Visual Basic 2008 Programming

Business Applications with a Design Perspective

Jeffrey J. Tsay

Table of Contents

Appendix A

Number Systems and Bit-wise Operations

This appendix discusses two related topics: number systems and bit-wise operations. A discussion of the number systems should help build a foundation for a deeper understanding of numeric and bit-wise operations. Most of the VB 2008 logical operators operate on a bit by bit basis. An understanding of how these operators work helps to eliminate unexpected errors resulting from code that performs operations that the programmer has not anticipated. It also helps the programmer takes advantages of these operators' special features.

A.1. Number Systems

Human beings use the decimal system for numbers. We count from 0 to 9 before adding another digit in presenting the number. Computers, on the other hand, operate on bits (binary digits) and bytes. A number is represented internally in the computer by setting various bits on or off. For example, the number zero is represented by setting off all bits used for that number; and one, the lowest bit, on. Because a bit has only two states (on and off), to go to the next number (2), the lowest bit is turned off, while the next lowest bit is turned on. Such a system is recognized as the binary number system. This coding system can be depicted as follows:

Binary system	Decimal system
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

Converting Between the Two Systems

If you examine the binary system closely, you may discover that each bit position represents a value of two raised to a certain power. For example, the lowest bit and the second lowest bit represent 2^0 and 2^1, respectively. This can be depicted as follows:

Binary number	1	1	1	1	1	1	1	1
Decimal value	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

From Binary to Decimal

To find the corresponding decimal value for any binary representation, multiply the bits by their corresponding positional value and sum the total. For example, a binary number, 10101, can have its decimal value computed as follows:

Binary value	1	0	1	0	1
multiply by	2^4	2^3	2^2	2^1	2^0
Results	16 +	0 +	4 +	0 +	1 = 21

From Decimal to Binary

Conversely, to find the binary value for a decimal number, divide the number by two and find the remainder, which is the bit value for the lowest position; then divide the integer quotient by two again to obtain the bit value for the next higher position. Repeat this process to find the bit value for each successive higher position until the resulting quotient is zero. The following table shows how the binary representation for 13 can be obtained.

Step	**4**	**3**	**2**	**1**
Value To Be Divided by 2	1	3	6	13
Resulting Integer Quotient	0	1	3	6
Remainder (Binary Equivalent)	1	1	0	1

As the table shows, in the first step (last column), 13 is divided by 2, resulting in a quotient 6 and a remainder 1. In step 2, 6 (the previous quotient) is divided by 2, resulting in a quotient 3 and a remainder 0. This process continues until step 4, when the resulting quotient is 0. The remainder row shows the bit representation for 13, that is, 1101.

The Hex Decimal Representation

Although the binary number system corresponds to the internal coding exactly, it is inconvenient to show a long number with this system. To simplify the representation, the hex decimal system has been introduced. Under this system, each digit has 16 possible values (0 to 15), as opposed to 2 in the binary system and 10 in the decimal system. The numbers are represented as shown in the following table:

Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D

14	1110	E
15	1111	F

As you can see from the table, the letters A, B, C, D, E, and F are used in the hex decimal system to represent 10, 11, 12, 13, 14, and 15 in the decimal system, respectively, so that each of the 16 numbers in the hex decimal system is one digit. The next number after F will be 10 because we have exhausted all symbols to represent a number in one hex digit.

Notice that the letters used in this context have no direct association with the letters used in any text. They are simply symbols used to represent the numbers in the hex system. Notice also a hex decimal digit can be conveniently used to represent four bits in the binary system. For example, a hex number F0 will indicate that the lower four bits are zeros (off), while the upper four bits are 1s (on).

Converting Between Hex and Decimal Numbers

Converting numbers between the two systems is similar to converting between decimal and binary numbers. Each position, p, in the hex system represents a value of 16 raised to the power of p - 1, beginning from the lowest. To convert a hex decimal number to a decimal value, multiply each hex digit by its positional value and sum the total. For example, the hex number F3D can be converted to a decimal value as follows:

```
Hex number               F      3      D
Decimal equivalent      15      3     13
Multiply by             16^2   16^1   16^0
Result                3840 +   48 +   13   = 3901
```

