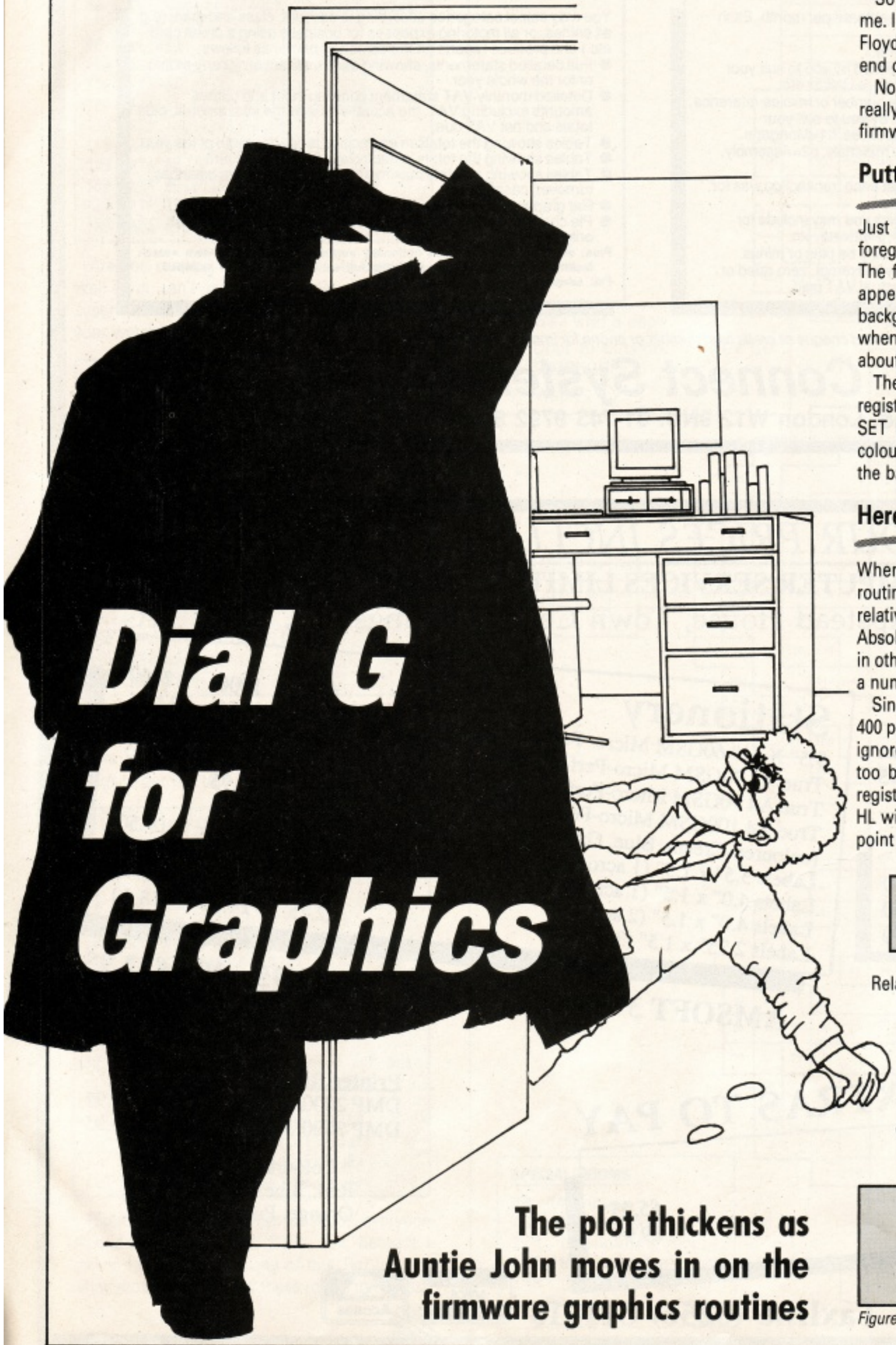
Dial G for Graphics

The plot thickens as Auntie John moves in on the firmware graphics routines



WELCOME once again to the exciting world of Auntie John's machine code. As you may know, there is a short delay between when I write this and when it actually appears in print. This delay is typically about 15 years, which means I am writing this is 1973.

