Bitwise Operators. C++ also provides other numeric operators, including operations that can be applied to integer data at the individual bit level: ~ (negation), & (bitwise and), | (bitwise or), ^ (bitwise exclusive or), << (bitshift left), and << (bitshift right).

In the following descriptions, b, b_1, and b_2 denote binary digits (0 or 1); x and y are integers.

Operator	Operation	Description
~	**bitwise negation**	~b is 0 if b is 1; ~b is 1 if b is 0
&	**bitwise and**	b_1 & b_2 is 1 if both b_1 and b_2 are 1; it is 0 otherwise
|	**bitwise or**	b_1 | b_2 is 1 if either b_1 or b_2 or both are 1; it is 0 otherwise
^	**bitwise exclusive or**	b_1 ^ b_2 is 1 if exactly one of b_1 or b_2 is 1; it is 0 otherwise
<<	**bitshift left**	x << y is the value obtained by shifting the bits in x y positions to the left
>>	**bitshift right**	x >> y is the value obtained by shifting the bits in x y positions to the right*

* *Note*: There is also an unsigned right shift operator >>> that fills the vacated bit positions at the left with 0s. >> is a signed right-shift operator that fills these positions with the sign bit of the integer being shifted.

To illustrate the behavior of these operators, the statements

```
int i = 6;                    // 0110
cout << (i | 5) << endl;      // 0110 OR 0101 = 0111
cout << (i & 5) << endl;      // 0110 AND 0101 = 0100
cout << (i ^ 5) << endl;      // 0110 XOR 0101 = 0011
cout << (i << 1) << endl;     // 0110 LS 1    = 1100
cout << (i >> 1) << endl;     // 0110 RS 1    = 0011
cout << (~i) << endl;        // NEG 0110     = 111...11001
```

produce the following output:[1]

```
7
4
3
12
3
-7
```

1. For the last output statement, see the *Part Of The Picture* section of Chapter 2 regarding two's complement representation of integers.

In practice, such operations are used in programs that must inspect memory or interact directly with a computer's hardware, such as low-level graphics methods or operating system methods.