To convert a decimal number to a hex representation, divide the number by 16 to find the integer quotient and the remainder, which gives the value at the lowest position. Divide the quotient by 16 again to obtain the remainder for the value at the next lowest position. Continue this process until the integer quotient is zero. For example, the following table illustrates how a decimal number, 3901, can be converted to a hex number:

```
Step                          3      2      1
Number To Be Divided by 16   15    243   3901
Integer Quotient              0     15    243
Remainder                    15      3     13
Result: Remainder in Hex      F      3      D
```

As you can see in step 1 (last column) from the table, the number 3901 is divided by 16, giving an integer quotient of 243 and a remainder of 13, which can be represented in the hex system as D. In a similar fashion, the previous quotient, 243, is divided by 16 to obtain the quotient and remainder for the second round. The process continues until the resulting quotient is zero in step 3. The remainder in hex representation F3D gives the solution.

Representing Hex Numbers in VB

VB allows a direct representation of hex numbers. You can code a hex number by preceding it with an &H. For example, to represent a hex number F, you code &HF. If the number is fairly large, attach an & at the end to indicate it is a Long integer (64 bit). For example, code &HAB0CD0AC2& instead of &HAB0CD0AC2.

Why Discuss Number Systems?

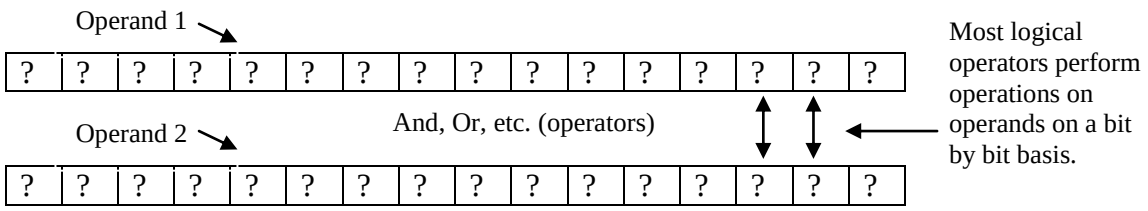
Usually, VB shows results with decimal values. After all, the decimal system is what we are most familiar with. So, why discuss the other number systems? Here are some reasons:

- In some cases, it is much easier to think in terms of bits. In some applications, many "states" are represented by "Flags," which are collectively represented by an integer with its specific bits set on or off. An understanding of the binary number system will make it easier for us to see the relationship between "setting a bit on" and an integer value. Incidentally, a Boolean value is stored as a two-byte integer. The value False has all its bits off (&H0000), whereas True has all its bits on (&HFFFF). You can easily see that False has a numeric equivalent of 0. Also, when an integer's highest bit is on, it is a negative number. Its complement (a value that adds that number to make it zero) is the number's negative value. Because adding 1 to &HFFFF will clear all its bits, making it zero, you can see why True has a numeric value of -1. (See Explore and Discover exercise C-1 at the end of this Appendix.)
- Some operators operate bit-wise. For example, the logical operators such as And and Or operate by bit. When you understand the binary number system, you will be able to understand the results of the operations much more easily. The bit-wise logical operators are discussed in the next section.
- Some numbers are more conveniently represented by hex numbers. Each hex digit can represent a certain state more readily.
- Some constants are traditionally represented by hex numbers. For example, the parameter constants used in API (an acronym for Application Program Interface) calls are all in hex representation. Getting acquainted with the hex number system will alleviate your fear of the mystery associated with this representation.

A.2 Bit-Wise Operation of Logical Operators

Most VB logical operators actually operate on data on a bit-by-bit basis and can be used to perform operations not only on Boolean data but also on integer (Long, Integer and Short) data (see Figure A-1). In this context, a bit is considered True when it is on (with a value 1) and False when it is off (with a value 0). (Notice that the context is *a bit not a data field*.)

Figure A-1
Bitwise operation of logical operators



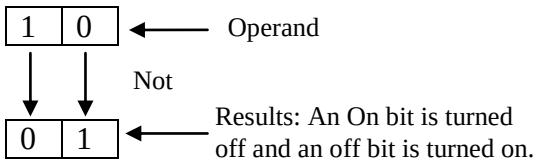
The Not, And, Or, Xor Operators

The following discussion focuses on four operators commonly used in the bit-wise operation: Not, And, Or, and Xor. Once you gain some understanding of these operations, you will be ready to learn how they can be used in various applications.

The Not Operator

Specifically, the *Not operator* will reverse the on or off state of each bit; that is, an on bit will become off, and an off bit will become on, as shown in Figure A-2.

