本功能开始，即执行程序。

计算机通常可以执行用其原生机器语言编写的程序。这种语言中的每一条指令都简单到可以用相对少量的电子电路来执行。为了简便，称这种语言为 L0。

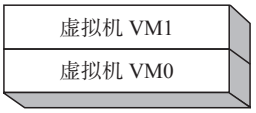
由于 L0 极其详细，并且只由数字组成，因此，程序员用其编写程序就非常困难。如果能够构造一种较易使用的新语言 L1，那么就可以用 L1 编写程序。有两种实现方法：

- 解释（Interpretation）：运行 L1 程序时，它的每一条指令都由一个用 L0 语言编写的程序进行译码和执行。L1 程序可以立即开始运行，但是在执行之前，必须对每条指令进行译码。
- 翻译（Translation）：由一个专门设计的 L0 程序将整个 L1 程序转换为 L0 程序。然后，得到的 L0 程序就可以直接在计算机硬件上执行。

1. 虚拟机

与只使用语言描述相比，把每一层都想象成有一台假设的计算机或者虚拟机会更容易一些。通俗地说，虚拟机可以定义为一个软件程序，用来模拟一些其他的物理或虚拟计算机的功能。虚拟机，将其称为 VM1，可以执行 L1 语言编写的指令。虚拟机 VM0 可以执行 L0 7

语言编写的指令：



每一个虚拟机既可以用硬件构成也可以用软件构成。程序员可以为虚拟机 VM1 编写程序，如果能把 VM1 当作真实计算机予以实现，那么，程序就能直接在这个硬件上执行。否则，用 VM1 写出的程序就被翻译 / 解释为 VM0 程序，并在机器 VM0 上执行。

机器 VM1 与 VM0 之间的差异不能太大，否则，翻译或解释花费的时间就会非常多。如果 VM1 语言对程序员来说还不够友好到足以用于应用程序的开发呢？可以为此设计另一个更加易于理解的虚拟机 VM2。这个过程能够不断重复，直到虚拟机 VM*n* 足够支持功能强大、使用方便的语言。

Java 编程语言就是以虚拟机概念为基础的。Java 编译器把用 Java 语言编写的程序翻译为 Java 字节码（Java byte code）。后者是一种低级语言，能够在运行时由 Java 虚拟机（JVM）程序快速执行。JVM 已经在许多不同的计算机系统上实现了，这使得 Java 程序相对而言独立于系统。

2. 特定的机器

与实际机器和语言相对，用 Level 2 表示 VM2，Level 1 表示 VM1，如图 1-1 所示。计算机数字逻辑硬件表示为 Level 1 机器。其上是 Level 2，称为指令集架构（ISA，Instruction Set Architecture）。通常，这是用户可以编程的第一个层次，尽管这种程序包含的是被称为机器语言的二进制数值。

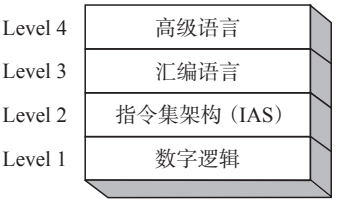


图 1-1 虚拟机层次结构

指令集架构（Level 2） 计算机芯片制造商在处理器内部设计一个指令集来实现基本操作，如传送、加法或乘法。这个指令集也被称为机器语言。每一个机器语言指令或者直接在机器硬件上执行，或者由嵌入到微处理器芯片的程序来执行，该程序被称为微程序。微程序的讨论不在本书范围内，如果想要了解其更多细节，可以参阅 Tanenbaum 的著作。

汇编语言（Level 3） 在 ISA 层，编程语言提供了一个翻译层，来实践大规模软件开发。汇编语言出现在 Level3，使用短助记符，如 ADD、SUB 和 MOV，易于转换到 ISA 层。

8 汇编语言程序在执行之前要全部翻译（汇编）为机器语言。

高级语言（Level 4） Level4 是高级编程语言，如 C、C++ 和 Java。这些语言程序所包含的语句功能强大，并翻译为多条汇编语言指令。比如，查看 C++ 编译器生成的列表文件输出，就可以看到这样的翻译。汇编语言代码由编译器自动汇编为机器语言。

本节回顾

1. 用自己的话描述虚拟机概念。
2. 为什么认为翻译的程序比解释的程序执行起来更快？
3.（真 / 假）：当 L1 语言编写的解释程序运行时，其每一条指令都由用 L0 语言编写的程序进行解码和执行。
4. 当处理不同虚拟机层次的语言时，说明翻译的重要性。
5. 本节给出的虚拟机示例中，汇编语言出现在哪一层？

6. 什么软件程序使得被编译的 Java 程序能够在几乎所有计算机上运行？
7. 从低到高，说出本节命名的四个虚拟机层次。
8. 为什么程序员不用机器语言编写应用程序？
9. 图 1-1 中，哪个虚拟机层次使用机器语言？
10. 汇编语言虚拟机的语句被翻译为哪个层次的语句？

1.3 数据表示

汇编语言程序员处理的是物理级数据，因此他们必须善于检查内存和寄存器。通常，二进制数被用于描述计算机内存的内容；有时也使用十进制和十六进制数。所以必须熟练掌握数字格式，以便快速地进行数字的格式转换。

9

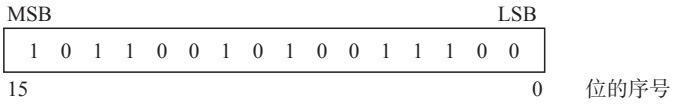
每一种数制格式或系统，都有一个基数（base），也就是可以分配给单一数字的最大符号数。表 1-2 给出了数制系统内可能的数字，这些系统是硬件和软件手册中最常使用的。在表的最后一行，十六进制使用的是数字 0 到 9，然后字母 A 到 F 表示十进制数 10 到 15。在展示计算机内存的内容和机器级指令时，使用十六进制是相当常见的。

