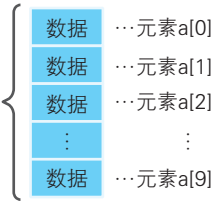
1.6　通过地址和索引实现数组

接下来我们看一下表 1-1 中出现的**基址寄存器**和**变址寄存器**。通过这两个寄存器，我们可以对主内存上特定的内存区域进行划分，从而实现类似于数组[①]的操作。

首先，我们用十六进制数[②]将计算机内存上 00000000～FFFFFFFF 的地址划分出来。那么，凡是该范围的内存区域，只要有一个 32 位的寄存器，即可查看全部的内存地址。但如果想要像数组那样分割特定的内存区域以达到连续查看的目的，使用两个寄存器会更方便些。例如，查看 10000000 地址～1000FFFF 地址时，如图 1-9 所示，可以将 10000000 存入基址寄存器，并使变址寄存器的值在 00000000～0000FFFF 变化。CPU 则会把基址寄存器＋变址寄存器的值解释为实际查看的内存地址。变址寄存器的值就相当于高级编程语言程序中数组的索引功能。

① 数组是指同样长度的数据在内存中进行连续排列的数据构造。用一个数组名来表示全体数据，通过索引来区分数组的各个数据（元素）。例如，一个 10 个元素的数组 a，其中的各个数据就用 a[0]～a[9] 来表示。[] 内的数字 0～9 就是索引。



② 二进制数的位数较多、不易理解时，通常使用十六进制数来代替二进制数。这是一种数到 16 就进位的计数方式。我们用 A～F 来分别表示 10～15，那么，二进制数的 4 位（0000 ～FFFF）就可以用十六进制数的 1 位（0～F）来表示。32 位的二进制数，就可以用 8 位的十六进制数来表示。

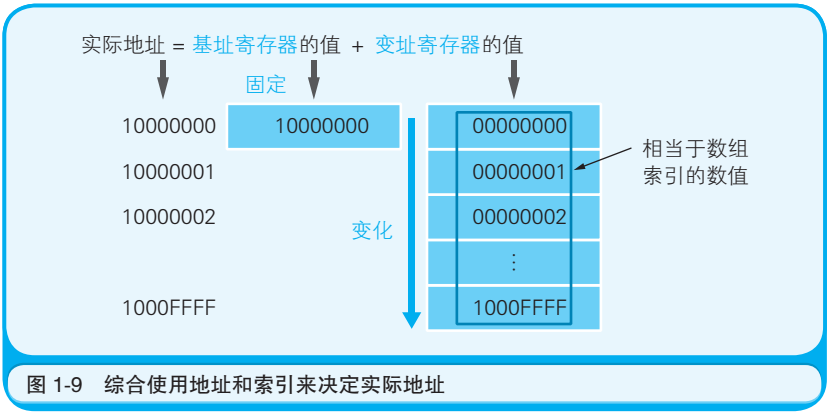


图 1-9 综合使用地址和索引来决定实际地址

1.7 CPU 的处理其实很简单

可能有些读者不知道机器语言和汇编语言的指令到底有多少种，因而对 CPU 的运行没什么概念。为了消除大家心中的这个疑团，接下来我们就来看一下机器语言到底有哪些种类。表 1-2 按照功能对 CPU 能执行的机器语言指令进行了大体分类。这里没有列出指令的具体名称（汇编语言的助记符）。看完表后你会惊奇地发现，原来 CPU 可以进行的处理非常少。虽然高级编程语言编写的程序看起来非常复杂，但 CPU 实际处理的事情就是这么简单。这样一来，大家是不是能够消除“计算机机制看起来很难”这个印象了呢？

表 1-2 机器语言指令的主要类型和功能

类 型	功 能
数据转送指令	寄存器和内存、内存和内存、寄存器和外围设备[①]之间的数据读写操作
运算指令	用累加寄存器执行算术运算、逻辑运算、比较运算和移位运算
跳转指令	实现条件分支、循环、强制跳转等
call/return 指令	函数的调用 / 返回调用前的地址