Figure A-2
The Not operation

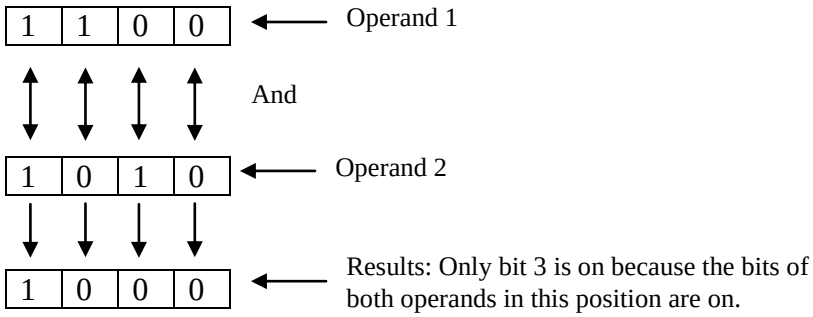


Incidentally, recall that a Boolean variable takes two bytes of storage. False is coded with all the bits turned off, so in the hex decimal representation, it is &H0000, which, if assigned to a Short variable, should have a value of 0. Applying a Not operation on the value will result in all bits being turned on to become &HFFFF, representing True, equivalent to a value -1.

The And Operator

The *And operator* will produce a result of 1 (on) if both its operands' corresponding bits are 1 (on); otherwise, the result will be 0 (off), as shown in Figure A-3.

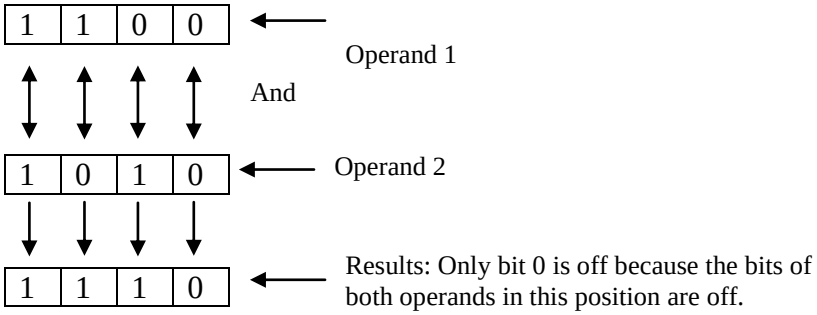
Figure A-3
The And operation



The Or Operator

The *Or operator* will produce a result of 1 if either of the corresponding bit operands is 1 (on); otherwise, the result will be 0 (off), as illustrated in Figure A-4.

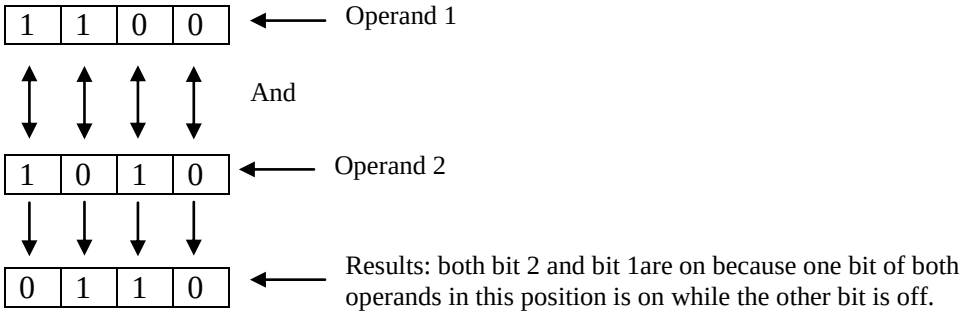
Figure A-4
The Or operation



The Xor Operator

The *Xor operator* will produce a result of 1 (on) if *one and only one* of the two bit operands is on; otherwise, the bit will be 0 (off), as shown in Figure A-5.

Figure A-5
The Xor operation



Various Uses of the Logical Operators

Now you know how the bit-wise operators work, but are they used and for what? The following discussion highlights some of these operators' uses.