表 1-2 二进制、八进制、十进制和十六进制数字

系统	基数	可能的数字
二进制	2	01
八进制	8	01234567
十进制	10	0123456789
十六进制	16	0123456789ABCDEF

1.3.1 二进制整数

计算机以电子电荷集合的形式在内存中保存指令和数据。用数字来表示这些内容就需要系统能够适应开 / 关（on/off）或真 / 假（true/false）的概念。二进制数（binary number）用 2 个数字作基础，其中每一个二进制数字（称为位，bit）不是 0 就是 1。位自右向左，从 0 开始顺序增量编号。左边的位称为最高有效位（Most Significant Bit，MSB），右边的位称为最低有效位（LSB，least significant bit）。一个 16 位的二进制数，其 MSB 和 LSB 如下图所示：



二进制整数可以是有符号的，也可以是无符号的。有符号整数又分为正数和负数，无符号整数默认为正数，零也被看作是正数。在书写较大的二进制数时，有些人喜欢每 4 位或 8 位插入一个点号，以增加数字的易读性。比如，1101.1110.0011.1000.0000 和 11001010.10101100。

1. 无符号二进制整数

从 LSB 开始，无符号二进制整数中的每一个位代表的是 2 的加 1 次幂。下图展示的是，对一个 8 位的二进制数来说，2 的幂是如何从右到左增加的：

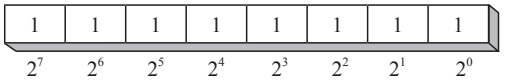


表 1-3 列出了从 2^0 到 2^{15} 的十进制值。

10

表 1-3 二进制位的位置对应值

2^n	十进制值	2^n	十进制值
2^0	1	2^8	256
2^1	2	2^9	512
2^2	4	2^{10}	1024
2^3	8	2^{11}	2048
2^4	16	2^{12}	4096
2^5	32	2^{13}	8192
2^6	64	2^{14}	16384
2^7	128	2^{15}	32768

2. 无符号二进制整数到十进制数的转换

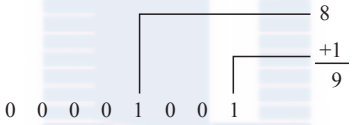
对于一个包含 n 个数字的无符号二进制整数来说，*加权位记数法*（weighted positional notation）提供了一种简便的方法来计算其十进制值：

$$dec = (D_{n-1} \times 2^{n-1}) + (D_{n-2} \times 2^{n-2}) + \cdots + (D_1 \times 2^1) + (D_0 \times 2^0)$$

D 表示一个二进制数字。比如，二进制数 00001001 就等于 9。计算该值时，剔除了数字等于 0 的位：

$$(1 \times 2^3) + (1 \times 2^0) = 9$$

下图表示了同样的计算过程：



3. 无符号十进制整数到二进制数的转换

将无符号十进制整数转换为二进制，方法是不断将这个整数除以 2，并将每个余数记录为一个二进制数字。下表展示的是十进制数 37 转换为二进制数的步骤。余数的数字，从第二行开始，分别表示的是二进制数字 D_0、D_1、D_2、D_3、D_4 和 D_5：

除法	商	余数	除法	商	余数
37/2	18	1	4/2	2	0
18/2	9	0	2/2	1	0
9/2	4	1	1/2	0	1

11

将表中余数列的二进制位逆序连接（D_5，D_4，…），就得到了该整数的二进制值 100101。由于计算机总是按照 8 的倍数来组织二进制数字，因此在该二进制数的左边增加两个 0，形成 00100101。

> **提示** 有多少位呢？设无符号十进制值为 n，其对应的二进制数的位数为 b，用一个简单的公式就可以计算出 b：b=（$\log_2 n$）的上限。比如，如果 n=17，则 $\log_2 17$=4.087 463，取其上限的最小整数 5。大多数计数器没有以 2 为底的对数运算，但是有些网页可以帮助实现这种计算。

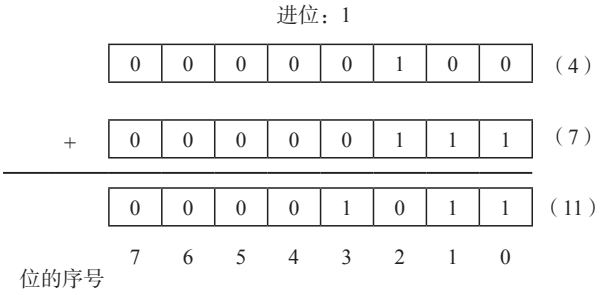
1.3.2 二进制加法

两个二进制整数相加时，是位对位处理的，从最低的一对位（右边）开始，依序将每一

对位进行加法运算。两个二进制数字相加，有四种结果，如下所示：

0 + 0 = 0	0 + 1 = 1
1 + 0 = 1	1 + 1 = 10

1 与 1 相加的结果是二进制的 10（等于十进制的 2）。多出来的数字向更高位产生一个进位。如下图所示，两个二进制数 0000 0100 和 0000 0111 相加：



从两个数的最低位（位 0）开始，计算 0+1，得到底行对应位上的 1。然后计算次低位（位 1）。在位 2 上，计算 1+1，结果是 0，并产生一个进位 1。然后计算位 3，0+0，还要加上位 2 的进位，结果是 1。其余的位都是 0。上图右边是等价的十进制数值加法（4+7=11），可以用于验证左边的二进制加法。

有些情况下，最高有效位会产生进位。这时，预留存储区的大小就显得很重要。比如，如果计算 1111 1111 加 0000 0001，就会在最高有效位之外产生一个 1，而和数的低 8 位则为全 0。如果和数的存储大小最少有 9 位，那么就可以将和数表示为 1 0000 0000。但是，如果和数只能保存 8 位，那么它就等于 0000 0000，也就是计算结果的低 8 位。 12

