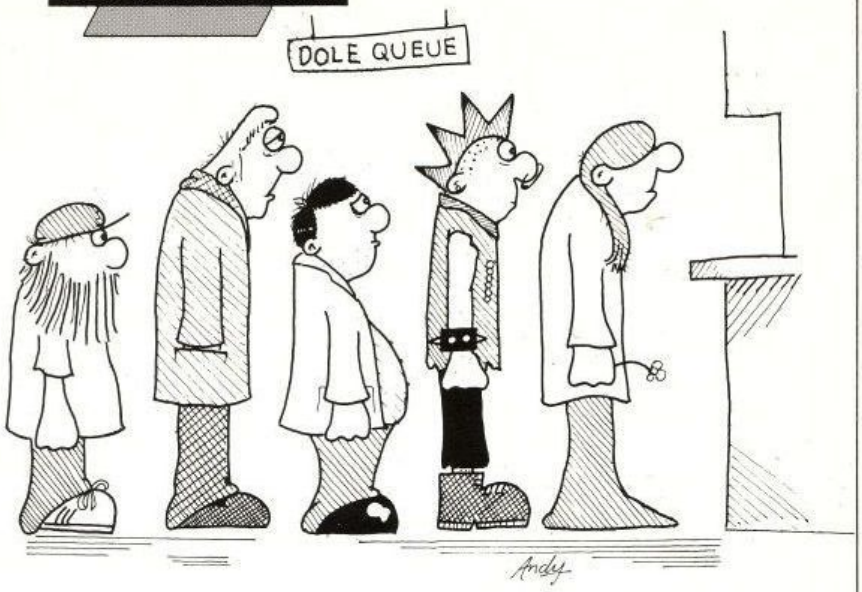
Room with a view

Auntie John shows the girl of his dreams how to do it on the CPC screen



AUNTIE John looked around the crowed room. It was worse than he had expected. And he had expected something that was pretty bad. The room was stifling hot. Condensation trickled down the windows. Shirts were a nasty shade of yellow at the armpits. People stood in queues, staring at the back of the person's head in front, trying not to notice the dandruff.

AJ sighed and joined the shortest queue, casting his mind back to the night before, which he had spent slumped over his computer.

He caught himself examining the head of the person in front of him with considerable interest. The top of it just came up to his line of sight. It was covered with long, auburn hair and there was not a trace of dandruff. The owner turned round, recognition twinkling in her eyes.

"You're Auntie John", she cried. "I don't suppose you could tell me how the CPC screen memory is laid out could you?"

AJ gulped with surprise. He hadn't been expecting this. He took a deep breath and smiled.

"Why of course. But this queue isn't moving very fast, why don't we find somewhere to sit down?"

"Good idea", she said. "There's a little cafe just around the corner, let's go there".

"Fine", said AJ, and led the way to the exit. He paused, holding the door open for her. She smiled and walked through.

Line	Address				
0	C000	C001	C002	*up to*	C04F
1	C800	C801	C802	*up to*	C84F
2	D000	D001	D002	*up to*	D04F
3	D800	D801	D802	*up to*	D84F
4	E000	E001	E002	*up to*	E04F
5	E800	E801	E802	*up to*	E84F
6	F000	F001	F002	*up to*	F04F
7	F800	F801	F802	*up to*	F84F
8	C050	C051	C052	*up to*	C09F
9	C850	C851	C852	*up to*	C89F
:	:	:	:	:	:
:	:	:	:	:	:
198	F780	F781	F782	*up to*	F7CF
199	FF80	FF81	FF82	*up to*	FFCF

Figure I: The CPC screen memory (addresses are in hexadecimal)

Pass the source

A short walk later they were facing each other across a red, checked table cloth, with two cups of steaming coffee beside them.

"The problem is this", she started. "I want to be able to print large graphics anywhere on the screen, but Basic is too slow and the firmware isn't much better. I've been thinking of a look-up table approach, but I'm not sure how to go about it".

"It's not too difficult once you've sussed how the screen data is laid out. Here look at this". AJ produced a pen from his jacket and proceeded to write out a section of the screen memory on a table napkin (see Figure I).

"The screen memory takes up a whole 16k. It's not arranged in a simple sequential pattern, but is influenced by the character positions. The screen is easiest to understand if you split it up into its 200 horizontal lines, with each line starting with a certain address and continuing for 80 bytes. You may notice that 200 x 80 is not exactly 16k (16,384), but 16,000".

"Oh yes", said the girl, "that must have something to do with scrolling the screen, musn't it?"

"Yes exactly", continued AJ. "Immediately after a MODE statement the screen is laid out in the way I've drawn on the napkin. But when the screen scrolls, the addresses change and some memory that wasn't previously visible suddenly appears. It's an interesting effect, and worth studying, but let's keep things simple."