This presents me with some problems, not least the fact that I am only six years old. Also, since Z80 microprocessors won't exist for a number of years yet, I am in a sense working blind when I give specific listings. I hope you will bear this in mind if any mistakes crop up. Thank you.

So back to reality, and this month we're going to take a look at the set of Firmware calls that deal with everyone's favourite topic: Graphics.

These calls are particularly simple, and deal with moving, plotting, testing, drawing and setting up graphics windows. In fact, you can't do anything from machine code that isn't a lot easier from Basic, and the speed of machine code does not really make any difference when plotting pixels.

So why bother looking at it at all? Sure beats me. I tell you what, I'll go and listen to some Pink Floyd and you talk among yourselves until the end of the column.

No, it doesn't work, does it? Deep down you really want to know all about the graphics VDU firmware calls. So here we go...

Putting pen to paper

Just like in Basic, the graphics routines have a foreground and background colour to work with. The foreground is the colour that all the points appear in when you PLOT a point, and the background is the colour that the screen goes when you clear the graphics window – more about the window later.

These colours can be chosen by loading the A register with the ink required and calling GRA SET PEN (&BBDE) to change the foreground colour, and GRA SET PAPER (&BBE4) to change the background colour.

Here and there

When calling the plot, move, test and draw routines you have a choice between absolute and relative coordinates – just like PLOT and PLOTR. Absolute coordinates are given in 16 bit form. Or, in other words, two bytes. Or, in yet other words, a number from 0 to 65535.

Since the screen is always 640 points across by 400 points high, any values bigger than these are ignored. (Two bytes are needed because 640 is too big to fit into a single byte, innit). The DE register pair is loaded with the X coordinate, and HL with the Y coordinate. For example, to plot a point in approximately the middle of the screen:

```
ld de,320   ;x co-ordinate
ld hl,200   ;y co-ordinate
call &bbea ;gra plot absolute
```

Relative coordinates need signed 16 bit num-

```
Decimal = 42
Binary  = 0010 1010
Invert 1s and 0s = 1101 0101
Add  1 = 1101 0100 = −42
```

Figure I: Converting to Two's-complement binary

```
    0010 1010  (42 in decimal)
+   1101 0110  (-42 in decimal)
    ---------
  1 0000 0000  (0 in decimal)
  the overflow bit is ignored
```

Figure II: Checking the result

bers. If you haven't come across the concept of signed binary numbers, then you have a treat in store. Here is a quick explanation.

Sign of the times

With eight bits, you can represent the numbers 0 to 255, OK? Now, if we treat the same eight bits in a different way, we can get them to represent the numbers –128 to 127. Nothing has changed in the binary number itself, its just the way we treat it.

The technique normally used is called Two's-complement, and to convert a positive binary number to its negative Two's-complement you just change all the ones to zeroes, all the zeroes to ones, and add one. An example, using the number 42, can be seen in Figure I.

So, the Two's complement of 42 is – in binary – 11010110. Notice that in decimal this number could be taken to be 214. Adding 42 and 214 gives 256, but since we are using 8 bit numbers, this leaves with us the value zero, which is what you would expect when you add 42 and minus 42. See Figure II if this is confusing you.

Of course, if your assembler allows you to put a minus sign in front of numbers – like LD DE,–10 – then you don't have to know any of this. Still, it's nice to know what's going on. So I'm told.

So, DE and HL are loaded with signed X and Y offsets, and then the routine is called. For example, to perform the equivalent to Basic's TESTR –2,4:

```
ld de,-2   ;or ld de,&fffe
ld hl,4
call &bbf3 ;gra test relative
           ;A now contains ink value
```

Drawing the line

The firmware LINE routine is the equivalent to DRAW in Basic. You supply the relative or absolute coordinates of the end of the line, call the relevant routine and the computer will draw a line from the last cursor position to the point you have specified.

The "last cursor position" is the last point you moved to, plotted, tested, or even finished a drawing an earlier line to. If you haven't done any of those things yet, the line will start from (0,0).

