BBC · MASTER · ELECTRON · ARCHIMEDES MAY 1989 £1·40

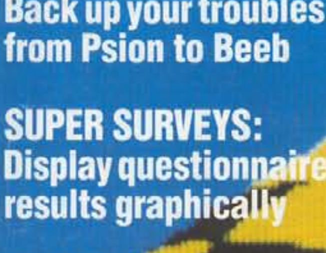
# BBC ACORN USER

**BUDGET PRINTERS:**
Three new low-price models reviewed

**GET ORGANISED:**
Back up your troubles from Psion to Beeb

**SUPER SURVEYS:**
Display questionnaire results graphically

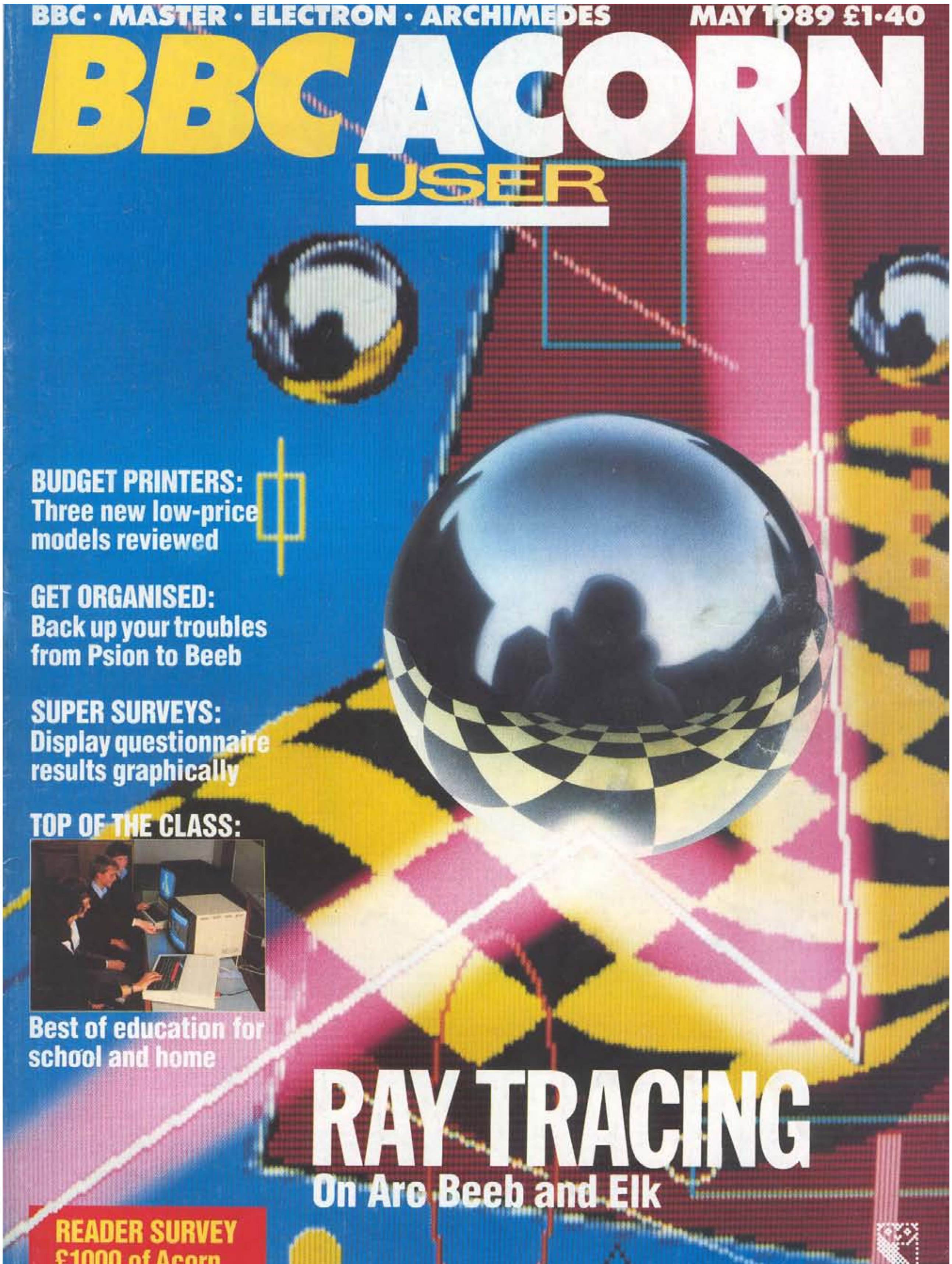
**TOP OF THE CLASS:**



Best of education for school and home

## RAY TRACING
On Arc Beeb and Elk

**READER SURVEY**
£1000 of Acorn kit to be won

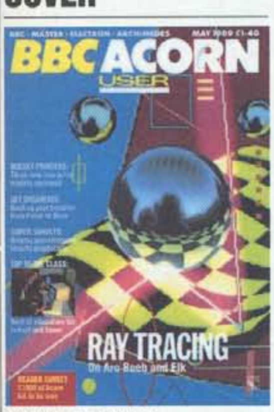


# BBC ACORN USER

ISSUE NO 82

MAY 1989

## COVER



**COVER FEATURE**
Ray tracing comes to the BBC micro and even the Electron on pages 66-71

Cover by Paul Allen

## EDITORIAL

**How often have you, with your model B, tried to get data out of one program into another? If it's a 20k screen from a drawing program, or perhaps just text then it may be possible. But more complex interworking is rare on the 8-bit machines. At BBC Acorn User we've tried to ease this, with explanatory articles and even January's poster on file transfer.**

**But file transfer is much more important with the Arc – in fact Risc OS encourages it. No single program can do everything, so the idea is to use sets of programs together. To make this work, software writers must get it clear: it's vital that you can swap data with other programs easily. After the desktop itself, this is the biggest step from Beeb to Arc.**

**This is my last month as editor – I'm moving on to make DTP work at Redwood Publishing. Good luck to Geoff Bains who's taking over.**

## NEWS

**Unix on Econet, 7**
DTP from Computer Concepts, plus the latest on the *BBC Acorn User* Show