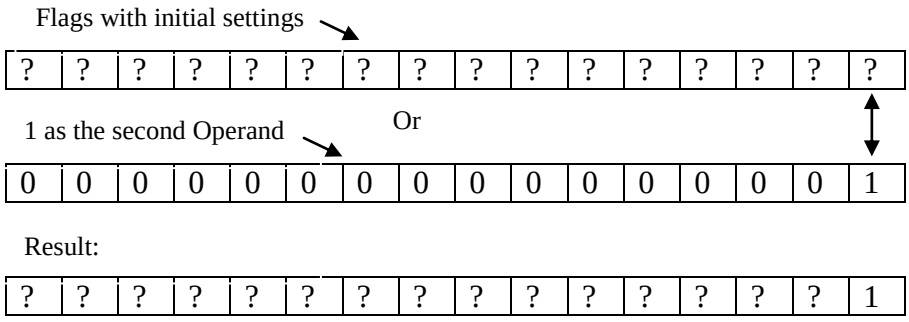
Setting a Flag

These operators can be used to set, test, or toggle Flags. A *Flag* is a bit of data representing the on/off state of something. A Flag is on when that particular bit is set to 1, and off when it is set to 0. Because a Short variable has 16 bits, it is often used to hold a group of Flags. As an illustration, suppose you have a Short variable, Flags that you want to use to represent several classifications of an account. You want to use the lowest bit to represent whether the account affects cash flow and the second lowest bit to represent whether the account is a control account. As you may recall from the preceding section (A-1), the lowest bit has a numeric value of 1 when it is on; and the second lowest bit has a numeric value of 2 when it is on. If you want to set the lowest bit on, you can code the following:

```
    Flags = Flags Or 1
```

Recall that the Or operator will set the resulting bit to 1 when either of the operands (bits) is 1. In this case, the second operand has only its lowest bit set on (to 1). When the two operands are Ored, the result will be exactly the same as that for Flags, except that its lowest bit will be on regardless of its previous state. This result can then be assigned to Flags to reflect the Flags' new state. This operation is illustrated in Figure A-6.

Figure A-6
Setting a Flag with the Or operator



Every bit in the result remains the same as the original Flags except that the lowest bit is now 1 (on) regardless of its previous setting.

By the same token, to set the second lowest bit on, you code:

```
    Flags = Flags Or 2
```

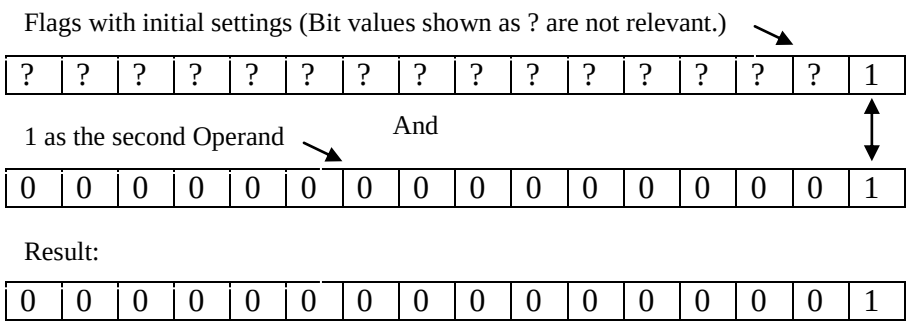
Testing a Flag

To test whether the lowest bit of Flags has been set on, you can use the And operator. For example, you can code the following:

```
    Test = Flags And 1
```

Again, recall that the result of an And operation will be 1 (on) for a bit only if both corresponding operands are 1 (on). The result will be 1 only if Flags' lowest bit is also on, as illustrated in Figure A-7.

Figure A-7
Testing a Flag with the And operator



The result will be either 0 or 1 depending on the setting of the lowest bit in Flags; all other bits will be off (0) because all higher order bits in the second operand are zeros.

By the same token, you can test whether the second lowest bit is on by coding the following:

```
    Test = Flags And 2
```

Testing for an Odd or Even Number

The previous illustration of testing whether the lowest bit of Flags is on has an additional interesting application. You may be aware that all odd integers have the lowest bit on, whereas even integers have it off. The same code can be used to test whether a number is odd or even. To experiment and verify, place a text box and a button on a new form. Name the text box **txtNumber** and the button **btnTest**. Also, set the Text property of the button to **Test**; then, enter the following code:

```
Private Sub btnTest_Click(ByVal Sender As Object, ByVal e As
  System.EventArgs) Handles btnTest.Click
    Dim Test As Integer
    Test = CInt(txtNumber.Text) And 1
    If Test = 0 Then
        'The lowest bit is off. This is an even number.
        MsgBox(txtNumber.Text & " is an even number.")
    Else
        'The lowest bit is on. This is an odd number.
        MsgBox(txtNumber.Text & " is an odd number.")
    End If
End Sub
```

After you run the project, enter a number and then click the Test button. The program should be able to tell you whether you have entered an odd or even number.

