```
1   int sum(int x, int y) {
2       return x + y;
3   }
```

当我们在示例机器上编译时，生成如下字节表示的机器代码：

Linux 32　55 89 e5 8b 45 0c 03 45 08 c9 c3
Windows　55 89 e5 8b 45 0c 03 45 08 5d c3
Sun　81 c3 e0 08 90 02 00 09
Linux 64　55 48 89 e5 89 7d fc 89 75 f8 03 45 fc c9 c3

我们发现指令编码是不同的。不同的机器类型使用不同的且不兼容的指令和编码方式。即使是完全一样的进程，运行在不同的操作系统上也会有不同的编码规则，因此二进制代码是不兼容的。二进制代码很少能在不同机器和操作系统组合之间移植。

计算机系统的一个基本概念就是，从机器的角度来看，程序仅仅只是字节序列。机器没有关于原始源程序的任何信息，除了可能有些用来帮助调试的辅助表以外。在第 3 章学习机器级编程时，我们将更清楚地看到这一点。

2.1.6　布尔代数简介

二进制值是计算机编码、存储和操作信息的核心，所以围绕数值 0 和 1 的研究已经演化出了丰富的数学知识体系。这起源于 1850 年前后乔治·布尔(George Boole，1815—1864)的工作，因此也称为*布尔代数*(Boolean algebra)。布尔注意到通过将逻辑值 TRUE(真)和 FALSE(假)编码为二进制值 1 和 0，能够设计出一种代数，以研究逻辑推理的基本原则。

最简单的布尔代数是在二元集合{0，1}基础上的定义。图 2-7 定义了这种布尔代数中的几种运算。我们用来表示这些运算的符号与 C 语言位级运算使用的符号是相匹配的，这些将在后面讨论到。布尔运算~对应于逻辑运算 NOT，在命题逻辑中用符号 ¬ 表示。也就是说，当 P 不是真的时候，我们就说¬P 是真的，反之亦然。相应地，当 P 等于 0 时，~P 等于 1，反之亦然。布尔运算 & 对应于逻辑运算 AND，在命题逻辑中用符号 ∧ 表示。当 P 和 Q 都为真时，我们说 $P \wedge Q$ 为真。相应地，只有当 $p=1$ 且 $q=1$ 时，$p \& q$ 才等于 1。布尔运算 | 对应于逻辑运算 OR，在命题逻辑中用符号 ∨ 表示。当 P 或者 Q 为真时，我们说 $P \vee Q$ 成立。相应地，当 $p=1$ 或者 $q=1$ 时，$p|q$ 等于 1。布尔运算^对应于逻辑运算异或，在命题逻辑中用符号⊕表示。当 P 或者 Q 为真但不同时为真时，我们说 $P \oplus Q$ 成立。相应地，当 $p=1$ 且 $q=0$，或者 $p=0$ 且 $q=1$ 时，$p \hat{} q$ 等于 1。

~	
0	1
1	0

&	0	1
0	0	0
1	0	1

|	0	1
0	0	1
1	1	1

^	0	1
0	0	1
1	1	0

图 2-7　布尔代数的运算。二进制值 1 和 0 表示逻辑值 TRUE 或者 FALSE，而运算符~、&、|和^分别表示逻辑运算 NOT、AND、OR 和 EXCLUSIVE-OR

后来创立信息论领域的 Claude Shannon(1916—2001)首先建立了布尔代数和数字逻辑之间的联系。他在 1937 年的硕士论文中表明了布尔代数可以用来设计和分析机电继电器网络。尽管那时计算机技术已经取得了相当的发展，但是布尔代数仍然在数字系统的设计和分析中扮演着重要的角色。

我们可以将上述 4 个布尔运算扩展到*位向量*的运算，位向量就是固定长度为 w、由 0 和 1 组成的串。位向量的运算可以定义成参数的每个对应元素之间的运算。假设 a 和 b 分别表示位向量$[a_{w-1}, a_{w-2}, \cdots, a_0]$和$[b_{w-1}, b_{w-2}, \cdots, b_0]$。我们将 $a \& b$ 也定义为一个长度为 w 的位向量，其中第 i 个元素等于 $a_i \& b_i$，$0 \leqslant i < w$。可以用类似的方式将运算|、^