The example program in Listing I will draw a box in the current colour all around the outside of the screen. The routine draws a box starting from the bottom left, and proceeds in a anti-clockwise

```
org &4000

;gra move absolute
ld de,0
ld hl,0
call &bbc0

;gra line absolute
ld de,639
ld hl,0
call &bbf6

;gra line relative
ld de,0
ld hl,399
call &bbf9

;gra line absolute
ld de,0
ld hl,399
call &bbf6

;gra line relative
ld de,0
ld hl,-399
call &bbf9

ret
```

Listing I: Drawing a box around the edge of the screen

direction. I've used both relative and absolute coordinates to give you something to think about.

Other routines exist in the firmware to plot vertical and horizontal lines. These routines are SCR HORIZONTAL at &BC5F, and SCR VERTICAL at &BC62. Both require the A register to contain the encoded ink that the line is to be drawn in (encoded inks were mentioned last month). Since these routines duplicate functions already examined, there is little to be gained – except perhaps another paragraph – in examining them. So I won't.

Origin tonic

As from within Basic, you are allowed to change the origin of the graphics display – the "origin" is where the computer thinks the coordinates (0,0) are. Initially, the origin is at the bottom left-hand corner. This means all the coordinates used to plot any point of the screen are positive, which makes things nice and simple.

However, if you really want to, you can choose to put the origin anywhere in the screen that you like, and as an example we'll put it slap-bang in the centre, at coordinates 320,200. The code for this is:

```
ld de,320  ;x co-ordinate
ld hl,200  ;y co-ordinate
call &bbc9 ;gra set origin
```

Now if you were to use GRA PLOT ABSOLUTE with coordinates (0,0), the pixel would be plotted in the centre of the screen, at what used to be (320,200). Confused? Good. Now think of what you'll need to do if you wanted to plot a point at what used to be (100,100). Hmm, below and to the left of the origin. You'd need negative values, right? Which means good old two's-complement signed numbers again. Personally, I don't shift my origin around too often...

Clean windows

At last we come to the graphic window routines. These allow you to define an area of the screen to be used for graphics: Any points outside the window are not plotted, and when tested return the background ink.

To define a window, you must define first the left and right edges, and then the top and bottom edges. The coordinates used are all absolute, and don't depend on the position of the origin at all. That is, (0,0) is always the bottom left corner of the screen.

To define a graphics window in the centre of the screen:

```
ld de,160  ;left edge
ld hl,480  ;right edge
call &bbcf ;gra win width
ld de,100  ;top edge
ld hl,200  ;bottom edge
call &bbd2 ;gra win height
```

To clear the graphics window, simply call GRA CLEAR WINDOW at &BBDB which uses the current background ink. Using these routines, you could very quickly fill blocks of the screen to be any ink colour, and – unlike SCR FLOOD BOX – to pixel accuracy.

Chocolate digestives

Speaking of disc drives ... weren't we? Never mind, I find links very difficult. But I thought you might be interested to know who invented them.

About 10 years ago, in 1963, a Dr Hans Flexgutt was experimenting with digestive biscuits and the tricky problem of how to spread the chocolate over them in a nice even coating. He hit upon the idea of dropping a blob of melted chocolate on to the biscuit while rotating it at speed.

After several very messy experiments he discovered the exact speed to spin the biscuit and also developed the hardware needed to guide the chocolate spraying nozzle with the desired accuracy. It was with some surprise that he discovered that in the process of applying the chocolate in small bursts, he had invented the world's first disc-drive.

He went immediately to IBM with his invention, but they stole the idea and applied it to pizzas instead – and thus the eight inch disc was born. Dr Flexgutt was very upset about this and spent the rest of his days in the fruitless search for a substitute for food. Shortly later he was found dead in his laboratory by his assistant, a certain

Mr A.M.Sacarin. The post mortem revealed his stomach was full of iron filings.

However, now, in the '70s, a small company called Amstrad has rediscovered the chocolate digestive disc drive. Only time will tell whether or not it will become a viable proposition. Much development is needed – a small black plastic shell to put the biscuit in for a start, and probably a clear plastic box for protection.