1.3.3 整数存储大小

在 x86 计算机中，所有数据存储的基本单位都是字节（byte），一个字节有 8 位。其他的存储单位还有字（word）（2 个字节），双字（doubleword）（4 个字节）和四字（quadword）（8 个字节）。下图展示了每个存储单位所包含的位的个数：

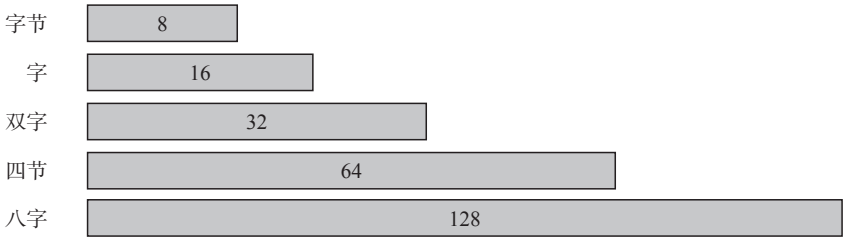


表 1-4 列出了所有无符号整数可能的取值范围。

大的度量单位 对内存和磁盘空间而言，还可以使用大的度量单位：

- 1 千字节（kilobyte）等于 2^{10}，或 1024 个字节。
- 1 兆字节（megabyte）（1MB）等于 2^{20}，或 1 048 576 字节。
- 1 吉字节（gigabyte）（1GB）等于 2^{30}，即 1024^3，或 1 073 741 824 字节。
- 1 太字节（terabyte）（1TB）等于 2^{40}，即 1024^4，或 1 099 511 627 776 字节。
- 1 拍字节（petabyte）等于 2^{50}，或 1 125 899 906 842 624 字节。

- 1 艾字节（exabyte）等于 2^{60}，或 1 152 921 504 606 846 976 字节。
- 1 泽字节（zettabyte）等于 2^{70} 个字节。
- 1 尧字节（yottabyte）等于 2^{80} 个字节。

表 1-4 无符号整数类型的取值范围和大小

类型	取值范围	按位计的存储大小	类型	取值范围	按位计的存储大小
无符号字节	0 到 2^8-1	8	无符号四字	0 到 $2^{64}-1$	64
无符号字	0 到 $2^{16}-1$	16	无符号八字	0 到 $2^{128}-1$	128
无符号双字	0 到 $2^{32}-1$	32			

1.3.4 十六进制整数

大的二进制数读起来很麻烦，因此十六进制数字就提供了一种简便的方式来表示二进制数据。十六进制整数中的 1 个数字就表示了 4 位二进制位，两个十六进制数字就能表示一个字节。一个十六进制数字表示的范围是十进制数 0 到 15，所以，用字母 A 到 F 来代表十进制数 10 到 15。表 1-5 列出了每个 4 位二进制序列如何转换为十进制和十六进制数值。

13

表 1-5 二进制、十进制和十六进制等值表

二进制	十进制	十六进制	二进制	十进制	十六进制
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	10	A
0011	3	3	1011	11	B
0100	4	4	1100	12	C
0101	5	5	1101	13	D
0110	6	6	1110	14	E
0111	7	7	1111	15	F

下面的例子说明了二进制数 0001 0110 1010 0111 1001 0100 是如何与十六进制数 16A794 等价的。

1	6	A	7	9	4
0001	0110	1010	0111	1001	0100

1. 无符号十六进制数到十进制的转换

十六进制数中，每一个数字位都代表了 16 的幂。这有助于计算一个十六进制整数的十进制值。假设用下标来对一个包含 4 个数字的十六进制数编号 $D_3D_2D_1D_0$。下式计算了这个整数的十进制值：

$$dec = (D_3 \times 16^3) + (D_2 \times 16^2) + (D_1 \times 16^1) + (D_0 \times 16^0)$$

这个表达式可以推广到任意 n 位数的十六进制整数：

$$dec = (D_{n-1} \times 16^{n-1}) + (D_{n-2} \times 16^{n-2}) + \cdots + (D_1 \times 16^1) + (D_0 \times 16^0)$$

> 一般情况下，可以通过公式把基数为 B 的任何 n 位整数转换为十进制数：$dec = (D_{n-1} \times B^{n-1}) + (D_{n-2} \times B^{n-2}) + \cdots + (D_1 \times B^1) + (D_0 \times B^0)$。

比如，十六进制数 1234 就等于 $(1 \times 16^3) + (2 \times 16^2) + (3 \times 16^1) + (4 \times 16^0)$，也就是十进制数 4660。同样，十六进制数 3BA4 等于 $(3 \times 16^3) + (11 \times 16^2) + (10 \times 16^1) + (4 \times 16^0)$，

也就是十进制数 15 268。下图演示了第二个数转换的计算过程：

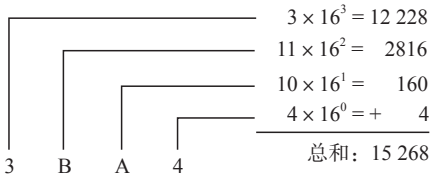


表 1-6 列出了 16 的幂从 16^0 到 16^7 的十进制数值。 14

表 1-6 以 16 为底的幂函数的十进制值

16^n	十进制值	16^n	十进制值
16^0	1	16^4	65 536
16^1	16	16^5	1 048 576
16^2	256	16^6	16 777 216
16^3	4096	16^7	268 435 456

