和。这个和的反码形成了携带在报文段首部的因特网检验和。如在 3.3 节讨论的那样，接收方通过对接收的数据（包括检验和）的和取反码，并且检测其结果是否为全 1 比特来检测检验和。如果这些比特中有任何比特是 0，就可以指示出差错。RFC 1071 详细地讨论因特网检验和算法和它的实现。在 TCP 和 UDP 协议中，对所有字段（包括首部和数据字段）都计算因特网检验和。在其他协议中，例如 XTP［Strayer 1992］，对首部计算一个检验和，对整个分组计算另一个检验和。

检验和方法需要相对小的分组开销。例如，TCP 和 UDP 中的检验和只用了 16 比特。然而，与后面要讨论的常用于链路层的 CRC 相比，它们提供相对弱的差错保护。这时，一个很自然的问题是：为什么运输层使用检验和而链路层使用 CRC 呢？前面讲过运输层通常是在主机中作为用户操作系统的一部分用软件实现的。因为运输层差错检测用软件实现，采用简单而快速如检验和这样的差错检测方案是重要的。在另一方面，链路层的差错检测在适配器中用专用的硬件实现，它能够快速执行更复杂的 CRC 操作。Feldmeier［Feldmeier 1995］描述的快速软件实现技术不仅可用于加权检验和编码，而且可用于 CRC（见后面）和其他编码。

5.2.3 循环冗余检测

现今的计算机网络中广泛应用的差错检测技术基于**循环冗余检测**（Cyclic Redundancy Check，CRC）**编码**。CRC 编码也称为**多项式编码**（polynomial code），因为该编码能够将要发送的比特串看作为系数是 0 和 1 一个多项式，对比特串的操作被解释为多项式算术。

CRC 编码操作如下。考虑 d 比特的数据 D，发送结点要将它发送给接收结点。发送方和接收方首先必须协商一个 $r+1$ 比特模式，称为**生成多项式**（generator），我们将其表示为 G。我们将要求 G 的最高有效位的比特（最左边）是 1。CRC 编码的关键思想如图 5-6 所示。对于一个给定的数据段 D，发送方要选择 r 个附加比特 R，并将它们附加到 D 上，使得得到的 $d+r$ 比特模式（被解释为一个二进制数）用模 2 算术恰好能被 G 整除（即没有余数）。用 CRC 进行差错检测的过程因此很简单：接收方用 G 去除接收到的 $d+r$ 比特。如果余数为非零，接收方知道出现了差错；否则认为数据正确而被接收。

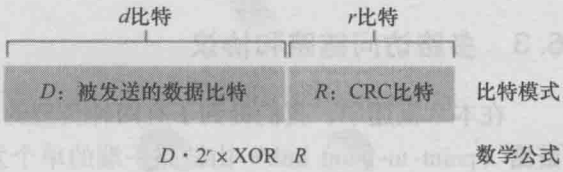

图 5-6 CRC

所有 CRC 计算采用模 2 算术来做，在加法中不进位，在减法中不借位。这意味着加法和减法是相同的，而且这两种操作等价于操作数的按位异或（XOR）。因此，举例来说：

1011 XOR 0101 = 1110

1001 XOR 1101 = 0100

类似的，我们还会有：

1011 − 0101 = 1110

1001 − 1101 = 0100

除了所需的加法或减法操作没有进位或借位外，乘法和除法与在二进制算术中是相同的。如在通常的二进制算术中那样，乘以 2^k 就是以一种比特模式左移 k 个位置。这样，给定 D 和 R，$D \cdot 2^r$ XOR R 产生如图 5-6 所示的 $d+r$ 比特模式。在下面的讨论中，我们将利用图 5-6 中这种 $d+r$ 比特模式的代数特性。