CHAPTER 6
Number Bases

We usually record numbers using the decimal system of notation. For instance 1432, which we call one thousand four hundred and thirty two, stands for the more awkward expression

$$1*10^3 + 4*10^2 + 3*10 + 2$$

We can rewrite this in an even more awkward way:

$$1*10^3 + 4*10^2 + 3*10^1 + 2*10^0$$

since 101 = 10, and 100 = 1. In other words, the number 1432 is interpreted as a sum of multiples of powers of 10. The integers 1, 4, 3, and 2 are called the digits of the number with 1 being the thousands digit, 4 the hundreds digit, 3 the tens digit and 2 the units digit. Technically we refer to this representation of the number as its *decimal representation* and say that the number is expressed to the *base* of 10. The word decimal comes from the Latin *decem*, ten.

The decimal system has a base of 10. But bases other than 10 can be used. Using different bases to interpret numbers is both interesting and useful. For example, numbers represented in base 2 have proved to be extremely important in computers and computer related activities.

Any integer greater than 1 can be used as a base, and any number can be expressed in any base. Furthermore, it is easy for your computer to convert a number expressed in one base into another base.

Let N stand for any positive integer, and let B be an integer greater than 1. To express the number N to the base B we need to write N in the following way:

$$N = X_m*B^m + X_{m-1}*B^{m-1} + . . . + X_1*B + X_0$$

where each of the numbers X_0, X_1 . . . X_m are integers between 0 and B-1. (See what happens if you substitute 10 for B.) The digits X_0, X_1, etc., are called the *coefficients* of the number N to base B.

Small values of B, the base, give long representations of the numbers. But they have the advantage of requiring fewer choices for the coefficients. The extreme case occurs when B = 2. The resulting system is called the *binary* number system (from the Latin, *binarius*, two). When a number is written in the binary system only the integers 0 and 1 can appear as coefficients. For example

$$\begin{aligned} 86 &= 64 + 16 + 4 + 2 \\ &= 1*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 1*2 + 0 \end{aligned}$$

Thus the number 86 expressed in binary form is 1010110. Binary numbers are used by computers because they are represented as strings of zeros and ones. The reason is that 0 and 1 can be easily expressed in a computer by a switch being either off or on.

For bases larger than 10 we need some extra symbols. The obvious symbols to use are the letters of the alphabet A, B, C, etc. A common base that is used is 16. A number expressed in the base 16 is called *hexadecimal*. The advantage of this base is that it requires few coefficients to express a number and yet hexadecimal numbers are easily converted to binary numbers.

To convert a number from base 10 to base B is quite straightforward. Suppose we want to convert the number N from base 10 to base B. First subtract all multiples of B from N.

```
M = INT(N/B) : R = N - B * M
```

alternatively

```
M = N DIV B : R = N MOD B
```

Record the remainder and call it R_0. Now repeat the process with M by setting N = M. Call the new remainder R_1. Continue in this way until the value of M reaches 0. Suppose that RS is the last remainder we find, then the original number N to base B is

$$R_S \ldots R_2R_1R_0$$

Let's go through a specific example. Suppose we want to convert the number 29 to base 3. The calculation proceeds as follows.

Step 1.

$$\begin{aligned} N &= 29 \\ M &= \text{INT}(29/3) \\ &= 9 \\ R_0 &= 29 - 3*9 \\ &= 2 \end{aligned}$$

Step 2.
$N = M$
$= 9$
$M = INT(9/3)$
$= 3$
$R_1 = 9 - 3*3$
$= 0$
Step 3.
$N = M$
$= 3$
$M = INT(3/3)$
$= 1$
$R_2 = 3 - 3*1$
$= 0$
Step 4.
$N = M$
$= 1$
$M = INT(1/3)$
$= 0$
$R_3 = 1 - 3*0$
$= 1$

The process stops after four steps when M reaches 0. The value of 29 to base 3 is 1002.

The next program converts integers from one base to another base. For example you could convert numbers in base 10 to base 16. For bases greater than 10 the letters A, B, C, etc., are used to represent the numbers 10, 11, 12, etc.