① 外围设备指的是连接到计算机的键盘、鼠标、显示器、设备装置、打印机等。

如果大家读完上文后有种恍然大悟的感觉，对程序的运行机制有了一个整体的印象，那么本书的目的也就达到了。只要对程序的运行机制有了一个整体印象，相信大家的编程能力和应用能力也会快速得到提高。现在再看之前写出来的程序，是不是感觉它们也变得活灵活现了呢？

本章在介绍标志寄存器时，提到过“位”这个专业术语。1 位代表二进制数的一个字节位，这一点对了解计算机的运算机制非常重要。在下一章中，我们将以位为基础，向大家介绍一下二进制数和浮点数这些数据形式，以及逻辑运算和位操作等相关知识。

第2章 数据是用二进制数表示的

热身问答

阅读正文前，让我们先回答下面的问题来热热身吧。

1. 32 位是几个字节？
2. 二进制数 01011100 转换成十进制数是多少?
3. 二进制数 00001111 左移两位后，会变成原数的几倍?
4. 补码形式表示的 8 位二进制数 11111111，用十进制数表示的话是多少?
5. 补码形式表示的 8 位二进制数 10101010，用 16 位的二进制数表示的话是多少?
6. 反转部分图形模式时，使用的是什么逻辑运算?

怎么样？是不是发现有一些问题无法简单地解释清楚呢？下面是笔者的答案和解析，供大家参考。

答案

1. 4 字节
2. 92
3. 4 倍
4. –1
5. 1111111110101010
6. XOR 运算

解析

1. 因为 8 位 = 1 字节，所以 32 位就是 32 ÷ 8 = 4 字节。
2. 将二进制数的各数位的值和位权相乘后再相加，即可转换成十进制数。
3. 二进制数左移 1 位后会变成原来的值的 2 倍。左移两位后，就是 2 倍的 2 倍，即 4 倍。
4. 所有位都是 1 的二进制数，用十进制数表示的话就是 –1。
5. 使用原数的最高位 1 来填充高位。
6. XOR 运算只反转与 1 相对应的位。NOT 运算是反转所有的位。

本章重点

要想对程序的运行机制形成一个大致印象，就要了解信息（数据）在计算机内部是以怎样的形式来表现的，又是以怎样的方法进行运算的。在 C 和 Java 等高级语言编写的程序中，数值、字符串和图像等信息在计算机内部都是以二进制数值的形式来表现的。也就是说，只要掌握了使用二进制数来表示信息的方法及其运算机制，也就自然能够了解程序的运行机制了。那么，为什么计算机处理的信息要用二进制数来表示呢？接下来我们就从其原因开始说起。

2.1 用二进制数表示计算机信息的原因

想必大家都知道计算机内部是由 IC[①] 这种电子部件构成的。第 1 章介绍的 CPU（微处理器）和内存也是 IC 的一种。IC 有几种不同的形状，有的像一条黑色蜈蚣，在其两侧有数个乃至数百个引脚；有的则像插花用的针盘，引脚在 IC 内部并排排列着。IC 的所有引脚，只有直流电压 0V 或 5V[②] 两个状态。也就是说，IC 的一个引脚，只能表示两个状态。

IC 的这个特性，决定了计算机的信息数据只能用二进制数来处理。由于 1 位（一个引脚）只能表示两个状态，所以二进制的计数方式就变成了 0、1、10、11、100…这种形式。虽然二进制数并不是专门为 IC 而设计的，但是和 IC 的特性非常吻合（图 2-1）。计算机处理信息的最

① IC 是集成电路（Integrated Circuit）的简称，有模拟 IC 和数字 IC 两种。本章介绍的是数字 IC。关于内存 IC，我们会在第 4 章详细说明。

