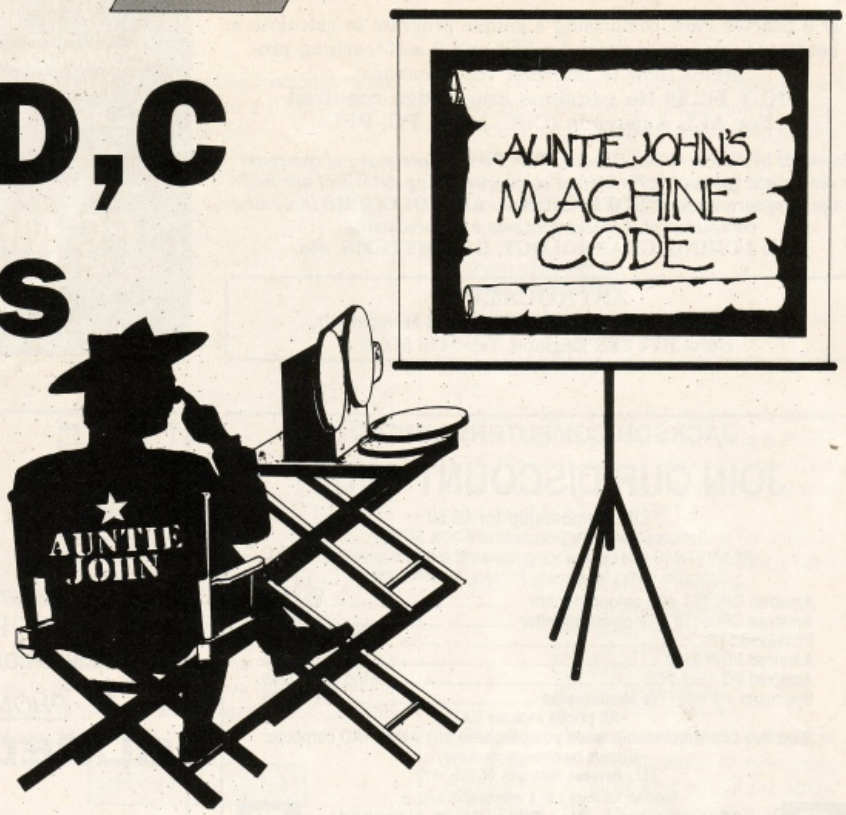
The LD,C scrolls



The screenplay continues. Cue part three of AJ's vade-mecum to the Firmware Guide

I SAID at the end of last month's column that we would look at some arithmetic. I hate arithmetic, don't you? Good, why don't we look at something else instead.

A glance through the wondrous Soft 968 – available from all good stockists – reveals quite a large section on what is called the Screen Pack. This area of firmware deals with the cassette operating system. No, only joking, it deals with the screen of course.

The Screen Pack firmware calls occur between &BBFF and &BC62. They cover everything from changing the screen mode to drawing lines, so let's have a look at some of the more interesting ones. Remember, you will need an Amsdos assembler to run the machine code programs listed hereabouts. Well, that is a lie too, because Basic programs that poke the code into memory are listed. But this is the last time!

Sorry, but my assembly listings got well and truly zapped last month. I'll kill that cat. If you would like correct copies of the original listings, please send a stamped addressed envelope to:

Auntie John's listings,
Amstrad Computer User,
169 Kings Road,
Brentwood,
Essex CM14 4EF

Panic button

The first call that merits our attention is SCR RESET (&BC02). This routine will return the screen colours and flash rates to normal.

How many times have you been experimenting with ink values and got yourself in a total mess with everything flashing bright pink and purple? No, I must admit I haven't done that either – but if I was going to, I would enter this Basic program line first:

```
KEY 128,chr$(13)+"call &bc02"+chr$(13)
```

Then, whenever I got into difficulty, all I would have to do is press function key f0 to restore normality.

Cardboard monitors

From the panic button call, we turn to the infinitely more interesting SCR SET OFFSET, which resides at &BC05, and its two friends SCR SET BASE and SCR GET LOCATION at &BC08 and &BC0B respectively. These routines tell the hardware of the computer to look at a different part of ram for the start of screen memory.

Imagine that the computer's ram is a large sheet of white paper (have a few drinks first if it helps). Then the hardware screen logic is like a piece of cardboard with a rectangular hole cut in it. When it sits on top of the white paper, what is seen through the hole – the screen memory – appears on the monitor.