2. 无符号十进制数到十六进制的转换

无符号十进制整数转换到十六进制数的过程是，把这个十进制数反复除以 16，每次取余数作为一个十六进制数字。例如，下表列出了十进制数 422 转换为十六进制的步骤：

除法	商	余数
422/16	26	6
26/16	1	A
1/16	0	1

表中，余数列的数字按照最后一行到第一行的顺序，组合为十六进制的结果。因此本例中，十六进制结果就表示为 1A6。同样的算法也适用于 1.3.1 节中的二进制整数。如果要将十进制数转换为其他进制数，就在计算时把除数（16）换成相应的基数。

1.3.5 十六进制加法

调试工具程序（称为调试器，debugger）通常用十六进制表示内存地址。为了定位一个新地址常常需要将两个地址相加。幸运的是，十六进制加法与十进制加法是一样的，只需要更换基数就可以了。

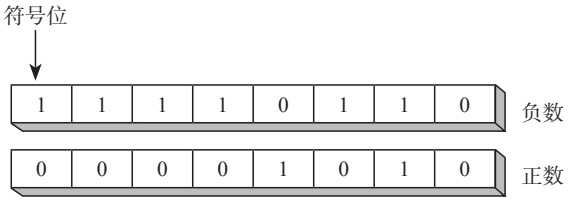
假设现在要将两个数 X 和 Y 相加，其基数为 b。对它们数字的编号从最低位（x_0）开始直到最高位。将 X 和 Y 中对应位 x_i 和 y_i 相加得到和值 s_i。如果 $s_i \geqslant b$，则再计算 $s_i = (s_i \text{ MOD } b)$，并产生一个进位 1。当计算下一对数字 x_{i+1} 和 y_{i+1} 的和时，将该进位加入和值。

比如，现在将两个十六进制数 6A2 和 49A 相加。在最低位上 2+A=12（十进制数），没有进位，用十六进制数 C 表示这个和值。在中间位上 A+9=19（十进制数），由于 19 ⩾ 16（基数），因此有进位。再计算 19 MOD 16=3，并向第 3 位产生一个进位 1。最后，在最高位上计算 1+6+4=11（十进制数），则在和数的第 3 位上为十六进制数 B。所以，整个和数的十六进制数为 B3C。 15

进位	1		
X	6	A	2
Y	4	9	A
S	B	3	C

1.3.6 有符号二进制整数

有符号二进制整数有正数和负数。在 x86 处理器中，MSB 表示的是符号位：0 表示正数，1 表示负数。下图展示了 8 位的正数和负数：



1. 补码表示

负整数用补码（two's-complement）表示时，使用的数学原理是：一个整数的补码是其加法逆元。（如果将一个数与其加法逆元相加，结果为 0。）

补码表示法对处理器设计者来说很有用，因为有了它就不需要用两套独立的电路来处理加法和减法。例如，如果表达式为 *A*-*B*，则处理器就可以很方便地将其转换为加法表达式：*A*+（−*B*）。

将一个二进制整数按位取反（求补）再加 1，就形成了它的补码。以 8 位二进制数 0000 0001 为例，求其补码为 1111 1111，过程如下所示：

初始值	00000001
第一步：按位取反	11111110
第二步：将上一步得到的结果加 1	11111110 +00000001
和值：补码表示	11111111

1111 1111 是 −1 的补码。补码操作是可逆的，因此，1111 1111 的补码就是 0000 0001。

十六进制数的补码 将一个十六进制整数按位取反并加 1，就生成了它的补码。一个简单的十六进制数字取反方法就是用 15 减去该数字。下面是一些十六进制数求补码的例子：

16

```
6A3D --> 95C2 + 1 --> 95C3
95C3 --> 6A3C + 1 --> 6A3D
```

有符号二进制数到十进制的转换 用下面的算法计算一个有符号二进制整数的十进制数值：

- 如果最高位是 1，则该数是补码。再次对其求补，得到其正数值。然后把这个数值看作是一个无符号二进制整数，并求它的十进制数值。
- 如果最高位是 0，就将其视为无符号二进制整数，并转换为十进制数。

例如，有符号二进制数 1111 0000 的最高有效位是 1，这意味着它是一个负数，首先要求它的补码，然后再将结果转换为十进制。过程如下所示：

初始值	11110000
第一步：按位取反	00001111
第二步：将上一步得到的结果加 1	00001111 + 1
第三步：生成补码	00010000
第四步：转换为十进制	16

由于初始值（1111 0000）是负数，因此其十进制数值为 −16。

有符号十进制数到二进制的转换 有符号十进制整数转换为二进制的步骤如下：

1）把十进制整数的绝对值转换为二进制数。

2）如果初始十进制数是负数，则在第 1 步的基础上，求该二进制数的补码。

比如，十进制数 -43 转换为二进制的过程为：

1）无符号数 43 的二进制表示为 0010 1011。

2）由于初始数值是负数，因此，求出 0010 1011 的补码 1101 0101。这就是十进制数 −43 的二进制表示。

有符号十进制数到十六进制的转换 有符号十进制整数转换为十六进制的步骤如下：

1）把十进制整数的绝对值转换为十六进制数。

2）如果初始十进制数是负数，则在第 1 步的基础上，求该十六进制数的补码。

有符号十六进制数到十进制的转换 有符号十六进制整数转换为十进制的步骤如下：

1）如果十六进制整数是负数，求其补码，否则保持该数不变。

2）把第 1 步得到的整数转换为十进制。如果初始值是负数，则在该十进制整数的前面加负号。

> 通过检查十六进制数的最高有效（最高）位，就可以知道该数是正数还是负数。如果最高位≥ 8，该数是负数；如果最高位≤ 7，该数是正数。比如，十六进制数 8A20 是负数，而 7FD9 是正数。