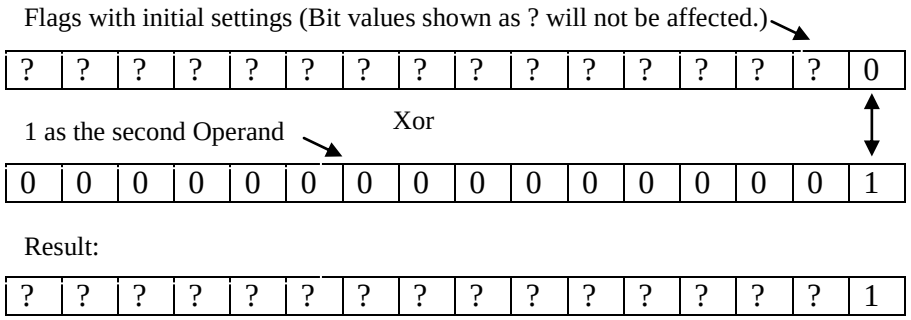
Toggling a Flag

The Xor operator also has a very interesting application. Often, it is used to toggle a Flag; that is, to turn a Flag from one state to the other. As explained earlier, this operator sets the result of a bit to 1 only when one of the two operands is 1 (on), so if you repetitively perform the Xor operation with a constant "on" bit to another operand, this latter operand will be off when it was originally on and on when it was originally off. Therefore, the following code should toggle the setting of Flags' lowest bit:

```
    Flags = Flags Xor 1 'Toggle the lowest bit
```

The Xor operation is illustrated in Figure A-8.

Figure A-8
Toggling a Flag with the Xor operator



The result will be either 0 or 1 depending on the setting of the lowest bit in Flags; repeating the operation will thus reverse the current result. All other bits will not be affected (0) because all higher order bits in the second operand are zeros.

Again, by the same token, you can toggle the second lowest bit by coding the following:

```
    Flags = Flags Xor 2 'Toggle the second lowest bit
```

Use of the Xor Operator for Encryption

The toggling capability of Xor makes it a popular operator to perform encryption operations. If some data is Xored with a "key", the data changes its original value; however, when the encrypted data is Xored again with the same key, the data is restored to its original value.

To test this method of encryption, set up a new project as follows:

- Place three labels and three text boxes on the form.
- Set the text properties of the three labels to Original Text, Encrypted Text, and Restored Text.
- Name the text boxes **txtOriginal**, **txtEncrypted**, and **txtDecrypted**.

Figure A-9 shows the visual interface in action.

Figure A-9
Visual interface for the encryption example

With the code below, whatever is entered into the first textbox (txtOriginal) will be encrypted and shown in the second text box (txtEncrypted), which is then decrypted and displayed in the third text box (txtDecrypted).

Assume you want to use the letter X as the password for encryption. Enter the following code in your new project:

```
Private Sub txtOriginal_KeyPress(ByVal sender As Object, ByVal e As
  System.Windows.Forms.KeyPressEventArgs) Handles txtOriginal.KeyPress
    Dim Encrypted As Integer
    Dim Decrypted As Integer
    Dim KeyAscii As Integer
    ' Encrypt the key being entered with "X"
    KeyAscii = AscW(e.KeyChar)
    Encrypted = KeyAscii Xor Keys.X
    ' Add the encrypted key to the encrypted text
    txtEncrypted.Text &= Chr(Encrypted)
    ' Restore the encryption by applying the same key
    Decrypted = Encrypted Xor Keys.X
    ' Add the restored key to the restored text
    txtDecrypted.Text &= Chr(Decrypted)
End Sub
```

The encryption/decryption routine is placed in the text box's KeyPress event, which is triggered when the user enters a key into the text box. This procedure has an event argument, KeyChar that gives the key pressed. The code first obtains the key code (Asc) value of this key and then encrypts this value by Xoring it with the ASCII value of X (Keys.X). The encrypted character (along with all previously encrypted characters) is displayed in the label named txtEncrypted. The encrypted ASCII value is then transformed with another Xor operation using the same key (Keys.X). This operation should restore the key to what was originally entered. The decrypted key (along with all previously restored keys) is displayed in the label named txtDecrypted.
Run the project and enter some text into the text box. As you enter a key, you should be able to see the encrypted result in txtEncrypted. You should also be able to verify that the decrypted text in txtDecrypted is the same as the text you originally entered.
Note that this routine ignores the possibility that the user may press a control key such as the Enter key or the Backspace key. You should be able to fine tune the program before finishing this appendix. The refinement is left to you to complete.

