Data Representation

Introduction

All data in the computer memory can only consist of patterns of 1's and 0's. Memory cells can only hold 8 1's and 0's – called 8 bits, or 1 byte. However, we know that the computer can manipulate numbers (both integers and real number), characters or strings (sequences of character) and Boolean values. It can also store images and music. All of these things must be stored as patterns of 1's and 0's.

Here we'll examine how the 1's and 0's can be interpreted indifferent ways such that they represent such data. Besides a brief mention of character representations we'll focus on representing integers and real numbers.

Representing Characters

Traditionally characters have been represented using the ASCII code. Often textbooks will list tables of characters and their corresponding binary codes; you shouldn't have trouble finding such a table if you look for it. The ASCII code is a 7 bit code; although stored with 8 bits the leading bit is not used (it's constant).

1. How many characters can be represented using the ASCII code?

More recently the Unicode representation has been used to represent characters. It uses considerably more bits thereby allowing a great many more characters to be represented. Thus characters from most of the world's languages can be represented.

Representing Integers

The Unsigned Integer code

Of course integers may be positive or negative whole numbers, but as the name suggests this code is able to represent only positive integers. Consider 8 bits … the possible codes are

0000 0000
0000 0001
0000 0010
0000 0011
0000 0100
0000 0101
…
1111 1111

2. How many such codes are there in this list?

The first in this list (are. 0000 0000) is taken to represent the integer 0, the next represents 1, etc.

3. What integer does the last in the list (i.e. 11111111) represent?

This code is the Byte data type in VB 2005. Write a program that let you explore the limits of arithmetic using the Byte data type.

The 2s Complement code

Naturally, in order to perform arithmetic we must be able to represent negative integers as will as positive integers. The 2s complement code permits this by using half of the set of possible binary codes to represent positive integers and the other half to represent negative integers. Suppose we are using 16 bits. The possible codes are

Binary code	Decimal	Binary code	Decimal
0000 0000 0000 0000	0	1000 0000 0000 0000	? (Q.5)
0000 0000 0000 0001	1	1000 0000 0000 0001	
0000 0000 0000 0010	2	1000 0000 0000 0010	
0000 0000 0000 0011	3	1000 0000 0000 0011	
0000 0000 0000 0100	4	1000 0000 0000 0100	
…		…	
		1111 1111 1111 1110	-2
0111 1111 1111 1111	? (Q.4)	1111 1111 1111 1111	-1

The codes on the left represent positive integers from 0 to the maximum value in this 16 bit code, and those on the right represent negative values from -1 to the largest negative value.

4. What integer does the last code in the list on the left (i.e. 0111 1111 1111 1111) represent?

5. What integer does the first code in the list on the right (i.e. 1000 0000 0000 0000) represent? (This is the largest negative value.)

The Short data type in VB is a 16-bit 2’s complement code. The Integer and Long data types are also 2’s complement codes, but use 32 and 64 bits respectively.

Representing Real numbers

Of course we can’t do much arithmetic without real numbers either. So while the integer representations are important data types the Floating Point codes are perhaps even more widely used. We’ll discuss a made-up floating point code that nevertheless illustrates the main concepts of the real codes used in a computer.

The basic idea is to use some of the bits in a code to represent the fractional part of a number and other bits to represent an exponent. Thus a real number is represented as
$+/- 2^E \times 1.m$
where E is the exponent and in (the mantissa) is the fraternal part. Both E and m are binary codes. In addition the sign (+/-) is represented by one bit.

Thus suppose we have 16 bits – they might be partitioned as one bit for the sign (s), 6 bits for the exponent and 9 bits for the mantissa …

0 000000 000000000
s E m

The Exponent

Consider E first, while s = 0 and m= 000000000, i.e. $+2^E \times 1.0$
The 6 bits of E will need to represent both positive and negative integers so that we can represent values such as $+2^5 \times 1.0$ and $+2^{-4} \times 1.0$

6. Since there are 6 bits what is the largest positive value for E?
7. What is the largest negative value for E?

Thus successive codes for E will represent real numbers such as
…
$+2^{-4} \times 1.0$ = 1/16
$+2^{-3} \times 1.0$ = 1/8
$+2^{-2} \times 1.0$ = 1/4
$+2^{-1} \times 1.0$ = 1/2
$+2^0 \times 1.0$ = 1
$+2^1 \times 1.0$ = 2
$+2^2 \times 1.0$ = 4
$+2^3 \times 1.0$ = 8
$+2^4 \times 1.0$ = 16
…

8. Based on your answers to Q.6 and Q.7 what is the largest value in this list and what is the value closest to zero?

You might be asking yourself how zero itself is represented! You can do a bit of research to find that out if you wish but as you'll see in the next section too we're interested only in the broad concepts rather than all the details. (Mind you representing zero is very important – hardly a detail!)

The Mantissa

Suppose E and s are constant, i.e., choosing E = 1 and s = 0 (i.e. +) we have $+2^1 \times 1.m$. Now consider what the different codes for m allow us to represent.

The 9 bits of m are interpreted as a binary fraction. The details of this are another topic we'll avoid. Suffice it to say the values for 1.m range from 1.0 to close to 2 (i.e. 1.999…). Thus we have the two limits $+2^1 \times 1.0$ and $+2^1 \times 1.999…$

What would be the next value after $+2^1 \times 1.999…$? We can't change m anymore, but we are close to the next exponent, i.e. $+2^2$, and that is the next sequential value.

Thus between the two values of E, $+2^1 \times 1.0$ and $+2^2 \times 1.0$, we have a set of values generated from all of the codes made possible by the 9 bits of m.

9. How many values are there in our made-up Floating Point code between $+2^1 \times 1.0$ and $+2^2 \times 1.0$?
10. How many values are there between $+2^8 \times 1.0$ and $+2^9 \times 1.0$?
11. How many values are there between $+2^{-7} \times 1.0$ and $+2^{-6} \times 1.0$?

The floating point codes can never represent all the real numbers. Between any two values that *can* be represented there are an infinite number of other real numbers that *cannot* be represented. Arithmetic that results in a value that is between two values that can be represented is rounded to the nearest one. Thus, calculations with floating point data types almost always involve rounding errors.

When the binary codes are converted to decimal values there are a limited number of significant figures (or precision) due to this fundamental fact, i.e. you can represent 1.342792 and 1.342794 but nothing in between (these values are hypothetical, made up to illustrate the point). In the Single data type of VB there are about 7 significant figures and in the Double data type about 15. This is due to the number of bits used for the mantissa (of course it's a lot more than the 9 of our made-up code).