**CUSTOMER HOTLINE 10**

**COMMS COLUMN 13**

**EDUCATION NEWS 15**
£4m from DTI helps schools buy much needed hardware

**NOTICEBOARD 18**
Details of next month's *BBC Acorn User*, results of December's Mega Music competition and the June dates for your diary.

**READER SERVICE 18**

**BLUNDERBOX 18**
Corrections to the PrtSc utility from the August 1988 issue.

## FEATURES

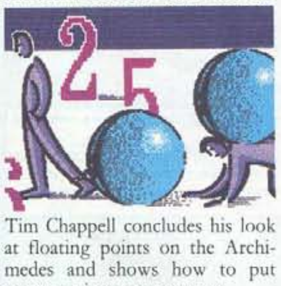
**BALL OF MIRRORS 66**

Richard Browning throws some light on the complex subject of ray tracing to create some realistic 3D graphics

**QUESTION TIME 73**
How can you use your BBC micro sensibly when conducting a survey? Joe Telford discovers the answer and shows some neat ways of presenting the results of a questionnaire

**DATA GETS GOING 80**
The Psion Organiser is the most portable of any computer, but it's size can be a drawback as well as an advantage. Simon Hewitt tells how wiring it up to your BBC micro may overcome this and help you get going

**ARITHMETIC ON THE DOT 85**



Tim Chappell concludes his look at floating points on the Archimedes and shows how to put constants in your programs

**SCHOOL REPORT 88**
Chris Drage and Nick Evans round up the best of educational software for the classroom and the home

## REVIEWS

**GAMES PAGE 121**
Sam Greenhill scans the software market and presents the latest and most interesting catalogue of games releases