The following table summarizes the use of logical operators in bit-wise operations.

Operator	Operation	Example	Result	Remark
Not	Reverse the content	`Not &HFFFF`	&H0000	Not - 1 = 0; that is, Not True = False
And	1 only if both corresponding bits are 1	`&H0F And &HF033`	&H3	Can be used to test whether certain bits are on and to mask off certain bits
Or	1 if one of the corresponding bits is 1	`&H0F Or &HF03`	&H0F	Can be used to turn on certain bits
Xor	1 only if one (but not both) of the corresponding bits is one	`&H0F Xor &H03`	&H0C	Can be used to toggle certain bits and to encrypt data

Summary

- Human beings count from 0 to 9 before adding another digit in presenting the number. In contrast, computers operate on bits (binary digits) and bytes. The number system human beings use is recognized as the decimal system; and the number system the computer operates is recognized as the binary system.
- To simplify the representation of a binary number, the hex decimal system has been introduced. Under this system, each digit has 16 possible values (0 to 15) and can be used to represent four bits.
- To find the corresponding decimal value for any binary or hex decimal representation, multiply the bits/hex digits by their corresponding positional value and sum the total.

- To find the binary/hex representation for a decimal number, divide the number by two/sixteen and find the remainder, which is the bit/hex value for the lowest position. Then divide the integer quotient by two/sixteen again to obtain the bit/hex value for the next higher position. Repeat this process to find the bit/hex value for each successive higher position until the resulting quotient is zero.
- Most of the logical operators such as Not, And, Or, and Xor are bit-wise operators. They set the results of the operation on a bit-by-bit basis.
- The Not operator can be used to reverse the content or state of flags; the And operator can be used to test whether a particular bit is on or off as well as to test whether a number is odd or even; the Or operator can be used to turn on a bit; and the Xor operator can be used to toggle bits as well as to encrypt or decrypt data.

Explore and Discover

A-1. Some Hex Values of Interest. Enter the following code in the form's Click event procedure. Run the project, click any part of the form, and examine the results.

```
Console.WriteLine(&HFF)
Console.WriteLine(&HFF%)
Console.WriteLine(&HFFFFFFFF%)
Console.WriteLine(&HFFFFFFFF&)
```

What values do you get? The first two values should give you no surprises. But what about the last two values? Why are they different? The value &HFFFFFFFF% is of the Integer type. The four bytes holding the value have all the bits on. The highest bit of a number represents its sign. When it is on, it is a negative number. Its complement (whatever is required to add the value to zero) is the number's negative value. Adding 1 to &HFFFFFFFF% will clear all bits, making it zero; therefore, &HFFFFFFFF% is -1. The value &HFFFFFFFF& is a Long integer (because an & type suffix is attached). Its lower four bytes are all on, but its higher four bytes are all zero; that is, its internal coding is actually &H00000000FFFFFFFF&. It is a positive number, and is `4294967295`. (Perform the conversion as discussed in the text to verify.)

A-2. Numeric Value of True and False. Enter the following code in the form's Click event procedure. Run the project, click any part of the form, and examine the results. What numeric values do True and False have?

```
MsgBox("10 times True is " & 10 * True)
MsgBox("10 plus True is " & 10 + True)
MsgBox("10 times False is " & 10 * False)
MsgBox("10 plus False is " & 10 + False)
```

True is internally coded as a two-byte integer with all its bits on: &HFFFF, which is -1. False is internally coded as a two-byte integer with all its bits off: &H0000, so it is numerically zero.

A-3. Use of Logical Operators. Relational operators work on both numeric and string data. Can logical operators do the same? Try the following statements. (*Hint:* To test them, place these lines in the Form Click event procedure with a Dim A as Integer statement at the beginning; then insert a line of code to display the result between the lines below.)

```
A = "34" And "32"
A = "34" Or 32
A = "XY" Or "AB"
```

Why do the first two run, but not the last? (*Answer:* Logical operators work on numeric data only. Data type conversions are performed automatically by VB in the first two cases, making the code executable. It is impossible to convert the strings to numbers in the last case, so the operation fails.)