Normally the cardboard is sitting right at the top of the paper, stretching from address &C000 to &FFFF, but by using the above calls we can move the cardboard cutout to a new position virtually instantaneously.

What we do is scroll the screen very quickly or shift it to a completely new area in ram to make a picture appear instantly.

This is good fun, but it's easy to get both the computer and yourself very confused; you may even lose the screen image somewhere in ram and not get anything displayed. It is, of course, totally impossible to damage the computer in any way by experimenting like this. If things get out of control, simply switching the computer off then on again will return everything to normal.

To make this a practical demonstration, first save a screen display to disc or tape. A pretty picture would be best, say a portrait of Fiona, or a view of the sun setting behind some mountains

```
org &8000           ;Start of assembly.

scr_set_base equ &bc08

ld a,&40
call scr_set_base
ret

ld a,&c0
call scr_set_base
ret

end                 ;End of assembly.
```

```
100 'Basic poker
110 MEMORY &3FFF
120 a=&4000
130 READ b$
140 IF b$="*" THEN 180
150 POKE a,VAL("&"+b$)
160 a=a+1
170 GOTO 130
180 PRINT "Code installed":END
190 DATA 3e,40,cd,08,bc,c9
200 DATA 3e,c0,cd,08,bc,c9
210 DATA *
```

Listing I: Program to shift the screen base address

from within a forest glade. What do you mean you don't have any pictures like that? You have an art package don't you?

OK. If you don't have any nice pictures, draw some squiggles with PLOT and DRAW statements, and save it with:

```
SAVE "!picture",B,&C000,&4000
```

It is very important that you use a MODE statement at the beginning of the program to generate your screen to make sure that no scrolling has taken place. If the screen is scrolled its start address is moved in memory and no longer starts at &C000, the address we have saved it from.

So we now have a screen file – all 16k of it – stored away on tape or disc. Now we need a program to shift the screen base address from its

```
org &4000           ;Start of assembly.

hw_roll equ &bc4d
wait_frame equ &bd19
encode_ink equ &bc2c

ld b,10             ;Loop counter.

 .loop
push bc
ld a,3              ;See text about
call encode_ink     ;this bit.
ld b,1              ;Scroll up.
call hw_roll
call wait_frame     ;Anti-flicker.
pop bc
djnz loop           ;B=B-1,
                    ;if B<>0 goto loop.
ret                 ;Return to Basic.
end                 ;End of assembly.
```

Listing III: Example hardware scrolling program

normal value of &C000 to the new value we are going to use of &4000.

Listing I is the assembler to do this. Notice that it is not assembled between &4000 and &7FFF because this is where the new screen display will be.

Look carefully at the listing. There are in fact two small programs: The first will move the screen to &4000 and the second will restore it to &C000.

The way the hardware works means that the screen must be in a nice neat 16k block, so we have the following alternatives for the screen base:

a) &0000 to &3FFF
b) &4000 to &7FFF
c) &8000 to &BFFF
d) &C000 to &FFFF

Options (a) and (c) both overlap some essential operation system workspace, so we are limited in our choice of screen bases to &4000 and &C000.

The routine SCR SET BASE knows that we only have four choices, so instead of having to give it a two byte address (&4000) we only need tell it the most significant byte (&40).

Now assemble Listing I. As a present to those struggling bravely on without an assembler, a Basic program that will poke the machine code into memory is supplied. Come to think of it, in this case the program is so small that its probably easier to use the Basic poking method than use as assembler. Stop sniggering at the back. This is a rare case – get an assembler!

```
org &4000           ;Start of assembly.

scr_set_ink equ &bc32
scr_set_border equ &bc38
scr_set_flashing equ &bc3e

ld a,1
ld b,6
ld c,18
call scr_set_ink

ld bc,0
call scr_set_border

ld h,1
ld l,1
call scr_set_flashing

ret

end                 ;End of assembly.
```

Listing IV: Example ink, border and flashing program

With the machine code in memory, and HIMEM set to &3FFF or below, we can safely load in the screen display that we prepared earlier. Just load it into &4000, thus:

```
LOAD "!picture",&4000
```

Tape users can grab some well earned sleep during this process.