He took a slurp of coffee and carried on. "Since each of the 200 horizontal lines has a unique address, and any address on that line can be easily calculated by adding a simple offset, it would make sense to put each of these addresses in a look-up table".

"I see", said the girl, "but hadn't we better get back to the DHSS? They'll be closing for lunch soon". AJ agreed.

```
; Input
;
; E register contains Y-coordinate.
;
; Output

; BC register contains screen addr.
;
; Note
;
; A look-up table of the 200
; possible addresses is required.

ld hl,address_of_table

ld d,0          ;Ensure D will not
                ;effect addition.
sla e           ;Multiply E by two.
add hl,de       ;HL now is addr of addr.
ld a,(hl)       ;Get LSB of addr,
ld c,a          ;and store it in C.
inc hl          ;Point HL to MSB.
ld a,(hl)       ;Get MSB of address,
ld b,a          ;and store it in B.

end             ;BC contains scr addr.
```

Figure II: Getting the screen address from a look-up table

Benchmarks

By the time they got back to the DHSS it was almost deserted, so they signed their respective names and left within the space of several minutes. They continued their conversation while walking through the park.

"So how would I use the look-up table?" she inquired.

"Hmm. That requires some arithmetic. To clarify the situation, what we need is a routine that when given an X coordinate and a Y coordinate, returns the screen address for that point. The coordinate specifies which of the addresses stored in our look-up table we need to access and the X coordinate determines the offset to be added to it".

"Each address in the look-up table uses two bytes, doesn't it?" said the girl. "So if we took the Y-coordinate, which would be in the range zero to 199, multiplied it by two and added it to the start address of the look-up table, we would get the

screen address, wouldn't we?"

"We would get the *address* of the screen address," corrected AJ. "We would need to peek the first address to get the second. The first address is like the number of a house, the second address is contained within the house". He wrote a short program on an income support form he had been given in the DHSS (see Figure II).

"Oh, right", she said, "and then adding the X coordinate is just a simple ADD instruction?"

"Yes, but it must be a 16 bit ADD. Although the X coordinate offset is only 8 bit, the result is a full 16". (see Figure III).

```
; Input
;
; BC contains screen address.
; L contains X-coordinate.
;
; Output
;
; HL contains updated screen addr.

ld h,0          ;ensure H will not
                ;effect result.
add hl,de

end     ;HL contains new scr addr.
```

Figure III: Adding the X offset to the screen address

"So now we have the address, how do we use it to print something?"

"A simple loop-within-a-loop will print a graphic of any given size. What is more difficult to calculate is the way that each pixel is represented on the screen. The mode the screen is in determines how each byte on the screen is treated – either as a colour code several bits long, or as a simple on/off flag".

"Ahh", she said, "that must be why different modes have different resolutions and different numbers of colours. The memory used stays constant – it is how it is treated that changes."

"Yes, that's right. In Mode 2, which is the highest resolution mode, each byte on the screen controls eight pixels. Each bit in the byte is either set to one, to indicate that the pixel is drawn in foreground colour, or reset to zero, to indicate background colour".

"How about Mode 1 then?" she asked, laughing as his face screwed up.

"Yeuch. It's awful", he said, "the bits are all over the place. Each byte controls four pixels. This means that each pixel has two bits controlling it, so four colours are possible. Mode 0 is similar, but two pixels are controlled by each byte, allowing 16 colours".

Mode 2

```
Pixel * * * * * * * *
Bit   7 6 5 4 3 2 1 0
```

Example: Only the end pixels are on. byte value is 129 (or 128+1).

Mode 1

```
Pixel *   *   *   *
Bit   3,7 2,6 1,5 0,4
```

Example: The left-most pixel only, in ink number 2. byte value 8.

Mode 0

```
Pixel *           *
Bit   1,5,3,7  0,4,2,6
```

Example: The rightmost pixel only, with ink number 9. byte value 65.

Figure IV: Modes and pixels. The bit number describes the power of two that the bit represents in the byte. For example, bit 0 has a value of 1, bit 1 has a value of 2. They can be expressed as "two to the power of n" where n is the bit number

He made another drawing, indicating how each bit related to each pixel for each of the modes (see Figure IV).

"Working out the data for, say, a space invader involves lots of squared paper and a calculator. It's not very exciting".

"Couldn't you peek the screen?" she asked. "Why not draw the space invader directly on the screen with an art program, or PLOT and DRAW it, and then peek each address in turn, noting the values?"

"Yes, I suppose you could do that. You could even write a program in Basic to PEEK the screen and produce a data file that could be loaded into your machine code later. Yes, that's a good idea".

