ONLY £1.85

NEW

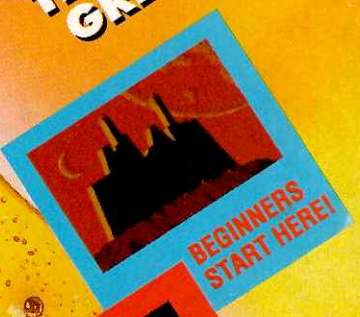
# CPC Attack!

THE CRUCIAL AMSTRAD CPC READ!

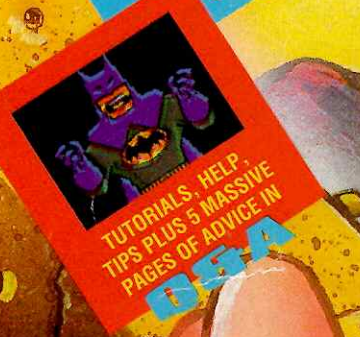
NO1 JUNE 1992

## CONSOLE CRAZY!

BUYING A CONSOLE? THEN GET THIS GREAT GUIDE



BEGINNERS START HERE!



TUTORIALS, HELP, TIPS PLUS 5 MASSIVE PAGES OF ADVICE IN Q+A

EXCLUSIVE!

## Oh Yes! Lemmings!

## GAMES REVIEWED

TOP CPC PRINTER BUYERS' GUIDE

+ WIN

THE EPSON LQ570 24-PIN PRINTER AND LOADSA GAMES

AMAZING FREE 3D GLASSES

CHECK OUT THE AMAZING 3D POSTER INSIDE

9 770965 719002 06

# A SCROLLY DEMO

MACHINE CODE TUTORIAL

## Dust down your assemblers and limber up your fingers - we're going to write a scrolly...

Demos are an up-and-coming part of CPC programming circles and scrolling messages, or "scrollies" to the initiated, are a main part of any programmer's work. In its usual form, the message scrolls across the screen, usually moving horizontally from left to right. It will contain "greetings" to other demo writers, the programmer's opinions on the CPC world and things like details of previous and forthcoming demos.

The first task is to plan exactly what the program will do. The mainstay behind a scrolling routine is to move a section of the screen left, by shifting bytes in the screen memory, and to print the next character of the message to the right of this. This process is repeated again and again, thus creating a scrolling effect! The process must be synchronised to the display's "frame flyback" (vertical blank) to ensure smooth movement. The message we will create will be scrolled across the bottom two lines of the screen, with the message being repeated on both lines.

The most difficult aspect of writing screen routines is understanding how the screen memory is laid out. The Address in memory of the start of a row can be calculated by:

**address=&C000+(row*80)**

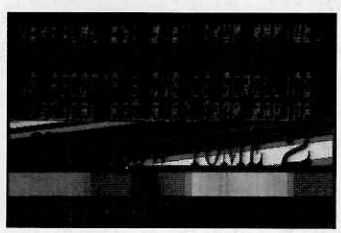
where row is the row number 0-24 (not 1-25). The equation is explained in that each screen row occupies 80 bytes, and that screen memory begins at address &C000 (49152 in decimal). This only calculates the start address of each character line, which itself comprises eight pixel lines. To address subsequent pixel lines, &800 must be added to the character line start address. The only remaining piece of information needed is that the remaining data for the line follows on sequentially from its start address.

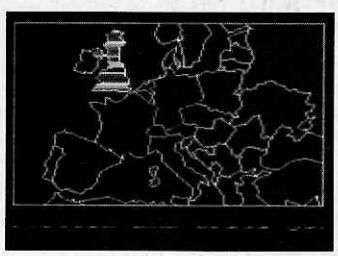
We need the addresses for the bottom two lines of the screen, i.e. lines 23 and 24 (counting from 0). Using the equation, they are calculated at &C730 and &C780 respectively. To move these lines left, we'll have to shift a block of 79 bytes - the whole line minus one byte, from, for example, &C731 to one byte left, that is &C730. This will not only have to be repeated for &C781 to &C780, but for all other pixel lines: that's &CF30 (&800 added to &C730), &D730, &DF30 etc..