② 大部分 IC 的电源电压都是 +5V。不过，为了控制电量的消耗，有的 IC 也会使用 +5V 以下的电压。如果 IC 使用的电源电压为 +5V，那么引脚状态就不只是 0V 和 +5V，还存在不接收电流信号的高阻抗（high impedance）状态。但在本书中，我们暂时不考虑高阻抗状态。

小单位——位，就相当于二进制中的一位。位的英文 bit 是二进制数位（binary digit）的缩写。

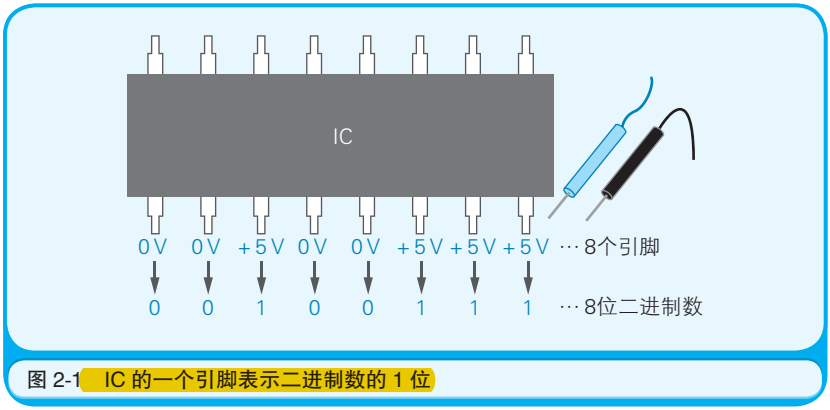


图 2-1　IC 的一个引脚表示二进制数的 1 位

二进制数的位数一般是 8 位、16 位、32 位……也就是 8 的倍数，这是因为计算机所处理的信息的基本单位是 8 位二进制数。8 位二进制数被称为一个**字节**[①]。字节是最基本的信息计量单位。位是最小单位，字节是基本单位。内存和磁盘都使用字节单位来存储和读写数据，使用位单位则无法读写数据。因此，字节是信息的基本单位。

用字节单位处理数据时，如果数字小于存储数据的字节数（＝二进制数的位数），那么高位上就用 0 填补。例如，100111 这个 6 位二进制数，用 8 位（=1 字节）表示时为 00100111，用 16 位（= 2 字节）表示时为 0000000000100111。奔腾等 32 位微处理器，具有 32 个引脚以用于信息的输入和输出。也就是说，奔腾一次可以处理 32 位（32 位 = 4 字节）的二进制数信息。

① 字节是由 bite（咬）一词而衍生出来的词语。8 位（8 bit）二进制数，就类似于“咬下的一口”，因此被视为信息的基本单位。

程序中，即使是用十进制数和文字等记述信息，在编译后也会转换成二进制数的值，所以，程序运行时计算机内部处理的也是用二进制数表示的信息（图 2-2）。

图 2-2　计算机内部所有信息都用二进制数处理

对于用二进制数表示的信息，计算机不会区分它是数值、文字，还是某种图片的模式等，而是根据编写程序的各位对计算机发出的指示来进行信息的处理（运算）。例如 00100111 这样的二进制数，既可以视为纯粹的数值作加法运算，也可以视为“‘’（单引号，single quotation）文字而显示在显示器上，或者视为■■□■■□□□这一图形模式印刷出来。具体进行何种处理，取决于程序的编写方式。

2.2　什么是二进制数

什么是二进制数？为了更清晰地说明二进制数的机制，首先让我们把 00100111 这个二进制数值转换成十进制数值来看一下。二进制数的值转换成十进制数的值，只需将二进制数的各数位的值和位权相乘，然后将相乘的结果相加即可（图 2-3）。

假使有人问你：“为什么使用这样的转换方法呢？你能解释一下吗？”你这么回答是不行的：“不知道原因，只是把方法背下来了。”我们了解了二进制数的机制后，再看二进制数转换成十进制数的方法，就没有死记硬背的必要了。下面我们会对照着十进制数来说明二进制数的机制，这部分是重点，请大家一定要掌握。