Listing 6.1
LIST

```
   10 REM Base converter
   20 MODE 1:COLOUR 3:PRINT ' TAB(13);"B
ase converter"':@%=10
   30 PRINT "This program converts integ
ers fom one base to another.":COLOUR 1
   40 REPEAT
   50  INPUT '"Enter base to convert fro
m: ";A
   60  IF A<11 THEN AA=47+A ELSE AA=54+A
   70  IF A<2 OR A>35 OR A<>INT(A) THEN
COLOUR 3:PRINT '"Silly - try again.":COL
```

```
OUR 1
   80 UNTIL A>1 AND A<36 AND A=INT(A)
   90 REM Next part checks that N$ is in
 the correct form
  100 REPEAT
  110  TEST=-1
  120  INPUT '"Number to be converted: "
;N$
  130  IF N$="" THEN COLOUR 3:PRINT '"No
t a number - try again please.":COLOUR 1
:TEST=0
  140  IF TEST THEN PROCTest
  150 UNTIL TEST
  160 REM Store N$ in array A(I)
  170 DIM A(L)
  180 FOR I=1 TO L
  190  N=ASC(MID$(N$,I,1))
  200  IF N<58 THEN A(I)=N-48
  210  IF N>64 THEN A(I)=N-55
  220 NEXT
  230  REM Convert number from base A to
 base 10
  240 N=VAL(N$)
  250 IF A<>10 THEN N=0:FOR I=1 TO L:N=A
(I)+N*A:NEXT
  260 PRINT '"Decimal form of number: ";
:COLOUR 2:PRINT ;N:COLOUR 1
  270 REPEAT
  280  INPUT '"Enter base to convert to:
 ";B
  290  IF B<2 OR B>35 OR INT(B)<>B THEN
COLOUR 3:PRINT '"Silly - try again.":COL
OUR 1
  300 UNTIL B>1 AND B<36 AND INT(B)=B
  310 REM Convert N to base B
  320 N$=""
  330 REPEAT
  340  M=INT(N/B):R=N-B*M:N=M
  350  IF R<10 THEN N$=CHR$(48+R)+N$
  360  IF R>9 THEN N$=CHR$(55+R)+N$
  370 UNTIL N=0
  380 PRINT '"Number to base ";B;" is ";
  390 IF LEN(N$)>40-POS THEN PRINT '
  400 COLOUR 2:PRINT ;N$
  410 COLOUR 3:PRINT CHR$(7) '' TAB(10);
"Another go? Y or N ";
  420 REPEAT:G$=GET$:UNTIL G$="Y" OR G$=
"N"
  430 IF G$="Y" THEN RUN
  440 CLS:PRINT '"Bye for now.":END
  450 DEF PROCTest
  460 L=LEN(N$)
  470 FOR I=1 TO L
  480  N=ASC(MID$(N$,I,1))
  490  IF N<48 OR (N>57 AND N<65) OR N>A
A THEN COLOUR 3:PRINT '"Not a positive i
nteger!":COLOUR 1:TEST=0:I=L
  500 NEXT
  510 ENDPROC
```

RUN

```
             Base converter

This program converts integers fom one b
ase to another.

Enter base to convert from: ?2

Number to be converted: ?111115

Decimal form of number: 15

Enter base to convert to: ?16

Number to base 16 is F
```

```
          Another go? Y or N
```

Acorn numbers

A number between 0 and 255 can be represented as a binary number using at most 8 coefficients. For example

$$255 = 1*2^7 + 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2 + 1$$
$$128 = 1*2^7 + 0*2^6 + 0*2^5 + 0*2^4 + 0*2^3 + 0*2^2 + 0*2 + 0$$

On a computer these coefficients, or eight bits, are called a byte. The BBC and Electron micros store integers using four bytes, which we'll refer to as 0-byte, 1-byte, 2-byte and 3-byte. The 3-byte represents multiples of 256*256*256, the 2-byte represents multiples of 256*256, the 1-byte represents multiples of 256 and the 0-byte represents the remainder. For example the number 999 would be stored with 1-byte equal to 3 and 0-byte equal to 231, since 999 = 3*256 + 231. Storing numbers using bytes is the same as storing numbers to the base of 256.