A-4. The Value Property of the Radio Button. Place two radio buttons on a new form and then enter the following code:

```
Dim A As Integer
Private Sub RadioButton1_Click(ByVal sender As Object, ByVal e As
  System.EventArgs) Handles RadioButton1.Click
    A = RadioButton1.Checked
    MsgBox("A = " & A)
End Sub

Private Sub RadioButton2_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles RadioButton2.Click
    A = RadioButton1.Checked
    MsgBox("A = " & A)
End Sub
```

Run the project, and alternately click the two radio buttons. What value of A do you see each time? The numeric value for True is -1 (&HFFFF) and for False is 0 (&H0000).

A-5. The Value Property of the Check Box. Place a check box on a new form. Enter the following code:

```
Dim A As Integer
Private Sub CheckBox1_Click(ByVal sender As Object, ByVal e As
  System.EventArgs) Handles CheckBox1.Click
    A = CheckBox1.Checked
    MsgBox("A = " & A)
End Sub
```

Run the project and then click the check box. What result do you see each time? When the box is checked? And when it is unchecked? Are they the same as the radio buttons?

A-6. Assigning a Value to the Boolean Variable. Place a text box and a button on a new form. Name the text box **txtNumber** and the button **btnCheck**. Set the button's Text property to **Check**. Enter the following code:

```
Private Sub btnCheck_Click(ByVal Sender As Object, ByVal e As
  System.EventArgs) Handles btnCheck.Click
    Dim TheNumber As Single
    Dim BoolTest As Boolean
    TheNumber = Val(txtNumber.Text)
    BoolTest = TheNumber
    MsgBox("The resulting Boolean value is " & BoolTest)
    If BoolTest = TheNumber Then
        MsgBox("BoolTest and TheNumber are equal.")
    Else
        MsgBox("BoolTest and TheNumber are Not equal.")
```

```
    End If
End Sub
```

Run the program. Enter any number (try at least these numbers: 1000, -30, 0.005, and 0) and then click the button. What does the computer display? What conclusion can you draw from this experiment? What do you also learn? (*Answer:* Any nonzero value will be converted to True for a Boolean variable. Only a value of 0 is interpreted as False. Beware of the potential problem with comparing a Boolean variable with a variable of another data type after assigning the same value to both.)

A-7. Playing with the Truth. Enter the following code in the code window of a new project:

```
Private Sub Form1_Click(ByVal sender As Object, ByVal e As
  System.EventArgs) Handles MyBase.Click
    Dim B As Boolean
    Dim L As Integer
    Dim S As String
    B = 3 = 3
    L = 3 = 3
    S = 3 = 3
    MsgBox("B = " & B & ", L = " & L & ", S = " & S)
    MsgBox("Len(B) = " & Len(B) & ", Len(L) = " & Len(L) _
        & ", Len(S) = " & Len(S))
    S = L
    MsgBox("S = " & S & ", Len(S) = " & Len(S))
End Sub
```

Run the project and then click on the form. Does everything turn out to be as expected? Why does S have a length of 4 at one time and then 2 at another time? Should True have a length of two or four?

A-8. Using the And Operator on the Same Value. Draw a text box and a button on a new form. Name the text box **txtNumber** and clear its text. Name the button **btnAnd** and set its Text property to **And**. Place the following code in the code window:

```
Private Sub btnAnd_Click(ByVal Sender As Object, ByVal e As
  System.EventArgs) Handles btnAnd.Click
    Dim N As Integer
    Dim R As Integer
    R = Val(txtNumber.Text)
    N = R And R
    MsgBox("The result is " & N)
End Sub
```

Run the project. Enter an integer number and then click the And button. Repeat the same experiment with different numbers. What results do you see? (The corresponding bits of the two operands are the same. The result of the And operation should be the same as the operand.)

A-9. Using the Or Operator on the Same Value. (continued from exercise C-8) Add a button to the form in C-8. Name it **btnOr** and set its Text property to **Or**. Place the following code in the code window:

```
Private Sub btnOr_Click(ByVal Sender As Object, ByVal e As
System.EventArgs) Handles btnOr.Click
    Dim N As Integer
    Dim R As Integer
    R = Val(txtNumber.Text)
    N = R Or R
    MsgBox("The result is " & N)
End Sub
```

Run the project. Enter an integer number and then click the Or button. Repeat the same experiment with different numbers. What results do you see? (The corresponding bits of the two operands are the same. The result of the Or operation should be the same as the operand.)

