The Return of Auntie John

Back by popular demand for one more farewell gig, AJ explains how to clear off properly! (And with the appropriate amount of style!)

So there I was, happily whiling away the hours in suspended animation when the editor 'phoned me in a state of panic.

If you have ever met our illustrious editor you won't be very surprised by this, as he seems to be in a perpetual state of panic. Indeed, when he is not running around the place shouting, people get nervous.

"Quick, quick Auntie!" he said, "I need your help!"

"What, again?" said I. "Don't tell me: Nicola Hemming from Rainbow Arts is coming to visit and you want some hints on being cool."

"Not quite. I need you to do one more very last Auntie John's Machine Code Ever for me"

"Hmm, I don't know" I said. After all, it had taken a great effort to write the previous very last one ever.

"We'll pay you twice what me normally pay you!" he begged.

"What! Two Mars bars? Right – it's a deal."

"Great. Thanks AJ. Oh by the way, do you think I should wear sunglasses when she comes?"

So I fired up the word processor, and wrote one more Last Ever Auntie John's Machine Code. And here it is. Keep it to show your grandchildren: It's a one off special and you'll never see anything like it again. If you're lucky.

Since I'm about to clear off for a bit, what better to discuss than ways of making your computer clear off too. Well, clearing the screen display anyway.

The traditional way to clear the screen is to use the CLS command from BASIC. It certainly clears the screen, but it doesn't do it in a very spectacular way. One minute the screen is full of stuff, the next it's blank. Pretty boring actually.

What we need is something a little more showy. Something with a little class. We need the Auntie John's Screen Clearing program!

As a special Christmas pressie, I'm going to give you a collection of screen clears. They are all built into the long BASIC listing called "THE LONG BASIC LISTING". What this program does is Poke all the machine code instructions into memory and then perform a quick demo for you. First however, we'll need to examine the machine code routines needed to clear the screen.

The first routine is called "Blinds", because it sort of looks like a set of blinds closing on the screen. Well, it does to me anyway. Don't worry about it.

It works in a way which if you have been following the last few month's articles will be quite familiar. Yup, it pokes into the screen ram.

Now we know that as long as the screen hasn't been scrolled up or down that it starts at hex location &C000. In other words, if we did a quick POKE &C000,255 we would get a little line at the top left of the screen.

Now if we repeat the poke for the entire screen (all 16K of it!) and used different values each time we would get a nice pattern. Probably. Here's the listing, have a look.

```
;Blinds
            ld a,255          ;Colour3
            call blinds
            ld a,240
            call blinds       ;Colour 2
            call binds
            ld a,15           ;Colour 1
            call blinds
            ld a,0            ;Nothing!
            call blinds
            ret
.blinds
            ldhl,&C000        ;Start of screen memory.
            ld d,a            ;Keep a register 'cos we muck
.loop1      ld a,d            ;it up later.
            ld (hl),a         ;Do the POKE
            inc hl            ;Move to next location
            ld a,h            ;Check for end of screen by
            or 1              ;Checking if HL has got so big
            cp 0              ;it has passed FFFF and become
            jr nz,loop1       ;zero again.
            ret
```

The only tricky bit in the program is the final check to see if all the screen has been cleared. We could have used another register pair to count from &0000 to &4000 for us (remember: &4000 is 16K – the length of the screen). Instead we use a bit of cunning. Since the screen starts at &C000, the end of the screen will be &C000 + &4000 which equals &0000. It actually equals &10000, but as we are dealing with register pairs which can only hold a maximum of &FFFF, the number 'clocks' to zero. All we have to do then is check to see if both registers in the HL pair are 0.

Time for more cunning. Instead of checking each one, we OR them together, which is a bit like adding them. If the answer is not zero, we know one of them ISN'T zero so we keep going. Otherwise, we return to BASIC.