You can see how your computer stores the four bytes of an integer by PEEKing. (To PEEK at a memory location X, simply type PRINT ? X.) Remember that integer variables are specified by the per cent (%) sign after a variable name. Type the following, pressing return at the end of each line:

```
NEW : CLEAR
CK% = 999
```

Now PEEK at memory locations LOMEM + 5 to LOMEM + 8, and the four bytes used to store the integer CK% will be revealed. (*Note:* Do not replace CK% by any of the 26 integer variables A% to Z%, the resident integer variables, because they are stored elsewhere.) The next program performs all of the necessary PEEKs automatically.

Listing 6.2
LIST

```
   10 REM Byte dislayer
   20 MODE 1:COLOUR 3:PRINT ' TAB(13);"B
yte displayer"':@%=10
   30 PRINT "This program displays how y
our micro    stores integers using 4 byt
es."':COLOUR 1
```

```
   40 CLEAR:I%=LOMEM + 5:CK%=0
   50 PRINT '"Enter the integer you want
 displayed."':COLOUR 2
   60  INPUT '"Integer: ";CK%
   70 PRINT:COLOUR 1
   80 FOR K%=I% TO I%+3:PRINT K%-I%; " B
yte: ";? K%:NEXT
   90 COLOUR 3:PRINT CHR$(7) '' TAB(10);
"Another go? Y or N ";
  100 REPEAT:G$=GET$:UNTIL G$="Y" OR G$=
"N"
  110 IF G$="Y" THEN RUN
  120 CLS:PRINT '"Bye for now.":END
```

RUN 1

```
             Byte displayer

This program displays how your micro
stores integers using 4 bytes.

Enter the integer you want displayed.


Integer: ?129

         0 Byte: 129
         1 Byte: 0
         2 Byte: 0
         3 Byte: 0


          Another go? Y or N
```

RUN 2

```
            Byte displayer

This program displays how your micro
```

```
stores integers using 4 bytes.


Enter the integer you want displayed.


Integer: ?257

          0 Byte: 1
          1 Byte: 1
          2 Byte: 0
          3 Byte: 0


           Another go? Y or N
```

The largest integer that the BBC and Electron can store is 2147483647 which equals 127*256*256*256 + 255*256*256 + 255*256 + 255. Numbers with 3-byte equal to 128 or larger are negative numbers. Indeed negative numbers are stored by first looking at the ABSolute value of the number, calculating the 0, 1, 2 and 3-bytes, and then subtracting the values of the 0, 1 and 2-bytes from 256 and the value of the 3-byte from 255. Thus -1 would have the 0, 1, 2 and 3-bytes all equal to 255. As another example, -2147483647 has the 0-byte equal to 1, the 1-byte equal to 0, the 2-byte equal to 0 and the 3-byte equal to 128.

The following two program lines show you how to calculate a number from the four bytes. Let B0, B1, B2 and B3 be the four bytes.

```
NUMBER = B3*256*256*256 + B2*256*256 + B1*256 + B0
IF B3 >= 128 THEN NUMBER =
-((255 - B3)*256*256*256 + (256-B2)*256*256 + (256-B1)*256
+ (256-B0))
```

Try POKEing numbers in locations LOMEM + 5 to LOMEM + 8, then get your computer to PRINT CK% and compare the answers. (To POKE the number Y into memory location X simply type ? X = Y.) The next program performs these operations for you.

Listing 6.3

LIST

```
   10 REM Byte POKE
   20 MODE 1:COLOUR 3:PRINT ' TAB(15);"B
yte POKE"':@%=10
   30 PRINT "This program allows you to
POKE some    numbers into a location use
d to store   an integer."'
   40 PRINT "The resulting integer is th
en displayed directly and by calculating
."
   50 CLEAR:I%=LOMEM + 5:CK%=0:X%=0:Y%=1
   60 PRINT '"Enter the 4 bytes of the i
nteger."':COLOUR 2
   70 FOR J%=0 TO 3
   80  K%=I%+J%:IF J%>0 THEN Y%=Y%*256
   90  REPEAT
  100   PRINT ' J%;:INPUT " Byte: ";N%
  110   IF N%<0 OR N%>255 THEN COLOUR 1:
PRINT '"Between 0 and 255 please!":COLOU
R 2
  120  UNTIL N%>=0 AND N%<256
  130  ? K% = N%
  140  IF J%<3 OR (J%=3 AND N%<128) THEN
 X%=X%+N%*Y% ELSE X%=X%-256-255*256-255*
256*256-(255-N%)*Y%
  150 NEXT
  160 PRINT:COLOUR 1
  170 PRINT '"    Number stored: ";CK%
  180 PRINT '"Number calculated: ";X%
  190 COLOUR 3:PRINT CHR$(7) '' TAB(10);
"Another go? Y or N ";
  200 REPEAT:G$=GET$:UNTIL G$="Y" OR G$=
"N"
  210 IF G$="Y" THEN RUN
  220 CLS:PRINT '"Bye for now.":END
```