**GAMES REVIEWS 123**

Our games fanatics provide the lowdown on *Tank Attack*, the battleship board-game from CDS. Plus there's *Superman* and *Circus Games* the long awaited duo from Tynesoft

**THRIFTY THREESOME 126**

Three low-cost printers prove that nine pins are enough for Graham Bell

**DOS SOLUTIONS 129**
Dave Futcher found Shibumi's *Problem Solver* could give him and his Master 512 an easier life

**PICTURE PERFECT? 132**
Sean Sollé is art critic to Kathy Lewis' *ProArtisan* masterpieces

## SURVEY

**Reader survey 30**
Tell us what you'd like to see in *BBC Acorn User*

## REGULARS

**HINTS AND TIPS 41**
Drawing in mode 7, a Basic line editor and utilities for the Beeb and Elk – all from David Atherton

**ARC AGORA 46**
David Acton introduces a keystrip routine that will also run on your Master, and tells us how to speed up our ROMs and provide easy text-editing

**YELLOW PAGES 97**

**LETTERS 137**

**ADVERTISERS'S INDEX 142**

**FREE ADS 143**

**ACORN ABUSERS' DIARY 144**

**READER OFFERS 65,78,94,135**

## NEXT MONTH

An introduction to PostScript and the Qume Crystal Print Publisher, the more affordable PostScript printer. What are the best databases for education? Plus all your favourites – News, Agora and Hints and Tips.

**Editor** Graham Bell. **Assistant Editor** Pauline McLernon. **Technical Editor** Robert Miller. **Editorial Assistant** Christina Neal. **Production Assistant** Sally-Ann Sweeney. **Art Director** Ian Findlay. **Art Editor** Paul Holmes. **Art Assistant** Robyn Damp. **Ad Manager** Roger Mullins. **Deputy Ad Manager** Perry Hodder. **Sales Executives** Richard Power, Duncan Pringle. **Ad Production** David Noakes. **Publishing Services Executive** Seran Anderson Haddick. **Group Editor** Tony Quinn. **Managing Editor** Ellen Brush. **Publisher** Seamus Geoghegan **Publishing Director** Michael Potter. **Editorial Director** Christopher Ward.

ABC

Published by Redwood Publishing, 20-26 Brunswick Place, London N1 6DJ. Tel: 01-490 1444. Telecom Gold 81: REDoo1. Micronet 919992492. Printed by Riverside Press, Gillingham. Typeset by Ician Communications Group, Harlow. Colour by Trumps Studio, Ware, Herts. Print production by Aquarius Print and Design, London. Distributed by BBC Magazines, 35 Marylebone High Street, London W1. © Redwood Publishing 1988. All rights reserved. Acorn is a registered trademark of Acorn Computers Ltd. Redwood Publishing is a registered data user. ISSN 0263 7456

# ARITHMETIC ON THE DOT

**We conclude our look at floating point instructions by showing you how to put constants in your programs**

***Tim Chappell***

Last month's article showed you how to use the Archimedes floating point emulator, and introduced the complete set of floating point instructions that you can use within your ARM machine code programs. Basic 5 can't handle the assembly of these instructions itself, and so on last month's *BBC Acorn User* monthly disc, there was a Basic library program that allowed you to assemble the floating point instructions. So you could include the line:

```
EQUS ("MNFEAL F1,F0")
```

in your listing to assemble an MNF floating point instruction. But with last month's program, there were still a number of problems with assembling floating point machine code.

This month, there's an improvement to the original floating point assembler program. Due to the length of the listing, it can't be carried in the yellow pages, but the whole thing (the original plus the improvements) are on this month's disc.

The improvements to the original encoder provide several new features. The original demanded that each instruction carried its condition codes – so AL (meaning the instruction is always carried out) wasn't optional as it is with the normal Basic ARM assembler.