A-10. Use of the Xor Operator on the Same Value. (continued from exercise C-9) Add another button to the form in C-9. Name the button **btnXor** and set its Text property to **Xor**. Place the following code in the code window:

```
Private Sub btnXor_Click(ByVal Sender As Object, ByVal e As
  System.EventArgs) Handles btnXor.Click
    Dim N As Integer
    Dim R As Integer
    R = Val(txtNumber.Text)
    N = R Xor R
    MsgBox("The result is " & N)
End Sub
```

Run the project. Enter an integer number and then click the Xor button. Repeat the same experiment with different numbers. What result do you see? (The corresponding bits of the two operands are the same. The result of the Xor operation should give zero.) Can you generalize the results of performing the And, Or, and Xor operations on the same number?

Exercises

A-11. Converting Numbers Between Systems.

A. Convert the following numbers to binary values:
   - 41
   - 255
   - 4095

B. Convert the following numbers to hex values:
   - 1000
   - 255
   - 32768 (*Hint:* Check the result in C below for verification.)

C. What are the decimal values of the following hex numbers? (*Hint:* To verify your computation, write simple code to obtain the results; for example, `Console.WriteLine(&HFF)` will print 255.)
   - &H1000
   - &HFFFE
   - &H10000
   - &H10000&

A-12. Testing Divisibility by 4. In this appendix, you have learned how you can test whether an integer is an odd or even number. The same idea can also be used to test whether a number is divisible by 4 by using the And operator. A number is divisible by 4 when its lowest two bits are zero.

Place a text box and a button on a new form. Name the text box txtNumber and the button btnTest and then provide the code to test whether the number the user has entered into the text box is divisible by 4. (*Hint:* Use the And operator. A number with all bits off but the lowest two bits on has a value of 3.)

Use the MsgBox to display the result. Also, use the Mod operator to test the same (place the code in the Form Click event). Compare the results obtained from both approaches.

A-13. Encrypting a Number. A company keeps its employees' salary data in a file in encrypted form. The data-entry screen for the salary data has a masked text box for the employee number such as the Social Security number (SSN) and a text box for the salary amount. When the user clicks the Save button, the salary amount is encrypted and then saved. Both the SSN and the salary are internally treated as Integers. The Xor operation is performed on the salary using the SSN as the key. The result is displayed in a text box. Your program then displays a message, "Salary saved." (*Note:* Just display the encrypted result and the message. Ignore the Save operation.)

A-14. Playing with the Flags. You have three check boxes in a general ledger account entry form with the following texts:

- Is this a control account? (Question 1)
- Is this a contra account? (Question 2)
- Does this account affect cash flow? (Question 3)

You have decided to handle these questions internally with a Flags variable; that is, values in these check boxes will be collectively stored in only one Short variable named Flags. Bits 0, 1, and 2 will be used to keep track of questions 1, 2, and 3, respectively. When a check box is checked, its corresponding bit will be set on; when it is unchecked, its corresponding bit will be set off.

Provide the code to handle these Flags and display the results (the value of Flags) when a change occurs. (*Hint:* Place the code in the check boxes' CheckedChanged event handlers. Use the And operator to turn a Flag off and the Or operator to turn a Flag on. Use a text box [name it **txtResult**] to display the value of Flags at the end of each Click event. When all three Flags are on, Flags should have a value of 7.)

A-15. Rotating Background Colors. Write a procedure that rotates the background color of a label, lblSign, in blue, red, green, yellow, and back to blue each time the user clicks a button named btnChange, with the text "Change." (*Hint:* Use a Static integer variable to keep track of the count. Use the And operator [although you could use Mod] to generate the number sequence 0, 1, 2, 3, 0, 1 … and so on, which can then be used in a Select Case block.)

A-16. Rotating Background Colors Automatically. Modify the preceding project (A-15) so that the colors are rotated automatically every half of a second once your program starts.

A-17. Alternative Solution to the Exploration Problem. Refer to the oil exploration investment decision example presented in Chapter 5, "Decision." The problem has an alternative solution. Write a line of code using relational and logical operators to come up with a correct value for the variable, Invest. (*Question:* Which coding alternative executes more efficiently?)