Gerald's Column
by Gerald Fitton

The theme of my current series is: "How have mathematicians influenced the development of the computer?" Later, but not this month, I shall proceed to the question: "Is there another way in which the computer might be developed?"

Numbers in a Computer

Natural numbers are numbers with which you count. They are numbers such as 0, 1, 2, 3, etc. In some books I have seen them referred to as the set of positive whole numbers.

It is my contention that Natural numbers are the only numbers which a computer uses and that it resorts to tricks in order to store other sorts of number. I explained how negative integers are stored as large positive integers and how modular or 'clock' arithmetic is used on these numbers.

A computer can not store all the Real numbers, indeed it can not store any of the irrational numbers such as the square root of two nor can it store transcendental numbers such as π, the ratio (circumference) ÷ (diameter) of a circle, or ***e,*** the base of the natural logarithms. Indeed it can not store many of the Rational numbers (fractions) accurately but only those for which the denominator is a power of two.

Most Real numbers are not stored accurately but are stored as approximations. They are not stored in the fixed point format which we are used to but in a format called Floating Point Format.

Fixed Point Format

This is the format which we all learned at school. For example the number 1234 is recognised as one thousand two hundred and thirty four. It is understood that the column in which the various digits, 1, 2, 3, and 4 appear represent thousands, hundreds, tens and units. This notation is called 'Place notation' and it depends on the existence of a character for zero; by this I mean that the number 1000 is understood to be one thousand because of the three trailing zeros. The Romans had no symbol for zero and wrote the number one thousand as the capital M. Place notation does not exist in the Roman number format.

The method of doing sums (say addition or multiplication) which we all learned at school depends on understanding the nature of place notation and its meaning.

Floating Point Format

Unless you did a course of Computer Studies at school or college it is unlikely that you have ever used floating point notation. Perhaps the nearest you will have come to it is what is known as Scientific notation. In Scientific notation a number such as 12 345 (twelve thousand three hundred and forty five) is written as 1.2345 e 4. The "e 4" at the end means 'move the decimal point four places to the right' to obtain the number in the more usual format.

In Floating point format the number 12 345 is stored as two separate numbers which are related to the two parts of the Scientific format version of the number. The digits 12345 (without any decimal point) are called the ‘Mantissa’ and the number 4 (of the e 4) is called the exponent.

As a second example, a (standard format) number such as 0.000 123 456 is written as 1.234 56 e−4 in Scientific notation and, in Floating point notation, it is written as ‘Exponent −4, Mantissa 123 456’. It is common but not invariable practice to state the value of the exponent before that of the mantissa.

In a computer numbers are stored either as Integers or in Floating point format. As I have said, negative integers are stored as large positive integers; these large positive numbers are Natural numbers and not Negative Integers. In floating point format numbers are stored as a pair of Natural numbers, one representing the exponent and the other the mantissa; the sign is stored separately.

The Floating Point Emulator (FPE)

In my early version of the Programmers’ Reference Manual the chapter headed “Floating Point Emulator” starts with “The Acorn RISC machine has a general co-processor interface. The first co-processor envisaged is one which performs calculations to the IEEE standard. . . the machine contains a floating point emulator module which provides all the functionality of a hardware processor. The instructions it provides may be incorporated into any assembler text. . . . However, these instructions are not supported by the BASIC assembler . . .”.

Using the FPE

It was rather a long time ago, I see my files are dated May to July 1988, that I wrote a program (in BASIC) which allowed me to extend the BASIC assembler to include the instruction set supported by Acorn’s Floating Point Emulator. I have just tried the program out and, with a small tweak to allow it run from my hard disc instead of a floppy, it still works totally and completely either in a task window or full screen. With some reluctance I have decided not to include that BASIC program on the Archive monthly disc because I feel that it might be better to tidy it up before offering it for scrutiny by the Archive readership!

Some of the comments I shall make in this article have been checked using machine code programs assembled using my own floating point extension to the BASIC assembler. It is difficult to capture screenshots of suitably sized task windows but I shall include a couple which demonstrate, if a little crudely, the results I get from running my machine code program.