17

2. 最大值和最小值

n 位有符号整数只用 *n*−1 来表示该数的范围。表 1-7 列出了有符号单字节、字、双字、四字和八字的最大值与最小值。

表 1-7 有符号整数类型的范围与大小

类型	范围	存储位数	类型	范围	存储位数
有符号字节	-2^7 到 $+2^7-1$	8	有符号四字	-2^{63} 到 $+2^{63}-1$	64
有符号字	-2^{15} 到 $+2^{15}-1$	16	有符号八字	-2^{127} 到 $+2^{127}-1$	128
有符号双字	-2^{31} 到 $+2^{31}-1$	32			

1.3.7 二进制减法

如果采用与十进制减法相同的方法，那么从一个较大的二进制数中减去一个较小的无符号二进制数就很容易了。示例如下：

```
  0 1 1 0 1     ( 十进制数 13)
- 0 0 1 1 1     ( 十进制数 7)
-----------
```

位 0 上的减法非常简单：

```
  0 1 1 0 1
- 0 0 1 1 1
-----------
          0
```

下一个位置上执行（0−1），要向左边的相邻位借 1。其结果是从 2 中减去 1：

```
  0 1 0 0 1
- 0 0 1 1 1
-----------
        1 0
```

再下一位上，又要向左边的相邻位借一位，并从 2 中减去 1：

```
  0 0 0 1 1
- 0 0 1 1 1
-----------
      1 1 0
```

最后，最高两位都执行的是零减去零：

```
  0 0 0 1 1
- 0 0 1 1 1
-----------
  0 0 1 1 0      十进制数 6
```

18

执行二进制减法还有更简单的方法，即将被减去数的符号位取反，然后将两数相加。这个方法要求用一个额外的位来保存数的符号。现在以刚才计算的（01101−00111）为例来试一下这个方法。首先，将 00111 按位取反（11000）加 1，得到 11001。然后，把两个二进制数值相加，并忽略最高位的进位：

```
0 1 1 0 1       (+13)
1 1 0 0 1       (-7)
---------
0 0 1 1 0       (+6)
```

结果正是我们预期的 +6。

1.3.8 字符存储

如果计算机只存储二进制数据，那么它如何表示字符呢？计算机使用的是字符集，将字符映射为整数。早期，字符集只用 8 位表示。即使是现在，在字符模式（如 MS-DOS）下运行时，IBM 兼容微机使用的还是 ASCII（读为“askey”）字符集。ASCII 是美国标准信息交换码（American Standard Code for Information Interchange）的首字母缩写。在 ASCII 中，每个字符都被分配了一个独一无二的 7 位整数。由于 ASCII 只用字节中的低 7 位，因此最高位在不同计算机上被用于创建其专有字符集。比如，IBM 兼容微机就用数值 128 ～ 255 来表示图形符号和希腊字符。

ANSI 字符集 美国国家标准协会（ANSI）定义了 8 位字符集来表示多达 256 个字符。前 128 个字符对应标准美国键盘上的字母和符号。后 128 个字符表示特殊字符，诸如国际字母表、重音符号、货币符号和分数。Microsoft Windows 早期版本使用 ANSI 字符集。

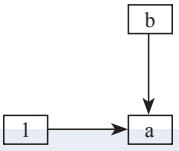
Unicode 标准 当前，计算机必须能表示计算机软件中世界上各种各样的语言。因此，Unicode 被创建出来，用于提供一种定义文字和符号的通用方法。它定义了数字代码（称为代码点（code point）），定义的对象为文字、符号以及所有主要语言中使用的标点符号，包括欧洲字母文字、中东的从右到左书写的文字和很多亚洲文字。代码点转换为可显示字符的格式有三种：

- UTF-8 用于 HTML，与 ASCII 有相同的字节数值。
- UTF-16 用于节约使用内存与高效访问字符相互平衡的环境中。比如，Microsoft Windows 近期版本使用了 UTF-16，其中的每个字符都有一个 16 位的编码。
- UTF-32 用于不考虑空间，但需要固定宽度字符的环境中。每个字符都有一个 32 位的编码。

ASCII 字符串 有一个或多个字符的序列被称为字符串（string）。更具体地说，一个

ASCII 字符串是保存在内存中的，包含了 ASCII 代码的连续字节。比如，字符串“ABC123”的数字代码是 41h、42h、43h、31h、32h 和 33h。*以空字节结束*（null-terminated）的字符串是指，在字符串的结尾处有一个为 0 的字节。C 和 C++ 语言使用的是以空字节结束的字符串，一些 Windows 操作系统函数也要求字符串使用这种格式。 19

使用 ASCII 表 本书文前的表格列出了在 Windows 控制台模式下运行时使用的 ASCII 码。在查找字符的十六进制 ASCII 码时，先沿着表格最上面一行，再找到包含要转换字符的列即可。表格第二行是该十六进制数值的最高位；左起第二列是最低位。例如，要查找字母 a 的 ASCII 码，先找到包含该字母的列，在这一列第二行中找到第一个十六进制数字 6。然后，找到包含 a 的行的左起第二列，其数字为 1。因此，a 的 ASCII 码是十六进制数 61。下图用简单的形式说明了这个过程：



ASCII 控制字符 0 ～ 31 的字符代码被称为 ASCII 控制字符。若程序用这些代码编写标准输出（比如 C++ 中），控制字符就会执行预先定义的动作。表 1-8 列出了该范围内最常用的字符，完整列表参见本书文前。

表 1-8 ASCII 控制字符

ASCII 码（十进制）	说明	ASCII 码（十进制）	说明
8	回退符（向左移动一列）	12	换页符（移动到下一个打印页）
9	水平制表符（向前跳过 *n* 列）	13	回车符（移动到最左边的输出列）
10	换行符（移动到下一个输出行）	27	换码符