RUN

```
               Byte POKE
```

```
This program allows you to POKE some
numbers into a location used to store
an integer.

The resulting integer is then displayed
directly and by calculating.

Enter the 4 bytes of the integer.

         0 Byte: ?1

         1 Byte: ?1

         2 Byte: ?0

         3 Byte: ?0

    Number stored: 257

Number calculated: 257

          Another go? Y or N
```

Small numbers

The program in the section before the previous one works on positive whole numbers. Any number, integral or non-integral, has a representation in any base. For example the decimal number 0.25 expressed in binary takes the form 0.01, while the decimal number 0.125 is 0.001 in binary. To see the first let's see what we mean by the decimal number 0.25. This number represents two-tenths and five-hundredths, that is

$$\begin{aligned} 0.25 &= 2/10 + 5/100 \\ &= 2*10^{-1} + 5*10^{-2} \end{aligned}$$

To express it to the base B means writing it in the form

$$Y_1*B^{-1} + Y_2*B^{-2} + \dots$$

where as usual $B^{-1} = 1/B$, $B^{-2} = 1/B^2$, etc.

The decimal numbers 0.25 and 0.125 may be written in the following way:

$$\begin{aligned}0.25 &= 1/4\\ &= 0*2^{-1} + 1*2^{-2}\\ 0.125 &= 1/8\\ &= 0*2^{-1} + 0*2^{-2} + 1*2^{-3}\end{aligned}$$

which explains the binary form of these numbers.

As another example, look at the number 0.6 expressed in terms of (negative) powers of 2.

$$\begin{aligned}0.6 = {} & 1*2^{-1} + 0*2^{-2} + 0*2^{-3} + 1*2^{-4}\\ & + 1*2^{-5} + 0*2^{-6} + 0*2^{-7} + 1*2^{-8}\\ & + 1*2^{-9} + 0*2^{-10} + 0*2^{-11} + 1*2^{-12}\\ & + 1*2^{-13} + 0*2^{-14} + 0*2^{-15} + 1*2^{-16}\\ & + \ldots\end{aligned}$$

In fact we need an infinite number of terms to express 0.6 accurately in binary form. This takes the form

0.10011001100110011001100110011001 . . .

Your computer only stores 32 of these digits starting with the first non-zero one: in addition it rounds up if the thirty-third significant digit is non-zero. Thus your computer stores 0.6 as

0. 10011001100110011001100110011010

in binary form.

The next program displays the binary form, as stored by your computer, of a decimal number between 0 and 1.

Listing 6.4

LIST

```
   10 REM Decimal to binary
   20 MODE 1:COLOUR 3:PRINT ' TAB(11);"D
ecimal to binary"':@%=10
   30 PRINT "This program prints the bin
ary form of anumber between 0 and 1."':C
OLOUR 2
   40 REPEAT
   50  INPUT '"Enter number: ";N
   60  IF N<=0 OR N>=1THEN COLOUR 1:PRIN
T '"Between 0 and 1 please!":COLOUR 2
   70 UNTIL N>0 AND N<1
   80 N$="0."
   90 FOR I=1 TO 32
  100  N=N*2:N$=N$+STR$(INT(N)):N=N-INT(
N)
  110 NEXT
  120 PRINT:COLOUR 1
  130 PRINT '"The binary form of your nu
mber is:"'
  140 PRINT N$
  150 COLOUR 3:PRINT CHR$(7) '' TAB(10);
"Another go? Y or N ";
  160 REPEAT:G$=GET$:UNTIL G$="Y" OR G$=
"N"
  170 IF G$="Y" THEN RUN
  180 CLS:PRINT '"Bye for now.":END
```

