### Comparisons

The relational comparison operators are used in numeric conditional expressions.

| Symbol | Meaning | Symbol | Meaning |
|---|---|---|---|
| == | equal | <= | less than or equal |
| != | not-equal | > | greater than |
| < | less than | >= | greater than or equal |

These operators return 1 for true and 0 for false.

The comparison operators work with numeric operands only. To do comparisons with string operands, use the **CmpStr** function.

Comparison operators are usually used in conditional structures but can also be used in arithmetic expressions. For example:

```
wave0 = wave0 * (wave0 < 0)
```

This clips all positive values in wave0 to zero.

See also **Example: Comparison Operators and Wave Synthesis** on page II-82.

## Operators in Functions

This section discusses operators in the context of user-defined functions. See **Operators** on page IV-6 for a discussion of operators used in the command line and macros as well as functions.

### Bitwise and Logical Operators

The bitwise and logical operators are also used in conditional expressions.

| Symbol | Meaning |
|---|---|
| ~ | Bitwise complement |
| & | Bitwise AND |
| \| | Bitwise OR |
| %^ | Bitwise exclusive OR |
| ! | Logical NOT |
| && | Logical AND |
| \|\| | Logical OR |
| << | Shift left (requires Igor7 or later) |
| >> | Shift right (requires Igor7 or later) |

The precedence of operators is shown in the table under **Operators** on page IV-6. In the absence of parentheses, an operator with higher precedence (higher in the table) is executed before an operator with lower precedence.

Because the precedence of the arithmetic operators is higher than the precedence of the comparison operators, you can write the following without parentheses:

```
if (a+b != c+d)
   Print "a+b is not equal to c+d"
endif
```

Because the precedence of the comparison operators is higher than the precedence of the logical OR operator, you can write the following without parentheses:

```
if (a==b || c==d)
   Print "Either a equals b or c equals d"
endif
```

The same is true of the logical AND operator, &&.

For operators with the same precedence, there is no guaranteed order of execution and you must use parentheses to be sure of what will be executed. For example:

```
if ((a&b) != (c&d))
   Print "a ANDED with b is not equal to c ANDED with d"
endif
```

See **Operators** on page IV-6 for more discussion of operators.

### Using Bitwise Operators

The bitwise operators are used to test, set, and clear bits. This makes sense only when you are dealing with integer operands.

| Bit Action | Operation |
|---|---|
| Test | AND operator (&) |
| Set | OR operator (\|) |
| Clear | Bitwise complement operator (~) followed by the bitwise AND operator (&) |
| Shift left | << operator (requires Igor7 or later) |
| Shift right | >> operator (requires Igor7 or later) |

This function illustrates various bit manipulation techniques.

```
Function DemoBitManipulation(vIn)
   Variable vIn

   vIn = trunc(vIn)              // Makes sense with integers only
   Printf "Original value: %d\r", vIn

   Variable vOut

   if ((vIn & 2^3) != 0)         // Test if bit 3 is set
      Print "Bit 3 is set"
   else
      Print "Bit 3 is cleared"
   endif

   vOut = vIn | (2^3)            // Set bit 3
   Printf "Set bit 3: %d\r", vOut

   vOut = vIn & ~(2^3)           // Clear bit 3
   Printf "Clear bit 3: %d\r", vOut

   vOut = vIn << 3               // Shift three bits left
   Printf "Shift three bits left: %d\r", vOut

   vOut = vIn >> 3               // Shift three bits right
   Printf "Shift three bits right: %d\r", vOut
End
```

For a simple demonstration, try this function passing 1 as the parameter.