Right, now the exciting bit. Get rid of the Basic program (if you used it) by typing NEW, then enter, save, and run the following short program.

```
10 MODE 1
20 CALL &8000
30 IF INKEY$<>"" THEN 30
40 CALL &8006
50 END
```

Isn't it simply amazing? No funny horizontal bands, just zap! and the screen appears. If your screen seems messed up, then somewhere along the line a scroll occurred and corrupted the addresses, or you might have to use a different value in line 10, depending on which mode your picture was drawn in. If you get it wrong, things will look rather strange.

Horizontal scroll

That was changing the screen base in vast chunks. If we change it only a byte or two at a time, we can shift the screen to the left or to the right.

Listing II does this by first getting the current address of the screen base – by using SCR GET LOCATION – changing it depending on a key press and telling the hardware about the new value using SCR SET OFFSET.

Once the program has assembled (or poked in with the Basic loader, sigh) the following Basic program will demonstrate it in action.

```
10 MODE 1
20 LOCATE 10,10
30 PRINT "HELLO"
40 CALL &4000
```

The speed of the routine is much faster than you could hope for with Basic, or even using machine code to move each part of the screen individually. The only drawback is that the column that has been scrolled off the screen comes back to haunt you in a different row. This is not really a problem, but causes the nasty flickering that some games possess. So now you know.

Vertical scroll

The firmware gives you two routines that will scroll the screen vertically. The first is a direct

```
org &4000           ;Start of assembly.

test_key equ &bb1e
set_offset equ &bc05
get_location equ &bc0b
wait_flyback equ &bd19

 .loop
ld a,8              ;Test for left
call test_key       ;cursor key.
call nz,moveleft    ;If pressed
                    ;call moveleft.
ld a,1              ;Test for right
call test_key       ;cursor key.
call nz,moveright   ;if pressed
                    ;call moveright.
call wait_flyback   ;To reduce flicker.
ld a,66             ;Test for ESC.
call test_key
jr z,loop
ret                 ;Return to Basic

 .moveleft
call get_location   ;Get current addr.
inc hl
inc hl              ;Change it.
call set_offset     ;Inform hardware.
ret

 .moveright
call get_location   ;Get current addr.
dec hl
dec hl              ;Change it.
call set_offset     ;Inform hardware.
ret

end                 ;End of assembly.
```

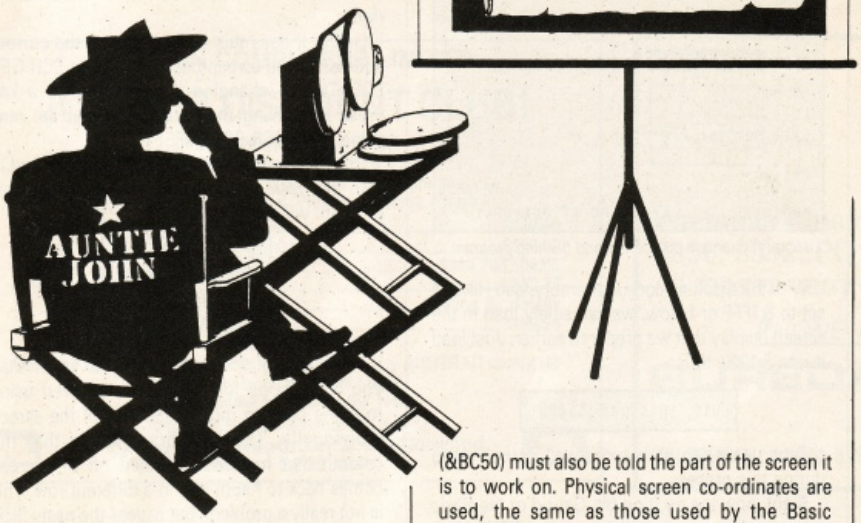
```
100 'Basic poker
110 MEMORY &3FFF
120 a=&4000
130 READ b$
140 IF b$="*" THEN 180
150 POKE a,VAL("&"+b$)
160 a=a+1
170 GOTO 130
180 PRINT "Code installed":END
190 DATA 3e,08,cd,1e,bb,c4,1b,40,3e,01
200 DATA cd,1e,bb,c4,24,40,cd,19,bd,3e
210 DATA 42,cd,1e,bb,28,e6,c9,cd,0b,bc
220 DATA 23,23 cd,05,bc,c9,cd,0b,bc,2b
230 DATA 2b,cd,05,bc,c9
240 DATA *
```

Listing II: Shifting the screen to the left or the right

equivalent of the horizontal scrolling program we have developed, SCR HW ROLL at &BC4D. The routine handles both up and down scrolls by looking at the B register. To scroll down, B must be zero and to scroll up it can be any other value. Listing III shows an example of how to scroll the screen up 10 times.