RUN

```
          Decimal to binary

This program prints the binary form of a
number between 0 and 1.

Enter number: ?0.6

The binary form of your number is:

0.10011001100110011001100110011010

          Another go? Y or N
```

Floating points

Integers in the BBC and Electron micros are stored using 4 bytes. But numbers are stored using 5 bytes, even if the number itself is an integer. Unless you declare your number to be an integer by using the per cent sign it will be stored as a real number using 5 bytes. A number can be expressed in binary form in the following way:

$1.X_1X_2X_3 \ldots X_m * 2^N$

where X_1, $X_2 \ldots X_m$ are either 0 or 1, and N is an integer (positive, negative or zero). The integer N is called the *binary exponent* of the number, the other part is called the *binary mantissa*. For instance decimal 10 is binary 1010 which may be rewritten as

$1.01 * 2^3$

so that 10 has binary exponent 3 and binary mantissa 1.01. As another example look at decimal 0.375 which is binary 0.011 and so may be written as

$1.1 * 2^{-2}$

and so decimal 0.375 has binary exponent -2 and binary mantissa 1.1.

We have said that your computer uses 5 bytes to store its numbers. The first byte is the binary exponent plus 129. The remaining four bytes give the binary mantissa and the sign of the number. Since the first term in the binary mantissa is always 1 we do not need to store it - we simply store all the digits to the right of the decimal place in the binary mantissa. The first bit of byte 2 stores the sign of the number, the remaining 31 bits in the last four bytes store the binary mantissa (ignoring the leading 1).

For example, decimal 10 would be stored as follows: The first byte is 129 plus the binary exponent 3, which totals 132. The first bit of the second byte would be 0 since the number is positive. The remaining 31 bits would be

0100000000000000000000000000000

since the binary mantissa of 10 is 1.01 and we ignore the leading 1. Thus the 32 bits for the last four bytes would be

00100000000000000000000000000000

which, when broken into four groups of 8, give

00100000 00000000 00000000 00000000

which in turn are 32, 0, 0, 0. Thus the 5 bytes used to store the decimal 10 would be 132, 32, 0, 0, 0.

We can reverse the process and find the number that the computer is holding 5 bytes. Suppose that a number N is stored with the five bytes P, Q, R, S, T. The following program lines calculate N from P, Q, R, S and T.

```
X = 1 : IF Q >= 128 THEN Q = Q - 128 : X = -1
N = X*2P-129*(1 + Q*2-7 + R*2-15 + S*2-23 + T*2-31)
```

To see your computer in action use the next program.

Listing 6.5

LIST

```
   10 REM Byte displayer for numbers
   20 MODE 1:COLOUR 3:PRINT ' TAB(13);"B
yte displayer"':@%=10
   30 PRINT "This program displays how y
our micro    stores numbers using 5 byte
s."':COLOUR 1
```

```
   40 CLEAR:I%=LOMEM + 4:CK=0
   50 PRINT '"Enter the number (or expre
ssion) you    want displayed."':COLOUR 2
   60 INPUT '"Number: ";N$
   70 CK=EVAL(N$)
   80 COLOUR 1:PRINT '"The number: ";CK
   90 PRINT '"Byte form:"
  100 FOR K%=I% TO I%+4:PRINT K%-I%; " B
yte :";? K%:NEXT
  110 COLOUR 3:PRINT CHR$(7) ''"        A
nother go? Y or N ";
  120 REPEAT:G$=GET$:UNTIL G$="Y" OR G$=
"N"
  130 IF G$="Y" THEN RUN
  140 CLS:PRINT '"Bye for now.":END
```

RUN

```
             Byte displayer

This program displays how your micro
stores numbers using 5 bytes.

Enter the number (or expression) you
want displayed.

Number: ?1 + 2^-24

The number: 1.00000006

Byte form:
         0 Byte :129
         1 Byte :0
         2 Byte :0
         3 Byte :0
         4 Byte :128

       Another go? Y or N
```