Cunning logic

Finally we come the routine GRA WR CHAR at &BBFC, which is the firmware equivalent to Basic's TAG command. Remember TAG? You use it (very often, I don't think) to print text at the graphics cursor. This allows you to place letters and numbers to pixels instead of character squares.

The final example program (Listing II) uses this routine to print strings of text in a very pretty way. It also uses the SCR ACCESS routine (&BC59), which determines how a plotted pixel will affect any pixels that are already on the screen. It does this by looking at the pixel that already exists and performing a logical operation – AND, OR, XOR – with it and the pixel to be plotted. The result is a mess. Well, it is in most of my programs.

In Listing II we print text in one colour, then overwrite it in another colour, but move a pixel to the side and a pixel up. The cunning bit is the use of the logical operations, which still allow the original colour to show through.

```
org &4000           ;start of assembly

gra_move_absolute equ &bbc0
gra_set_pen       equ &bbde
gra_wr_char       equ &bbfc
scr_access        equ &bc59
scr_set_mode      equ &bc0e

ld a,1
call scr_set_mode ;choose mode 1
ld a,3
call scr_access   ;choose OR mode
ld a,3
call gra_set_pen  ;set graphics pen 3
ld de,100              ;position
ld hl,300                ;graphics
call gra_move_absolute   ;cursor
ld hl,message      ;point at message
call print_message ;print message

ld a,2
call gra_set_pen  ;set graphics pen 2
ld de,102
ld hl,302
call gra_move_absolute ;offset cursor
ld hl,message      ;point at message
call print_message ;print message

ret                 ;return to Basic

.message db "Amstrad Computer User",0

print_message
;
; A subroutine to print a string
; pointed to by HL at the current
; graphics cursor position and in the
; current pen and plotting mode.
; String must end in 0.
;
.loop
ld a,(hl)
cp 0
ret z
push hl
call gra_wr_char
pop hl
inc hl
jr loop

end                 ;end of assembly
```

Listing II: Printing strings in graphics mode

Try experimenting with different ink values and different logical operations. The list of logical operations is in the Basic Users manual – in the section about Control Characters, beside character &17.

● *Until next month, take care and Keep On Assembling!*

Slash Viewdata Costs with the Cage COMMS ROM

THE package designed from the start to keep online time to a minimum. With new time charges, the Cage ROM can easily pay for itself in under three months use. It can cut online time to $^1/_4$ of that used with old fashioned viewdata software.

Time Saving features:

Sophisticated off line message preparation.
Frame Tagging – go directly to a chosen frame.
Ultra rapid frame capture – grab your frames or mail and read them offline at leisure.
Automated Log on/Log off
Automated message sending (Send up to 26 mailboxes of any type at speeds of up to 2400 bd – using a suitable modem – with just a single keypress).

The Cage Comms ROM replaces existing software in Amstrad/Pace/Honeyview, (AMS) or KDS/Sven (KDS) I/Fs

Now YOU can use the software chosen by Micronet for their NEW ProPak package.

Other Features:

Very easy to use
94 Frame stores (6128) to store captured frames, or messages
Full & part frame copying
Automatic Downloading and decompaction
Comprehensive 60+ page manual
Autodials any autodial modem
Superb printing facilities
Professional editing facilities

What the experts said of the Cage Comms Rom

Telephone bills – argh – can be kept to the minimum. Great for Information Providers. ACU

Very good indeed. An excellent viewdata Package . . . amazing. MICRONET – Amstrad Microbase Editor

Can slash your online time. Best CPC viewdata editor yet. I unreservedly recommend it for Viewdata. It's the first one I've used that is powerful, bug free and I've felt happy with. I now use it in preference to anything else. What more can I say? CwtA

This is the comms package that others will be measured by. Over the last few years, I have seen and used a lot of different combinations of machines and comms software. On the basis of what I have seen available, I have no hesitation on recommending the CAGE for any CPC viewdata user. AMSTRAD ACTION

Knocks spots off ALL its competitors. I use the CAGE (comms rom) constantly. WACCI

When ordering, please state type required, AMS or KDS

Cheques should be made payable to: I M Hoare

Amster's Cage (DP)
46 Connaught Road
LONDON NW10 9AG

Telephone: 01 965 8957
PRICE £34.50