Once this has been done, we can print a new character at the right of the screen. To do this, an address pointer is needed to mark where to fetch the next character to be printed. This pointer is updated each time a character is fetched. Unless you have a very large RAM expansion for your CPC - say about five megabytes - your scroll is probably going to end somewhere and return to the beginning, so a marker will need to be put in to tell the program to do this. The easiest way to do this is to put a zero byte at the end.

As for actually printing the character, for the time being we can use the firmware routine at &BB5A, which just prints a character on-screen. Later on in the series, the secrets of printing large, sprite-style letters will be revealed, but until then a simple character print will do.

We could scroll the line left by one whole MODE 1 character





If you've got any questions about demo coding, machine code programming, or if you've written a demo which you'd like to show off to the world, or if you want your demo distributed throughout Europe, write to:

DEMO Q&A
PROGRAMMING Q&A
DEMO ZONE
at:
CPC ATTACK
HHL PUBLISHING
Floor 3,
Greater London House,
Hampstead Road,
LONDON NW1
7QQ.
or FAX: 071-387 9518


(two bytes) and print a new character each frame flyback. This, however, is too fast to be readable. The solution is to scroll the line left by one byte - only half a MODE 1 character - each frame, and consequently print a new character every other frame.

Finally, on to the bits'n'bobs needed to make the demo work. The demo will have to sit somewhere in memory - &8000 will do nicely. The code also needs to set up the MODE and synchronise the scrolling to the frame flyback. Both of these functions are done using firmware calls.

**Next month, we'll be covering those infamous rasters: what are they, how are they programmed, do they require regular feeding, and can you take them out to the cinema without fear of being ostracised socially?**

# THE CODE

This is the final code. Hopefully you'll understand each instruction, but don't worry if you don't yet! The comments, which are preceded with semi-colons, don't need to be typed in.

The listing can be typed into your assembler, such as MAXAM, and assembled to produce a real, live, scrolling message - and then customised to your heart's content...

```
        org &8000               ;the code sits at &8000
        ld a,1
        call &BC0E              ;set mode 1 using the firmware
ffloop: call &BD19              ;wait for frame flyback
        ld de,&C730             ;the first line to scroll
        call scroll             ;call the scroll routine (see below)
        ld de,&C780             ;and the bottom line
        call scroll
        ld a, (toggle)          ;get the toggle value
        xor 1                   ;and toggle it between 0 and 1
        ld (toggle),a           ;store the new, toggled value
        jr z,ffloop             ;if it's 0, loop back

; you only get here after every other frame
        ld hl,(pointer)         ;get the current text pointer
        ld a,(hl)               ;and the character from it
        or a                    ;is it 0 - end of text?
        jr nz,not0              ;if not, skip this bit
        ld hl,message           ;start the text again
        ld a,(hl)               ;and get a character from it
not0:   push hl                 ;preserve HL from corruption
        push af                 ;and the same for A...
        ld hl,&2818             ;column 40, line 18
        call &BB75              ;like locate - corrupts AF,HL
        pop af                  ;restore A
        call &BB5A              ;and print the character
        push af                 ;same as above
        ld hl,2819              ;except for one line down
        call &BB75
        pop af
        call &BB5A
        pop hl                  ;restore HL!
        inc hl                  ;increase to the next position
        ld (pointer),hl         ;and store it as the pointer
        jr ffloop               ;finally, loop back!
scroll: ld a,8                  ;8 pixel lines to scroll
sloop:  ld h,d                  ;set HL (where we are moving
        ld l,e                  ;bytes from) to equal DE...
        inc hl                  ;plus one!
        ld bc,79                ;scroll 79 bytes
        ldir                    ;move the memory!
        ld bc,&7B1              ;we need to add &800 to get to
        ex de,hl                ;the next line, minus the 79
        add hl,bc               ;already added by the LDIR
        ex de,hl
        dec a                   ;have we done all the lines yet?
        jr nz,sloop             ;if not, go and do the next one!
        ret                     ;return from the subroutine
toggle: defb 0                  ;toggles between 0 and 1 to determine
                                ;whether to print a character or not
pointer: defw message           ;current text pointer value
message: defm "The CPC Attack demo column....      ",0
```