To Boldly Go...

Auntie John's several year mission grinds to a halt as he goes where no sensible person has gone before: Beyond the firmware

Where did we finish last month? Oh yes, I remember: slowly. We were bouncing a ball around the screen and everything was happening real slow. And what conclusion did we come to? Is rhetoric overrated? Is the new series on Star Trek any good?

Ahem. Sorry about that. Programming in machine code always gets me excited. Back to the real world. The way to speed up our graphics is to stop using the firmware. The firmware, as you should recall, is the set of routines built-in to the CPC to handle everything from the keyboard to the sound chip.

You might wonder why the firmware is a bit slow. You might think that the programmers who wrote it werea bit sloppy and didn't really want to go very fast. You'd be wrong.

The firmware routines were written to be flexible. They were also written to be reliable. So, for example, the routine to display a letter on the screen works in any screen mode. All the time. Reliably. This sort of coding takes time to execute, but I'm sure you would rather use a computer which worked all the time, rather than one which worked a bit faster but crashed a lot. Wouldn't you? What? You have seen an Atari ST?

The last program we wrote bounced a ball around the screen. However, since it used the firmware to draw the ball, it went slowly. So lets dump that part of the firmware.

To write things to the screen, you need to understand what is actually happening. It's actually quite simple, because what you see on the screen is nothing more than a big block of memory. Sixteen kilobytes (16K) to be exact.

When the screen is empty (just after a CLS for example), all the memory has been reset to hold zeros. Nothing in the screen memory means nothing on the screen. Simple.

When you change the contents of the screen memory, you suddenly start to see things on the screen. Here, try this BASIC program:

```
10 MODE 1
20  POKE &C000,1
30 GOTO 30
```

The last line is just to keep it busy while you examine the screen.

Pressing ESC twice will sort it out.

See the tiny pixel on the top left? That's a piece of the screen memory which you changed by POKEing it. The screen memory starts at (hex) address &C000.

Just how the value you place in memory effects what pattern of pixels is displayed on the screen is a complicated topic. It depends on the screen mode used and what colours your pens and paper are set to. The best way to find out is to experiment with the listing above, trying different numbers from 1. Keep notes, and you'll soon suss it out. Start with MODE 2, as this has only two colours and is simplest to understand.

If you remember your binary arithmetic, you'll have a head start. If you start playing with either MODE 1 or MODE 0 remember to set your logical pens to some obvious colours. The

the pen number, not the colour!

OK, so we can put a small line directly on to the screen with a POKE &C000,255. To put a line next to it, we can POKE &C001,255. In fact to draw a line across the entire screen we can keep adding one to the memory address. It's only until we get to the edge that the problems start.

You might hope that once at the far right of the screen, adding a further one will bring the line back on to the far left. It doesn't. What normally happens is that the little line goes into hyperspace for a bit, then reappears somewhwere else further down the screen..

Of course, there is a logical pattern to how the addresses refer to different parts of the screen. And of course it would take an entire article to explain.

So (as usual) we'll cheat.

Although I promised not to use the firmware to write things to the screen, I didn't say I wouldn't use the firmware to calculate the screen addresses. Sneaky, eh?

Just for us, those nice CPC designers gave us PREV_LINE and NEXT_LINE to calculate the screen address immediately above and below the current address. Yes, really! You just load hl with the address you know, call the relevant routien and Hey Tesco! hl now has the new screen address in it. As you can imagine, this is quite useful.

The example listing this month will draw a line all away across the screen. Then it waits for a bit, erases it and moves it down. This happens all the way down the screen. Then it starts moving up again. It looks very nice, take my word for it.

It also gives you lots of scope for experimenting. Try changing the value loaded into the A register. This is the avlue we POKE on to the screen, which explains why zero is used to erase it. The value of fifteen is used because in MODE 1 it draws a solid bar. When MODE 2 is used, you will end up with something completely different.

Finally, try this for a surprise. Instead of LD A, O put LD A,R in instead. I guarantee some interesting results!

Right, now I'm off for a rest. All this machine code has been playing on my mind and I need a break. If you care to remember, I have been Z80 coding now for years and years...

If you really want more programming, the best thing to do is write to the Editor and tell him exactly what.

In the meantime: Well it's been fun. See you around!

LISTING

```
org &8000

set_mode equ &8c0e
next_line equ &bc26
prev_line equ &bc29
frame equ &bd19

                ld a,1
                call set_mode               ; Try a different mode!

                ld b,199                    ; All the way down the screen...
loop1:          push bc
                call draw_thing
                call frame
                call erase_thing
                call move_down
                pop bc
                djnz loop1

                ld b,199                    ;and all the way up...
loop2:          push bc
                call draw_thing
                call frame
                call erase_thing
                call move_up
                pop bc
                djnz loop2

                ret

move_down
                ld hl, (thing_address)
                call next_line
                ld (thing_address),hl
                ret

move_up
                ld hl, (thing_address)
                call prev_line
                ld (thing_address), hl
                ret

draw_thing
                ld hl, (thing_address)
                ld a,15                     ;Change this number!
                ld b,80                     ; All away across the screen...
loop3:          ld (hl),a                   ;POKE it into the screen,
                inc hl                      ;and move it across.
                djnz loop3
ret

erase_thing
                ld hl, (thing_address)
                ld a,0                      ; Change this number!
                ld b,80
loop4:          ld (hl),a
                inch hl
                djnz loop4

ret

thing_address dw &c000                      ;A variable to store our screen
                                            ; address. Takes two bytes.
```