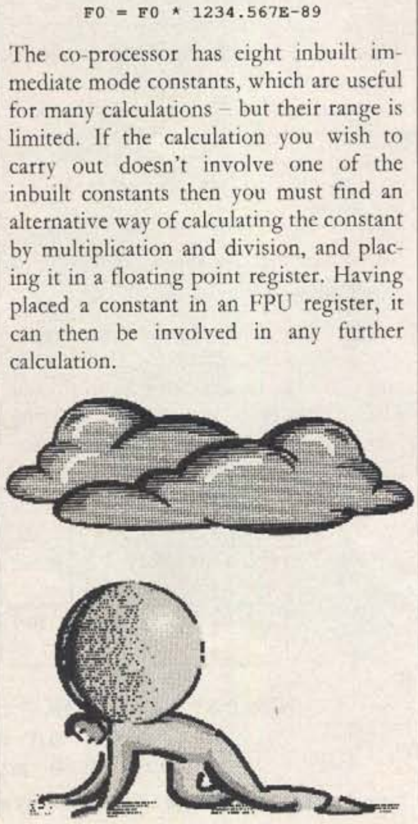
The second problem was storing floating constants like 10.2345 in your program. For integers, you can use EQUD to put a 32-bit value into memory, but there was no equivalent for floating point numbers.

The third problem is that of input and output. Printing a floating point number is difficult. The new version of the library program fixes all three of these points.

If you have experimented with the floating point facility included on last month's disc, you will no doubt have found that although its facilities are very powerful, it lacks an easy way of incorporating constants into calculations. For example the following calculation would prove very difficult to arrange:

```
F0 = F0 * 1234.567E-89
```

The co-processor has eight inbuilt immediate mode constants, which are useful for many calculations – but their range is limited. If the calculation you wish to carry out doesn't involve one of the inbuilt constants then you must find an alternative way of calculating the constant by multiplication and division, and placing it in a floating point register. Having placed a constant in an FPU register, it can then be involved in any further calculation.



There are several ways to do this. First, if the constant is an integer and can be stored in an ARM register, it can be transferred from there to an FPU register using the FLT instruction:

```
MOV R0,#32
EQUD FNencode("FLTE F0,R0"); F0=32
```

But this method is limited – only integers can be transferred, and many constants will not be integers. You are also limited to the 32-bit size of the ARM register.

Second, if the constant can be calculated using the FPU instructions, then this may be useful. So to place the value of pi in F0, use:

```
EQUD FNencode("MNFE F0,#1")
EQUD FNencode("ACSE F0,F0")
```

This works because pi is the arcosine of -1 (remember the trigonometric calculation work in radians).

Third, you could manufacture the constant by multiplication and division of either immediate constants or integers transferred from the ARM registers.

But the best way to do this is to load the constant directly from memory. This provides an adequate range of constants from 9E+9999 to 1E-9999 with 18 decimal places.

To store a constant in memory and then load it into an FPU register, you need to be aware of the format in which the FPU stores its numbers. There are in fact four different formats in which a register may be stored: single precision, double precision, extended double precision and packed binary coded decimal – the same as the four precisions that a number can be kept in. It is packed BCD which is of most use to us, since it is easier to convert normal scientific notation into this format. The format requires three 32-bit words to store the constant in, with the format shown in table 1.

Each decimal digit in a BCD number is put in four binary bits – so 205 would become 001000000101. The top four bits of the 96-bit packed BCD are the sign bits saying whether the number is positive or negative, and the nibble is arranged as follows:

| | | | |
|---|---|---|---|
| M | E | 0 | 0 |

M is the mantissa's sign, E is the exponent's sign. If the bit is set, it means the number is negative. A number like 1.23E-4 (0.000123) has a clear (positive) M bit but a set (negative) E bit.