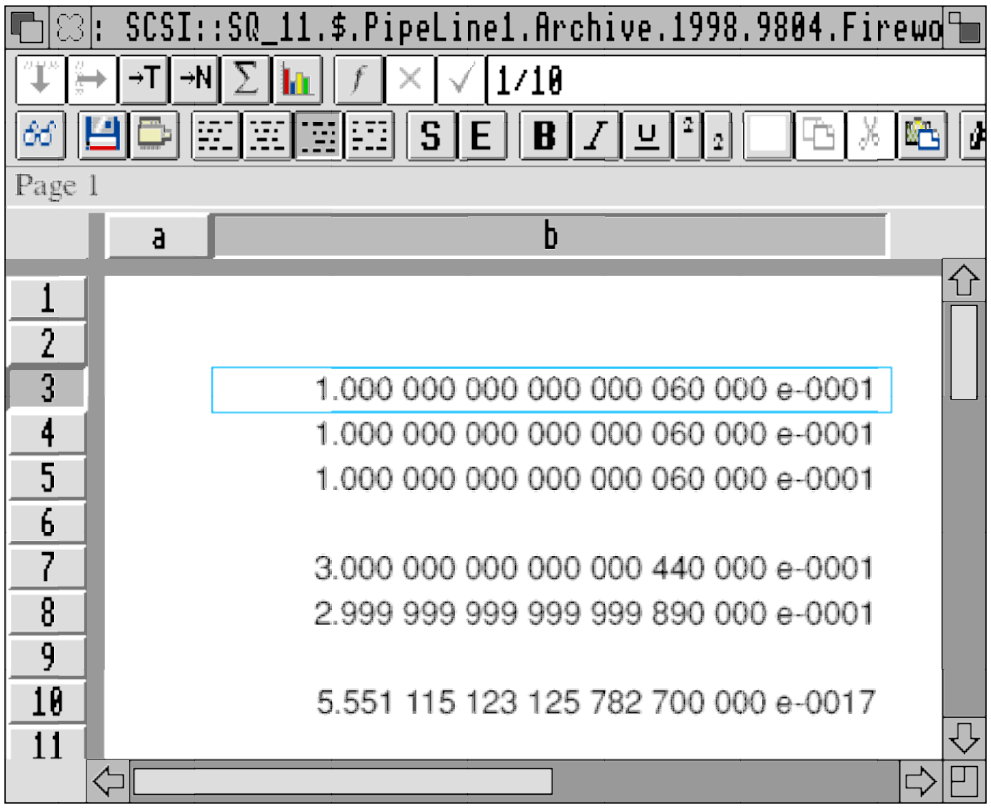
Puzzle Corner

I have no wish to intrude on Colin Singleton’s excellent series so I am offering him the follow up to any replies to the following question. My reason for introducing the question below is to motivate you to study in detail the clues I give to the answer.

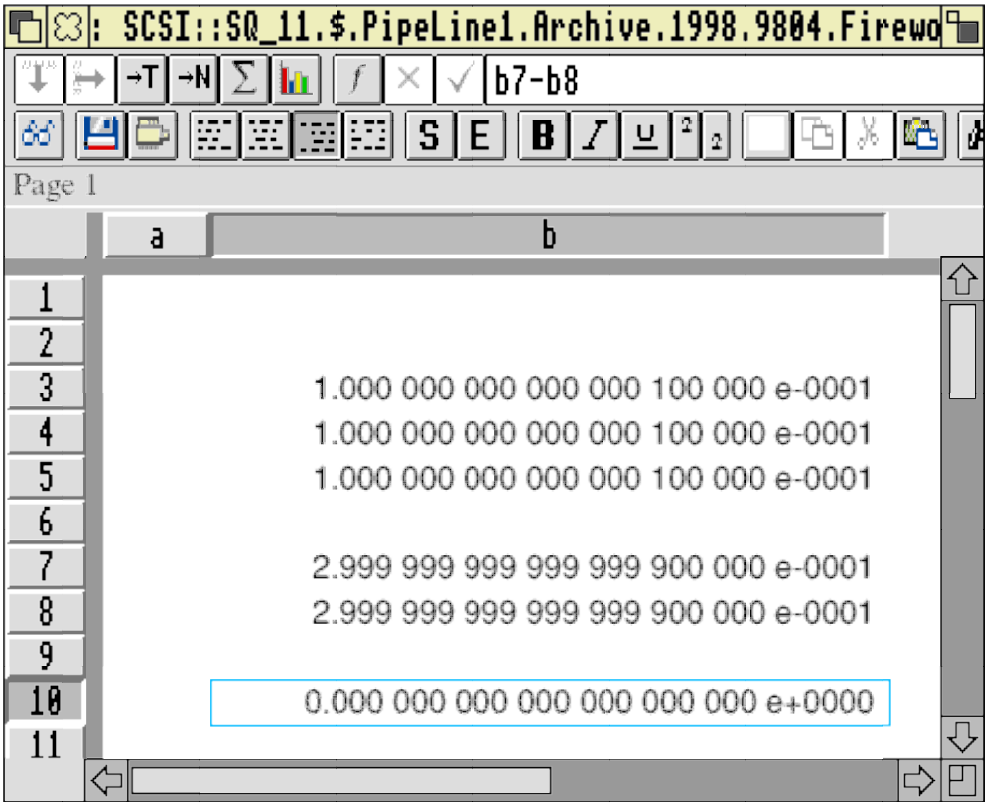
The Archimedes will not store all the Rational numbers but only a finite subset of all Rational numbers. My question is “How many different Rational numbers can be stored in the Archimedes computer?” If you prefer, a simpler version of the same question is “What is the range of Rational numbers which can be stored?”