The next clear screen routine doesn't have a name, because I'm not sure how to describe it. Best type it in and have a look yourself.

```
;Thingy
        RomOff EQU &b903
        RomOn EQU &b900

        call RomOff
        ld b,8
loop2:  call thingy
        djnz loop2
        call RomOn
        ret

.thingy
        ld hl,&c000     ;Start of screen memory.
.loop3  ld a,(hl)       ;Peek the value
        srl a           ;Change the value (divide it by 2)
        ld (hl), a      ;Do the POKE
        inc hl          ;Move to next location
        ld a,h          ;Check for end of screen by
        or 1            ;checking if HL has got so big
        cp 0            ;it has passed FFFF and become
        jr nz,loop3     ;zero again.
        ret
```

This one works by Peeking the screen value, dividing it by two and putting it back. Because our divide routine doesn't support decimal points or silly things like that, after any number has been halved eight times it is zero, and the screen is cleared.

Remember that because there is an operating system rom overlapping the screen ram, we must disable it before we can look at what is underneath. It's good manners to enable it again when we have finished.

Last routine is called "Static", because it makes the screen look like the static you get when you mistune a television set. Almost.

```
;Static
        ld b,8
loop4:  call static
        djnz loop2
        ret

.static
        ld hl,&c000     ;Start of screen memory
.loop5  ld a,r          ;Get a 'random' number
        ld (hl), a      ;Do the POKE
        inc hl          ;Move to next location
        ld a,h          ;Check for end of screen by
        or 1            ;checking if HL has got so big
        cp 0            ;it has passed FFFF and become
        jr nz,loop5     ;zero again.
        ret
```

The random number is obtained from the 'r' register. This little used register is not normally useful on the CPC. It is used by the Z80 processor when refreshing memory. As far as we are concerned it contains a number which we can't predict. When written to the screen, it looks like rubbish. Note that this routine cheats a bit and needs a proper CLS after it to remove all the garbage on the screen.

All these routines are merged into one big listing which anyone can type into their CPC (any version) and use from their BASIC program. Look at the BASIC demo bit to understand how to use them, and what addresses to call. Try using different screen modes!

Right that's it! It's over! No more machine code! I'm off to take a well-deserved lie down while I work out what to do with myself. Suggestions care of the editor, please.

Have fun,
Your Auntie,
John.
XXX

THE LONG BASIC LISTING

```
10 REM DEMO PROGRAM FOR CLEAR OFF!
20 REM BY AUNTIE JOHN
100 MEMORY &7FFF
110 A=&8000
120 READ B$
130 IF B$="STOP" THEN GOTO 190
140 FOR C=1 TO LEN(B$) STEP 2
150D=VAL ("&"+MID$(B$,C,2))
160 POKE A,D
170 A=A+1
180 GOTO 120
190 MODE 1
200 GOSUB 500
205 REM THE FIRST CLEAR OFF
210 CALL &8000
215 REM THE SECOND CLEAR OFF
220 GOSUB 500
225 REM THE THIRD CLEAR OFF
230 CALL &8023
240 GOSUB 500
250 CALL &8040
260 CLS
270 END
500 FOR T=1 TO 100
520 MOVE INT (RND*640), INT (RND*320)
520 DRAW INT (RND*640), INT (RND*320)
530 NEXT T
900 REM THE MACHINE CODE - BE CAREFUL TYPING!
910 REM * SAVE BEFORE YOU RUN *
1000 DATA "3EFFCD15803EF0CD"
1010 DATA "15803E0FCD15803E"
1020 DATA "00CD1580C92100C0"
1030 DATA "577A77237CB5FE00"
1040 DATA "20F7C9CD03B90608"
1050 DATA "CD318010FBCD00B9"
1060 DATA "C92100C07ECB3F77"
1070 DATA "237CBFFE0020F5C9"
1080 DATA "0608CD488010FBC9"
1090 DATA "2100C0ED5F77237C"
1100 DATA "B5FE0020F6C90000"
1110 DATA "STOP"
```