The routine in the new version of the floating point assembler provided on the May monthly disc converts a number passed to it in string form, into the corresponding 12-byte string which represents the constant to the FPU. Since the routine always returns a 12-byte string, there is no need for an ALIGN directive following an included constant. Constants may be introduced into a program using

```
        EQUS FNfconst("number")
```

Where the 'number' string is one of the following forms:

```
  +/-mmmmm.....                 -3456
  +/-m.mmmm                      3.456
  +/-mE+/-eeee                  -6E-23
  +/-m.mmmmE+/-eeee              6.789E10
```

STR$ (value) is also acceptable where value is a normal Basic variable.

The number for conversion is passed as a string because this allows numbers of much greater precision than Basic itself allows to be included as constants (otherwise Basic would round them to its usual nine or 10-digit precision before passing them to the routine).

The following code can be used to transfer a constant from memory to an FPU register:

```
..
ADR R0,const1
EQUD FNencode("LDFP F0,(R0)")
..
.const1 EQUS FNfconst("22.12E-69")
```

The ADR directive points the ARM register R0 to the storage area for the constant, and the LDFP command loads the BCD constant 22.12E-69 into F0.

To convert the number stored in packed BCD form to one of the other formats so it can be used in double or extended precision arithmetic, you use:

```
..
ADR R0,value
EQUD FNencode("LDFP F0,(R0)")
EQUD FNencode("STFE F0,(R0)")
..
.value EQUS FNfconst("11.04E-66")
```

This loads the value into F0 in BCD, then stores it back at .value in extended precision. Extended precision takes up three 32-bit words just like the packed BCD format, so you don't need to reserve any extra space.

This is particularly useful if you wish to carry out further calculations on the constant and don't wish to lose any accuracy. Storing a value in extended precision format ensures no further loss of accuracy occurs.

The third improvement to the encoder allows output of floating point numbers from the unit. The routine will convert any packed BCD (P) format number into a string with a specified number of decimal places:

```
..
ADR R0,store
EQUD FNencode("STFP F0,[R0]")
..
.store EQUD0:EQUD0:EQUD0; storage area
]
PRINT FNfprint(store,9)
```

The FNfprint routine has two parameters. The first is the address of the floating point number (which must be in packed BCD format only – it can't deal with the other three formats). The second is the number of decimal places required (so in the above example nine decimal places will be returned). This can be any value between 0 and 18.

Note that although there are a maximum of 18 decimal places, you can't always rely on all of these. Calculations in other precision modes will limit the precision of the calculation result. Generally, if a calculation has involved a single precision number, then only nine digits will actually be significant and for a double precision calculation only 17 digits. Note that the instructions FML, FRD and FDV are only calculated to single precision accuracy.

Since the routine returns a string, then you can convert it into an actual Basic variable using the Basic VAL instruction. Two words of warning – because the floating point system can manipulate numbers far larger than Basic, then an error may occur if the value is too large for a Basic variable to hold.

Similarly some values will be too small to be held in a Basic variable, and will be rounded to zero. Second, remember to assign the value to a floating point variable. So x=VAL string$, not x%=VAL string$.

As a bonus, you can check with the FNsuitable routine that a number can be expressed properly by a Basic variable. This it will return TRUE if it can, or FALSE if it is too large or small. The following is an example of its use:

```
IF FNsuitable (address,places) THEN
   A=VAL(FNfprint (address,places))
```

The two parameters required for the function are the memory address of the number to be checked (in packed BCD form) and the number of decimal places required.

That completes the floating point encoder, and gives you all you need to make full use of the floating point emulator in your ARM machine code programming.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| S | e3 | e2 | e1 | e0 | m18 | m17 | m16 |
| m15 | m14 | m13 | m12 | m11 | m10 | m9 | m8 |
| m7 | m6 | m5 | m4 | m3 | m2 | m1 | m0 |

```
Where: S holds the sign bits
       en is an exponent digit
       mn is a mantissa digit
```

Table 1. Packed BCD format allows a 19-digit mantisse and 4-digit exponent