Fireworkz

In the screenshot below I have chosen to display numbers in a Style which might be described as a Scientific Style. You can find a copy of the Fireworkz file [0point3] in all the usual places. Cell [0point3]b3 contains 1/10 (one tenth as a division sum); cell b7 contains 3/10 and b8 contains b7 − b8.



You will see that the answer in b3 is not exactly 0.1 but that it contains a 6 in the 18th decimal place. I have explained this anomaly before; it is because the number 0.1 can not be stored with complete accuracy in binary. Colin Singleton and others have commented in great detail about this anomaly so I shall not go into it in great depth. However, I am unable to resist including a screenshot of the same unaltered file which appears below.



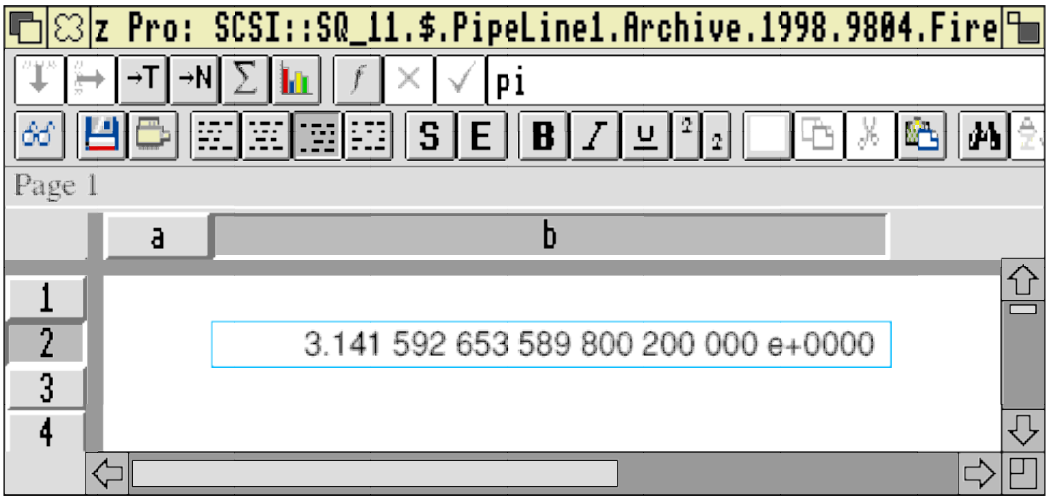
As a partial explanation of the differences I must tell you that the first screenshot uses Version 2.87 (09 Dec 1991) of the FPE module and the second uses V 4.03 (04 Jan 1994). I guess there are later versions around but I don't have a copy. If there is anyone who can email me a later copy then I'd like to give it a try.

Furthermore I have found that forcing a recalculation or Saving and then Loading sometimes changes the values in the spreadsheet!

