Exploring Floating point numbers.

Two sketches are presented to illustrate the steps involved in entering and reading floating point numbers. The first step is to distinguish between the two forms that our numbers can take:

Strings: These can can be entered at the PC keyboard and sent to the screen
Binary numbers: These can be used to perform arithmetic

Acquiring the numerical strings is not really part of this investigation and therefore subroutines that do this are placed in a "Resources" directory, from where they can be opened using "programmers notepad".

In the first sketch signed decimal numbers containing a decimal point are considered. The integer and decimal parts are converted separately and then combined together with a twos exponent.

The second sketch is an extension of the first. It includes the first as a subroutine and converts numbers containing a tens exponent.

Real numbers

Binary floating point numbers have a very definite format and therefore it might be better to start by considering non-integer numbers referred to here a real numbers.

An example of a real number in binary format is 111.111

The integer part 111 is decimal seven (4 + 2 + 1)

The non integer part 0.111 is 0.5 + 0.25 + 0.125 = 0.875

Therefore 111.111 equals 7.875

Real binary numbers can be shifted right and left in exactly the same way the integer numbers can.
Shifting one place to the left multiplies the number by 2.
Shifting one place to the right divides it by 2.

We can therefore multiply 111.111 by 8 to get 111111 = 63
and say that 111.111 = 111111 divided by 8 = 63/8 = 7.875

In a similar way we can say 111.111 = 0.111111 x 8
equals (0.5 + 0.25 + 0.125 + 0.0625 + 0.03125 + 0.015625) x 8 = 0.9843754 x 8 = 7.875

Note: In the number 7.875, the 7 (binary 111) is referred to as its LHS.
.875 (binary .111) is referred to as its RHS.
The RHS digits/bits are also referred to as the decimal/binary places.

Converting a fraction to decimal or binary A slight digression is required before going any further. Consider the fraction A/B where B > A.
To convert this to decimal we use a loop containing two lines of integer arithmetic.

The first line calculates (10 x A) /B which gives an answer between 0 and 9
The second line calculates the remainder of this sum which is (10 x A)%B
The first line is then repeated, however this time A is replaced by the remainder just calculated.

For binary conversion we replace the 10 by 2 and the division always gives a result of 0 or 1

Decimal to binary conversion Consider the number 7.875. We can process the LHS using familiar integer conversion. For the RHS we use the fraction to binary routine.

Consider 0.875: It is equal to the fraction 875/1000 where 1000 is known as the denominator
The first bit on the RHS of the binary point is 2 x 875/1000 = 1750/1000 = 1
This sum generates a remainder of 750
(750 x 2) /1000 is 1 remainder 500 and therefore the second bit is also 1
(500 x 2)/1000 is 1 remainder zero. Therefore the third and final bit is also 1

And we get 0.875 = 0.111 in binary.

Setting the number of significant bits The number 7.875 has 3 decimal digits following the decimal point. It also has 3 bits following the binary point.
However the number of these bits will vary greatly and it may be necessary to limit them.

Consider the number 7.877. A calculator will shown that, to a very good approximation this is equal to

111.1110 0000 1000 0011 0001 0010 0110 1110.

The fraction to binary routine generates one bit for each division.
We therefore set the number of divisions to the number of significant bits for which there is space.

The floating point format allows 24 bits for the real number.
The number 7.875 requires 3 bits for the integer part.
When calling the fraction to binary routine we specify that 21 divisions are required and 0.875 is converted to binary 0.111000000000000000000

Assembling a floating point number (FPN) The bits in our sample number 111.111 will be shifted to 1.11111 and the exponent will be set to 2. The single 1 on the LHS of the binary point will not be saved since it is present for all numbers. Lagging zeros will be added to the RHS of the binary point until 23 bits are occupied as shown below:

11111000000000000000000

The steps required to achieve this are as follows:

Convert the integer part of the real number to binary and save it as a long integer
Determine the length (n) of this integer
Shift the integer by (24 - n) bits so that its MSB occupies bit 23
Convert the RHS of the real number to a long integer
Perform the decimal to binary conversion repeating the division process (24 – n) times
The LHS and RHS of the binary number can now be combined using the OR function

Note: As can be seem from the sample number the exponent is given by n – 1.

Bits 23 to 30 are used for the exponent. 127 is added to it before it is saved. An exponent of 2 is therefore saved as 129. Similarly other exponents are saved as follows:

-126	as	1
0	as	127
+127	as	254

This arrangement allows zero and 255 (all ones) to be reserved for underflow and overflow.

Note:
Bit 23 (the single 1) on the LHS of the binary point is over written by the exponent.
Bit 31 is used for the sign (zero for a positive number and one for a negative).

Binary to decimal conversion Consider the FPN 11111 with exponent of 2
The first step towards converting it to a decimal sting is to reinstate the missing one.
This gives us the number 1.11111 x 4 which is equal to 111.111.

Consider the binary number 0.111
We can shift this to the LHS of the binary point without changing its value provided we remember to divide the result by 8.
Therefore 0.111 = binary 111 divided by 8 which equals 7/8

This is converted to decimal using a fraction to decimal routine as shown below:

We start with 7/8

70/8 is 8 remainder 6	the char (8 + '0') is sent to the PC
60/8 is 7 remainder 4	the char (7 + '0') is sent to the PC
40/8 is 5 remainder zero	the char (5 + '0') is sent to the PC

Scientific Numbers A scientific number as defined here is just a real number with an exponent for example: 45.875 E 3.

The string 45.875E3 entered at the keyboard is first converted to 4.5875 E4
4.5875 is converted to a floating point number which is then multiplied by 10 four times. This approach produces the maximum accuracy. Consider the number 0.00008. Intuitively we might expect 8E-5 to be converted more accurately. This is in fact the case.

Rounding There are two opportunities for rounding. In the first case we compute an additional binary place. We then use integer arithmetic to increment the result and shift it right by one place to loose the additional bit.
In the second we compute one more decimal place than is required, save the result to a string and convert this to binary. This time we add 5 to the result before converting it back to a string.

The Sketches These follow the steps outlined above hopefully including steps glossed over in this brief summary and of course also cover the reverse process (FPN to string).