There are several points to look out for here:

(1) We must preserve the contents of the BC register pair because we use it twice – once as a loop counter, and once to control the scrolling direction.



(2) We must set the A register before calling the routine. This value determines what colour the new line scrolled on to the screen will be.

It is not simply the ink colour, but a specially encoded value. For the computer to draw a pixel on the screen it must calculate a value to represent the colour of that pixel in the screen ram. This is a bit tricky; it depends on the ink and the current screen mode. But there is a firmware routine called SCR INK ENCODE at &BC2C that, given an uncoded (normal) ink value, will return the encoded value. So to make the new line appear in INK 3, we load A with 3, call the encoding routine, then carry on as normal.

The second scrolling routine in the firmware, SCR SW ROLL, is done totally by software. It isn't nearly as fast or smooth. However, it does allow us to scroll any one section of the screen at a time.

SCR SW ROLL	&BC50
A	encoded ink
B	0 or non-zero (scroll down or up)
H	leftmost column to be scrolled
D	rightmost column to be scrolled
L	top row to be scrolled
E	bottom row to be scrolled

Figure I: The registers and parameters needed

Again, B controls the direction of the scroll and A the colour of the new line, but SCR SW ROLL (&BC50) must also be told the part of the screen it is to work on. Physical screen co-ordinates are used, the same as those used by the Basic LOCATE command but with one subtracted from them. For example a Mode 1 screen has top left physical co-ords of 0,0 and bottom right co-ords of 39,24.

The registers must be supplied with data before the routine is called. See Figure I.

Setting modes

The firmware routines SET MODE and GET MODE are the machine code equivalent of the Basic MODE command. The parameters are passed in the A register. For example, to make the computer run in 80-column mode:

```
set_mode equ &bc0e
ld a,2
call set_mode
```

Setting inks

SCR SET INK (&BC32) is like the Basic INK command, and takes its parameters in the A and BC registers: A = ink number, B = first colour, C = second colour.

SCR SET BORDER controls the border colours and uses the A, B and C registers in the same way as SCR SET INK. Remember that the colours can be both the same or they can be different, in which case they will alternate.

SCR SET FLASHING sets the flashing rate. It uses the H and L registers to hold the flash rates, the same as those issued from Basic in a SPEED INK command.

Listing IV sets INK 1 to flash between bright red and bright blue, the border to black, and the flash to a brain-numbingly fast rate.

Not a very difficult program to understand is it? This is how you would set up the screen and colours at the start of your machine code masterpiece.

```
org &4000           ;Start of assembly.

ink_encode equ &bc2c
fill_box equ &bc44

ld a,1
call ink_encode

ld h,5
ld d,34
ld l,5
ld e,19
call fill_box

ret

end                 ;End of assembly.
```

Listing V: Example box filling program

A very fulfilling routine

Just when you think that you couldn't get any more excited with the screen pack firmware calls, Amstrad has included two that will fill rectangles anywhere on the screen for you. Although not as flexible as Basic 1.1's FILL command, the box fills can come in very handy if you want a box drawn in a hurry.

There are two routines for you to play with – one which uses a screen address and one which uses the more user-friendly LOCATE co-ordinate system. Guess which one we're going to look at? Yep, the easy one.

SCR FILL BOX at &BC44 needs to be told five things; what colour the box is going to be, and where the four sides are going to be. Again, physical co-ordinates are used, and they are laid out in exactly the same way that SCR SW ROLL has its data presented to it, except that HL and DE describe the boundaries of the box to be filled instead of the boundaries of the box to be scrolled. See Figure I again.

The more observant of you may have noticed the use of encoded inks again. So to fill a box in INK 1 in the middle of a MODE 1 screen, you would use the code in Listing V. This routine is quite fast, and should come in handy for setting up title pages and the like.

Well, we are getting near to the end of the screen pack. There are some routines to return the address in memory of a given pixel with a given character position. Vertical and horizontal lines are also handled from within the screen pack giving you the opportunity to rewrite the standard DRAW routines. I would rather eat an entire collection of Peterborough telephone directories than attempt this though.

Next time we will look at some more firmware calls, and might even do something useful with them. Bye for now.