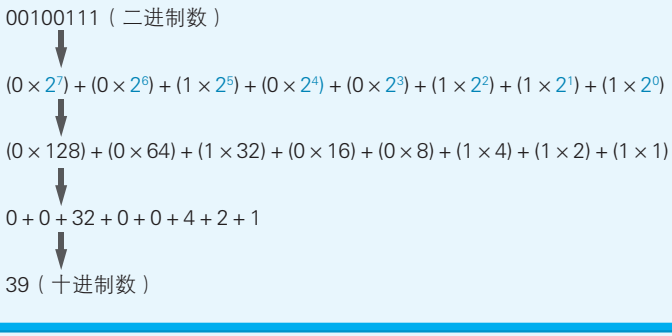


图 2-3　二进制数转换成十进制数的方法

首先，让我们从位权的含义说起。例如，十进制数 39 的各个数位的数值，并不只是简单的 3 和 9，这点大家应该都知道。3 表示的是 3×10 = 30，9 表示的是 9×1 = 9。这里和各个数位的数值相乘的 10 和 1，就是**位权**。数字的位数不同，位权也不同。第 1 位（最右边的一位）是 10 的 0 次幂[①]（= 1），第 2 位是 10 的 1 次幂（= 10），第 3 位是 10 的 2 次幂（= 100），依此类推。这部分相信大家都能够理解。那么，我们就继续讲一下二进制数。

位权的思考方式也同样适用于二进制数。即第 1 位是 2 的 0 次幂（= 1），第 2 位是 2 的 1 次幂（= 2），第 3 位是 2 的 2 次幂（= 4），……，第 8 位是 2 的 7 次幂（= 128）。“○○的 ×× 次幂”表示位权，其中，十进制数的情况下○○部分为 10，二进制数的情况下则为 2。这个称为**基数**[②]。十进制数是以 10 为基数的计数方法，二进制数则是以 2 为基数的计数方法。“○○的 ×× 次幂”中的 ××，在任何进制数中都是

① 所有数的 0 次幂都是 1。

② 数值的表现方法，进位计数制中各数位上可能有的数值的个数。十进制数的基数是 10，二进制数的基数是 2。

“数的位数 –1”。即第 1 位是 1 – 1 = 0 次幂，第 2 位是 2 – 1 = 1 次幂，第 3 位是 3 – 1 = 2 次幂。

接下来，让我们来解释一下各数位的数值和位权相乘后“相加”这个处理的原因。其实大家所说的数值，表示的就是构成数值的各数位的数值和位权相乘后再相加的结果。例如 39 这个十进制数，表示的就是 30 + 9，即各数位的数值和位权相乘后再相加的数值。

这种思考方式在二进制数中也是通用的。二进制数 00100111 用十进制数表示的话是 39，因为（0×128）+（0×64）+（1×32）+（0×16）+（0×8）+（1×4）+（1×2）+（1×1）= 39。大家明白了吗？

2.3 移位运算和乘除运算的关系

在了解了二进制数的机制后，接下来我们来看一下运算。和十进制数一样，四则运算同样也可以使用在二进制数中，只要注意逢 2 进位即可。下面，我们就来重点看一下二进制数所特有的运算。二进制数所特有的运算，也是计算机所特有的运算，因此可以说是了解程序运行原理的关键。

首先我们来介绍移位运算。**移位运算**指的是将二进制数值的各数位进行左右移位（shift = 移位）的运算。移位有左移（向高位方向）和右移（向低位方向）两种。在一次运算中，可以进行多个数位的移位操作。

代码清单 2-1 中列出的是把变量 *a* 中保存的十进制数值 39 左移两位后再将运算结果存储到变量 *b* 中的 C 语言程序。<< 这个运算符表示左移，右移时使用 >> 运算符。<< 运算符和 >> 运算符的左侧是被移位的值，右侧表示要移位的位数。那么，这个示例程序运行后，变量 *b*