数字数据表示术语 用精确的术语描述内存中和显示屏上的数字及字符是非常重要的。比如，在内存中用单字节保存十进制数 65，形式为 0100 0001。调试程序可能会将该字节显示为“41”，这个数字的十六进制形式。如果这个字节复制到显存中，则显示屏上可能显示字母“A”，因为在 ASCII 码中，0100 0001 代表的是字母 A。由于数字的解释可以依赖于它的上下文，因此，下面为每个数据表示类型分配一个特定的名称，以便将来的讨论更加清晰：

- *二进制整数*是指，以其原始格式保存在内存中的整数，以备用于计算。二进制整数保存形式为 8 位的倍数（如 8、16、32 或 64）。 20
- *数字字符串*是一串 ASCII 字符，例如“123”或“65”。这是一种简单的数字表示法，表 1-9 以十进制数 65 为例，列出了这种表示法能使用的各种形式。

表 1-9 数字字符串类型

格式	数值	格式	数值
二进制数字字符串	“01000001”	十六进制数字字符串	“41”
十进制数字字符串	“65”	八进制数字字符串	“101”

1.3.9 本节回顾

1. 术语解释：*最低有效位*（LSB）。
2. 下列无符号二进制整数的十进制表示分别是什么？

a. 11111000　　b. 11001010　　c. 11110000

3. 下列每组二进制整数的和分别是多少？

a. 00001111 + 00000010　　b. 11010101 + 01101011

c. 00001111 + 00001111

4. 下列每种数据类型各包含多少个字节？

a. 字　　b. 双字　　c. 四字　　d. 八字

5. 若要表示下列无符号十进制整数，则最少需要几位二进制数位？

a. 65　　b. 409　　c.16385

6. 写出下列二进制数的十六进制表示。

a. 0011 0101 1101 1010　　b. 1100 1110 1010 0011

c. 1111 1110 1101 1011

7. 写出下列十六进制数的二进制表示。

21 a. A4693FBC　　b. B697C7A1　　c. 2B3D9461

1.4 布尔表达式

布尔代数（boolean algebra）定义了一组操作，其值为真（true）或假（false）。它的发明者是十九世纪中叶的数学家乔治·布尔（George Boole）。在数字计算机发明的早期，人们发现布尔代数可以用来描述数字电路的设计。同时，在计算机程序中，布尔表达式被用来表示逻辑操作。

一个布尔表达式（boolean expression）包括一个布尔运算符以及一个或多个操作数。每个布尔表达式都意味着一个为真或假的值。以下为运算符集合：

- 非（NOT）：标记为¬或～或'
- 与（AND）：标记为∧或·
- 或（OR）：标记为∨或 +

NOT 是一元运算符，其他运算符都是二元的。布尔表达式的操作数也可以是布尔表达式。示例如下：

表达式	说明	表达式	说明
¬ X	NOT X	¬ X ∨ Y	(NOT X) OR Y
X ∧ Y	X AND Y	¬ (X ∧ Y)	NOT (X AND Y)
X ∨ Y	X OR Y	X ∧ ¬ Y	X AND (NOT Y)

NOT　NOT 运算符将布尔值取反。用数学符号书写为¬X，其中，X 是一个变量（或表达式），其值为真（T）或假（F）。下表列出了对变量 X 进行 NOT 运算后所有可能的输出。左边为输入，右边（阴影部分）为输出：

X	¬ X
F	T
T	F

真值表中，0 表示假，1 表示真。

AND　布尔运算符 AND 需要两个操作数，用符号表示为 X ∧ Y。下表列出了对变量 X 和 Y 进行 AND 运算后，所有可能的输出（阴影部分）：

X	Y	X ∧ Y
F	F	F
F	T	F
T	F	F
T	T	T

当两个输入都是真时，输出才为真。这与 C++ 和 Java 的复合布尔表达式中的逻辑 AND 是相对应的。 22

汇编语言中 AND 运算符是按位操作的。如下例所示，X 中的每一位都与 Y 中的相应位进行 AND 运算：

```
X:          11111111
Y:          00011100
X ∧ Y:      00011100
```

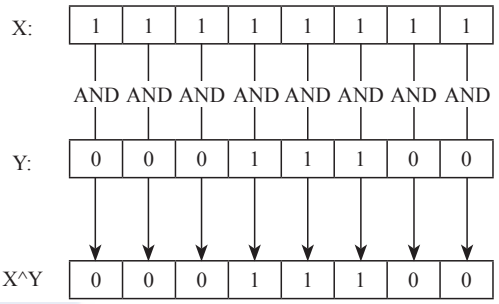


图 1-2 两个二进制整数按位 AND 运算

如图 1-2 所示，结果值 0001 1100 中的每一位表示的是 X 和 Y 相应位的 AND 运算结果。

OR 布尔运算符 OR 需要两个操作数，用符号表示为 X ∨ Y。下表列出了对变量 X 和 Y 进行 OR 运算后，所有可能的输出（阴影部分）：

X	Y	X ∨ Y
F	F	F
F	T	T
T	F	T
T	T	T

当两个输入都是假时，输出才为假。这个真值表与 C++ 和 Java 的复合布尔表达式中的逻辑 OR 对应。

OR 运算符也是按位操作。在下例中，X 的每一位与 Y 的对应位进行 OR 运算，结果为 1111 1100：

```
X:          11101100
Y:          00011100
X ∨ Y:      11111100
```

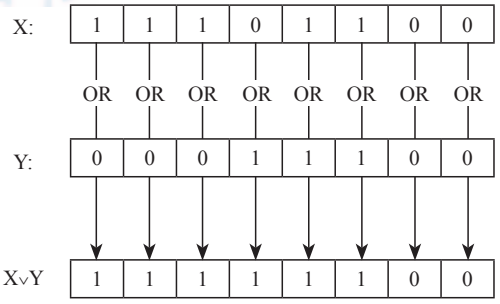


图 1-3 两个二进制整数按位 OR 运算

如图 1-3 所示，每一位都独立进行 OR 运算，生成结果中的对应位。 23

运算符优先级 运算符优先级原则（operator precedence rule）用于指示在多运算符表达式中，先执行哪个运算。在包含多运算符的布尔表达式中，优先级是非常重要的。如下表所示，NOT 运算符具有最高优先级，然后是 AND 和 OR 运算符。可以使用括号来强制指定表达式的求值顺序：