The Value of π

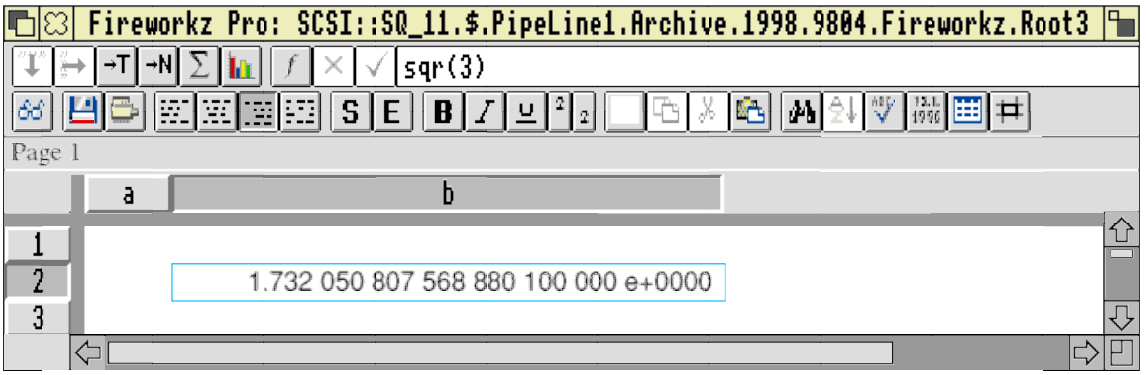
Fireworkz gives a value of π which you can compare with the more accurate value of 3.141 592 653 589 793 239 given by my machine code program. An even more accurate value for those interested is 3.141 592 653 589 793 238 46. You will see that my program gives a value which is correct except for the last place.

The screenshot below is the value of π returned by Fireworkz.



The square root of three

The value of the square root of three returned by Fireworkz is shown in the screenshot below. A more correct value is 1.732 050 807 568 877 293 5.



I apologise for the screenshot below; it is my best attempt to show the output from a task window from my machine code program.

```
Find the square root of what number = 3
When packed it becomes =          000003000
The square root is +1.7320508075688772 94
Again? (A) See the code? (C) The pbcd stor
```

You'll see that the answer from my machine code program does not contain non significant trailing values. Fireworkz regularly fails at the fifteenth significant figure; in general my machine code program is accurate to nearly nineteen significant figures.

Big numbers

I shall refrain from any more Fireworkz screenshots; I'm sure you get the idea. The largest number which can entered into a Fireworkz spreadsheet is 1.797 693 134 862 32 e308. Using my machine code program the largest number which can be handled is close to 1e4092. There are similar values for the big negative numbers and for small numbers close to zero; for example Fireworkz will not handle numbers between zero and about e−308.

Fireworkz V FPE

Fireworkz is not unique. You can discover for yourself similar results if you experiment with other Archimedes spreadsheets or even Excel (which runs on a Windows machine). My motivation for including the results of my own ten year old program is not pride but a demonstration that the Archimedes is capable of more Precision and a greater Range than you might think from the Precision and Range available in spreadsheets.

Floating Point Precision

So! Now we come to the heart of this article. The question I am about to discuss is "What Range of numbers can the Archimedes store and to what Precision?"

The IEEE specify only two levels of Precision, (S), Single and (D), Double Precision. The Archimedes is capable of a third precision they have called (E) Double Extended Precision and another format called (P) Packed Decimal. It is my guess that the many spreadsheets I have investigated use the IEEE (D) Double Precision standard or something close to it.

Within the floating point emulator or hardware the Archimedes uses a format which is not easily visible to a programmer. This doesn't matter because there is no need for the programmer to look into the way in which the floating point mechanism works. To quote from the Programmers' Reference Manual (PRM), "Floating point formats only become visible (to the programmer and user) when a number is transferred to memory using one of the levels of precision described below." The PRM also states "The working precision is ... a 64 bit mantissa, a 15 bit exponent and a sign bit." In theory the 64 bits available for the mantissa allows a precision of just under 19 significant figures and the 15 bit exponent a range of 10^±4932 or so.

Single Precision (S)

Single precision numbers are stored in 32 bits of which 23 are mantissa and 8 are exponent. I don't know of any package which uses Single Precision.

Double Precision (D)

Double Precision numbers are stored in 64 bits of which 52 bits are mantissa and 11 are exponent. According to my early version of the PRM, a Double Precision number has a maximum exponent of (decimal) 340 and 17 decimal digits of Precision. I think that there

is a mistake here. I believe that Fireworkz and the other spreadsheets I have studied use Double Precision and that the Precision of a Double Precision number is nearer to 15 than 17 decimal digits.

Double Extended Precision (E)

Double Precision numbers are stored in 96 bits of which 64 bits are mantissa and 15 are exponent. I do not have access to anything but the earliest Acorn manual for programming in the C Language so I can't check the following statement. I believe that packages such as Fireworkz written in C are written to some IEEE standard and that standard is close to Double Precision and not Double Extended Precision.