表达式	运算符顺序
¬ X ∨ Y	NOT, 然后 OR
¬ (X ∧ Y)	OR, 然后 NOT
X ∨ (X ∧ Z)	AND, 然后 OR

1.4.1 布尔函数真值表

布尔函数（Boolean function）接收布尔输入，生成布尔输出。所有布尔函数都可以构造一个真值表来展示全部可能的输入和输出。下面的这些真值表都表示包含两个输入变量 X 和 Y 的布尔函数。右侧的阴影部分是函数输出：

24 示例 1：¬ X∨Y

X	¬ X	Y	¬ X ∨ Y
F	T	F	T
F	T	T	T
T	F	F	F
T	F	T	T

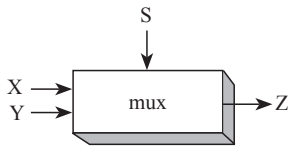
示例 2：X∧ ¬ Y

X	Y	¬ Y	X ∧ ¬ Y
F	F	T	F
F	T	F	F
T	F	T	T
T	T	F	F

示例 3：(Y∧S)∨(X∧ ¬ S)

X	Y	S	Y ∧ S	¬ S	X ∧ ¬ S	(Y ∧ S) ∨ (X ∧ ¬ S)
F	F	F	F	T	F	F
F	T	F	F	T	F	F
T	F	F	F	T	T	T
T	T	F	F	T	T	T
F	F	T	F	F	F	F
F	T	T	T	F	F	T
T	F	T	F	F	F	F
T	T	T	T	F	F	T

25 示例 3 的布尔函数描述了一个多路选择器（multiplexer），一种数字组件，利用一个选择位（S）在两个输出（X 和 Y）中选择一个。如果 S 为假，函数输出（Z）就和 X 相同；如果 S 为真，函数输出就和 Y 相同。下面是多路选择器的框图：



1.4.2 本节回顾

1. 描述布尔表达式¬ X ∨ Y。
2. 描述布尔表达式 (X ∧ Y)。
3. 布尔表达式 (T ∧ F)∨T 的值是什么？
4. 布尔表达式¬ (F ∨ T) 的值是什么？

5. 布尔表达式¬F ∨ ¬T 的值是什么？

1.5 本章小结

本书侧重于使用 MS-Windows 平台的 x86 处理器编程。内容涉及计算机体系结构、机器语言和底层编程的基本原则。读者将学到足够的汇编语言来测试自己已掌握的关于当前使用最广的微处理器系列的知识。

在阅读本书之前，读者应至少完成一门大学计算机编程课程或与之相当的课程。

汇编器是一种程序，用于把源程序从汇编语言转换为机器语言。与之配合的程序，称为链接器，把汇编器生成的单个文件组合成一个可执行程序。第三种程序，称为调试器，为程序员提供一种途径来追踪程序的执行过程，并检查内存的内容。

本书大部分内容都是关于 32 位和 64 位程序的，如果重点关注最后四章，则是 16 位程序。

通过本书将学习到如下概念：应用于 x86（和 Intel 64）处理器的基本计算机体系结构；基本布尔逻辑；x86 处理器如何管理内存；高级语言编译器如何将其语句转换为汇编语言和原生机器代码；高级语言如何在机器级实现算术表达式、循环和逻辑结构；有符号和无符号整数、实数和字符的数据表示。

汇编语言与机器语言是一对一的关系，即一条汇编指令对应于一条机器指令。汇编语言不具有可移植性，因为它是与具体处理器系统绑定的。 26

编程语言是一种工具，用于创建独立的应用程序或者部分应用程序。有些应用程序，如设备驱动和硬件接口程序，更适合使用汇编语言。而其他应用程序，如多平台商业和科学应用，用高级语言则更容易编写。

在展示计算机体系结构中的每一层如何表示为一个机器抽象时，虚拟机概念是一种有效的方式。每层可以用硬件或软件构成，其上编写的程序可以用其下一层进行翻译或解释。虚拟机概念可以与真实世界中的计算机层次相关，包括数字逻辑、指令集架构、汇编语言和高级语言。

二进制和十六进制数对在机器级工作的程序员来说，是非常重要的符号工具。因此，必须理解如何操作数制及其之间的转换，以及计算机怎样生成字符表示。

本章提出了 NOT、AND 和 OR 布尔运算符。一个布尔表达式包括一个布尔运算符以及一个或多个操作数。真值表是一种有效方法，用于展示布尔函数所有可能的输入和输出。

1.6 关键术语

ASCII
ASCII control characters（ASCII 控制字符）
ASCII digit string（ASCII 数字串）
assembler（汇编器）
assembly language（汇编语言）
binary digit string（二进制数字串）
binary integer（二进制整数）
bit（位 / 比特）
boolean algebra（布尔代数）
boolean expression（布尔表达式）
boolean function（布尔函数）
character set（字符集）
code interpretation（代码解释）
code point(Unicode)（代码点）
code translation（代码翻译）
debugger（调试器）
device driver（设备驱动程序）
digit string（数字串）