This Precision and Range of the E format is the same as that used internally by the floating point number cruncher. According to the PRM "Storing a floating point register in E format is guaranteed to maintain Precision when loaded back into the floating point system . . ."

Packed Decimal (P)

I shall describe this format in some detail, not only because it seems to me to have the same precision as the E format but also because I believe it to be the one most easily understood. Also it is the format which I have used for the output from the floating point emulator of my ten year old machine code program.

The 96 bits of which it is composed can be divided into 24 groups of four bits. Each of these four bit groups represents a decimal number. For example the four bit combination 0000 represents zero, 0001 represents one, 1000 represents 8 and 1001 represents 9. Any four bit combination between 1010 and 1111 inclusive is undefined and shouldn't happen!

If you have followed this explanation of what is usually called Binary Coded Decimal then you will realise that the 24 groups of four bits can represent 24 decimal numbers. These 24 decimal numbers are used as follows. Of the 24, 19 are used for the mantissa and 4 for the exponent. The remaining group of four bits is called 'Sign' and is used to determine the sign of the mantissa and exponent.

If all these (decimal) 19 mantissa and 4 exponent numbers could be used to the full then these numbers would have a full 19 digits of Precision and a Range up to 10^9999. Of course not all this Precision and Range is guaranteed.

Let's concentrate on the Range of numbers first. My machine code program outputs the answer in P, Packed (Binary Coded) Decimal, format. From my experiments I know I can input and output the number 1e4932 but not 1e4933. I have not fine tuned my experiment to find out the exact cut off but maybe you have a means of trying such numbers.

Now to the Precision. I have found that the last two of the 19 mantissa decimal digits are significant when I use the older version of the floating point emulator, V 2.87 but that they are always zero when I use V 4.03! This leads me to believe that 17 digits of precision is all I can hope for in future (for example on a RISC PC) but that, if I stick with V 2.87 on my A540 then I can have just under 19 digits of precision!

Number of Numbers

Under the heading Puzzle Corner I invited you to let Colin Singleton know your calculated value for the number of different Rational numbers which can be stored on the Archimedes. Here is my estimate. There are 17 (or 19) digits of precision for every one of the 4932 values of the exponent. Hence I reckon that a rough estimate is 10^17*4932*4. The 4 arises because (a) the exponent can be negative as well as positive and (b) there are about the same number of negative numbers as there are positive numbers. I make this about 2e21 or 2 000 000 000 000 000 000 000 different numbers; this is a big number but it is nothing like the total number of Rational numbers and is dwarfed by the massively infinite number of Real numbers. If you use the older V 2.87 floating point emulator then you can store about 100 times more Rational numbers on your 'older' machine.

My Machine Code Program

Sometime in the next few months I shall get around to tidying up my program which extends the range of instructions which can be handled by the BASIC assembler. If you have something similar then don't disillusion me but, instead, see if you can confirm what I have said about the Range and Precision inherent in the Archimedes and, with me, you might wonder why spreadsheet packages such as Fireworkz don't use the full precision available. If the people in Australia to whom I sent a copy of my program a decade ago ever got any use out of it then perhaps you'd let me know.

Even More Numbers

$\mathbb{N}$	=	$\{0, 1, 2, 3, 4, 5, 6, \ldots\}$	=	Natural (counting) numbers
$\mathbb{Z}$	=	$\mathbb{N} + \{0, -1, -2, -3, \ldots\}$	=	Integers (whole numbers)
$\mathbb{Q}$	=	$\mathbb{Z} + \{¼, ½, ¾, \ldots\}$	=	Rational numbers (fractions)
$\mathbb{R}$	=	$\mathbb{Q} + \{\pm\sqrt{2}, \pi, e, \ldots\}$	=	Real numbers (smooth continuum)
$\mathbb{C}$	=	$\mathbb{R} + \{\sqrt{-1}, (1 + \sqrt{-1}), \ldots\}$	=	Complex (and imaginary) numbers

We haven't quite exhausted the range of numbers yet. We still have to deal with the most enigmatic type, Complex (and Imaginary) numbers. To read what I have to say about them you'll have to wait until next month when I shall show you a couple of spreadsheets which have Complex numbers 'built in' to their set of functions.

Thanks for all your letters and emails. I'm pleased that you like this series.