embedded systems application（嵌入式系统应用）
exabyte（艾字节）
gigabyte（吉字节）
hexadecimal digit string（十六进制数字串）
hexadecimal integer（十六进制整数）
unsigned binary integer（无符号二进制整数）
UTF-8
UTF-16
UTF-32
virtual machine(VM)（虚拟机）
high-level language（高级语言）
instruction set architecture(IAS)（指令集架构）
Java Native Interface(JNI)（Java 原生接口）
kilobyte（千字节）
27 language portability（语言的可移植性）
least significant bit(LSB)（最低有效位（LSB））
machine language（机器语言）
megabyte（兆字节）
microcode interpreter（微代码解释器）
microprogram（微程序）
Microsoft Macro Assembler(MASM)（Microsoft 宏汇编器）
most significant bit(MSB)（最高有效位）
multiplexer（多路选择器）
null-terminated string（以空字节结束的字符串）
octal digit string（八进制数字串）
one-to-many relationship（一对多关系）
operator precedence（运算符优先级）
petabyte（拍字节）
registers（寄存器）
signed binary integer（有符号二进制整数）
terabyte（太字节）
Unicode（统一码）
Unicode Transformation Format(UTF)（Unicode 转换格式）
Virtual machine concept（虚拟机概念）
Visual Studio
yottabyte（尧字节）
zettabyte（泽字节）

1.7 复习题和练习

1.7.1 简答题

1. 在一个 8 位二进制整数中，哪一位是最高有效位（MSB）？
2. 下列无符号二进制整数的十进制表示分别是什么？

a. 00110101
b. 10010110
c. 11001100

3. 下列每组二进制整数的和分别是多少？

a. 10101111 + 11011011
b. 10010111 + 11111111
c. 01110101 + 10101100

4. 计算二进制减法（0000 1101−0000 0111）。
5. 下列每种数据类型各包含多少位？

a. 字　　b. 双字　　c. 四字　　d. 八字

6. 表示下列无符号十进制整数时，需要的最少二进制位分别是多少？

a. 4095　　b. 65534　　c. 42319

7. 下列二进制数的十六进制表示分别是什么？

a. 0011 0101 1101 1010　b. 1100 1110 1010 0011　c. 1111 1110 1101 1011

8. 下列十六进制数的二进制表示分别是什么？

a. 0126F9D4　　b. 6ACDFA95　　c. F69BDC2A

9. 下列十六进制整数的无符号十进制表示分别是什么？

a. 3A b. 1BF c. 1001 28

10. 下列十六进制整数的无符号十进制表示分别是什么？

a. 62 b. 4B3 c. 29F

11. 下列有符号十进制整数的 16 位十六进制表示分别是什么？

a. −24 b. −331

12. 下列有符号十进制整数的 16 位十六进制表示分别是什么？

a. −21 b. −45

13. 将下列 16 位十六进制有符号整数转换为十进制数。

a. 6BF9 b. C123

14. 将下列 16 位十六进制有符号整数转换为十进制数。

a. 4CD2 b. 8230

15. 下列有符号二进制数的十进制表示分别是什么？

a. 10110101 b. 00101010 c. 11110000

16. 下列有符号二进制数的十进制表示分别是什么？

a. 10000000 b. 11001100 c. 10110111

17. 下列有符号十进制整数的 8 位二进制（补码）表示分别是什么？

a. −5 b. −42 c. −16

18. 下列有符号十进制整数的 8 位二进制（补码）表示分别是什么？

a. −72 b. −98 c. −26

19. 下列每组十六进制整数的和分别是多少？

a. 6B4 + 3FE b. A49 + 6BD 29

20. 下列每组十六进制整数的和分别是多少？

a. 7C4 + 3BE b. B69 + 7AD

21. ASCII 字符大写“B”的十六进制和十进制表示分别是什么？

22. ASCII 字符大写“G”的十六进制和十进制表示分别是什么？

23. 挑战：129 位无符号整数能表示的最大十进制值是多少？

24. 挑战：86 位有符号整数能表示的最大十进制值是多少？

25. 构造一个真值表，表示布尔函数￢（A^B）所有可能的输入和输出。

26. 构造一个真值表，表示布尔函数（￢A^ ￢B）所有可能的输入和输出。说明此表与 25 题真值表中最右列之间的关系。听说过摩根定理吗？

27. 如果一个布尔函数有 4 个输入，则其真值表需要多少行？

28. 4 输入的多路选择器需要多少个选择位？

1.7.2 算法基础

下列编程练习可以选择任何高级编程语言。不要调用已有的库函数来自动完成这些任务。（比如标准 C 库中的 sprinf 和 sscanf 函数。）

1. 编写一个函数来接收一个 16 位二进制整数字符串。函数返回值为该字符串的整数值。
2. 编写一个函数来接收一个 32 位十六进制整数字符串。函数返回值为该字符串的整数值。
3. 编写一个函数来接收一个整数。函数返回值必须是包含该整数二进制表示的字符串。
4. 编写一个函数来接收一个整数。函数返回值必须是包含该整数十六进制表示的字符串。
5. 编写一个函数实现两个以 b 为基数的数字串相加，其中 $2 \leqslant b \leqslant 10$。每个字符串可包含多达 1000 个数字。函数返回和数，其形式为基数相同的字符串。
6. 编写一个函数实现两个十六进制字符串相加，每个字符串含 1000 个数字。函数返回一个十六进制字符串来表示输入之和。

7. 编写一个函数实现一个长度为 1000 位的十六进制数字串与单个位的十六进制数字的乘法。函数返
30 回一个十六进制字符串来表示乘积。
8. 编写一个 Java 程序实现如下计算，然后用 *javap -c* 指令对代码进行反汇编。为每行代码添加注释，以说明该行代码的目的。

```
int Y;
int X = (Y + 4) * 3;
```

9. 设计无符号二进制整数减法。用（1000 1000-0000 0101=1000 0011）来检验该方法。再用至少两组其他的整数来检验该方法，每组都是从较大数中减去较小数。

本章尾注

1. Donald Knuth，MMIX，*A RISC Computer for the New Millennium*，麻省理工学院讲座记录，1999 年
31 12 月 30 日。