"You still haven't told me how to go about putting the data on the screen", she said.

"It's just a matter of linking the routines that return the screen address for a given coordinate with two loops. The outer loop controls which horizontal line is to be poked into memory, and the inner loop does the actual poking. Like this..." He produced an example (see Figure V).

```
for down=1 to 10
get scrn addr
for across=1 to 4
read graphics data from table
poke graphics data into scrn addr
update scrn addr
next across
next down
```

Figure V: Poking data into memory. In a high level pseudo-code the routine is as above, assuming the graphic is 10 lines down by 4 bytes across

"Yes, that's simple enough", she said. "I could write a program to do that. As an input it has an X coordinate, a Y coordinate and the address of the graphics data. It also needs to be told how big the graphics data is, I mean how many bytes across and how many down" (see Figure VI).

"Yeah. This simple program has its limitations though. For a start it is not pixel accurate, but byte accurate. Also it will obliterate anything underneath it – proper sprites will preserve what is under them, and reprint it. But routines like that can be added to your program without much hassle. Even XORing the screen data with the graphics data will produce some good results".

"Oh yes, that's because XORing the data twice will produce a result that is equal to whatever was there before you started. Like 42 XOR 12 is 38, and 38 XOR 12 is 42. Clever. Well, I'd better be getting back home".

"Hold on!" said AJ. "You haven't even told me your name."

"My name?" said the girl. "Oh that's easy, it's..."

"...John! Are you up yet? You'll be late!"

AJ rolled over and blinked his eyes. It was his mother calling him.

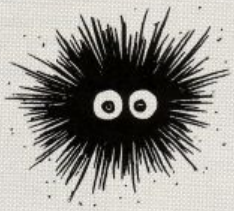
He sighed.

```
; The data in ".graphics_data" is a character sized block
; that will be drawn in ink three if Mode 1 is used.
;
; Input
;
; D contains the X-coordinate (from 0 to 79).
; E contains the Y-coordinate (from 0 to 199).
; HL points to a graphics table. The first two bytes in the
; table are the width and height of the graphics shape.
ld d,40                 ;Some example data.
ld e,50
ld hl,graphics_table

 .print_routine
ld a,(hl)               ;The height and width of
ld b,a                  ;the graphics character is
inc hl                  ;read from the table and
ld a,(hl)               ;stored in the BC register
ld c,a                  ;pair.
inc hl
;
 .loopa                 ;This is the outer or "down" loop.
push bc
ld b,c
push de
push bc
push hl
;
ld c,d                  ;This module calculates the
```

Figure VI: The finished program

```
ld b,0                  ;screen address for coordinates
ld d,0                  ;stored in the D and E registers.
sla e
rl d
ld hl,table
add hl,de
ld a,(hl)
inc hl
ld d,a
ld a,(hl)
ld h,a
ld l,d
add hl,bc
;
; At this point, DE contains the graphics data,
; and HL the screen address. The data is read into
; the A register and then poked into memory.
; The loop ".loopb" performs this action until
; the entire line of data has been poked in this way.
; This is the inner or "across" loop.
;
pop de
pop bc
 .loopb
ld a,(de)
ld (hl),a
inc hl:inc de
djnz loopb
;
pop hl
ex de,hl
inc e
pop bc
```



```
djnz loopa
ret
 .table
dw &c000,&c800,&d000,&d800,&e000,&e800,&f000,&f800
dw &c050,&c850,&d050,&d850,&e050,&e850,&f050,&f850
dw &c0a0,&c8a0,&d0a0,&d8a0,&e0a0,&e8a0,&f0a0,&f8a0
dw &c0f0,&c8f0,&d0f0,&d8f0,&e0f0,&e8f0,&f0f0,&f8f0
dw &c140,&c940,&d140,&d940,&e140,&e940,&f140,&f940
dw &c190,&c990,&d190,&d990,&e190,&e990,&f190,&f990
dw &c1e0,&c9e0,&d1e0,&d9e0,&e1e0,&e9e0,&f1e0,&f9e0
dw &c230,&ca30,&d230,&da30,&e230,&ea30,&f230,&fa30
dw &c280,&ca80,&d280,&da80,&e280,&ea80,&f280,&fa80
dw &c2d0,&cad0,&d2d0,&dad0,&e2d0,&ead0,&f2d0,&fad0
dw &c320,&cb20,&d320,&db20,&e320,&eb20,&f320,&fb20
dw &c370,&cb70,&d370,&db70,&e370,&eb70,&f370,&fb70
dw &c3c0,&cbc0,&d3c0,&dbc0,&e3c0,&ebc0,&f3c0,&fbc0
dw &c410,&cc10,&d410,&dc10,&e410,&ec10,&f410,&fc10
dw &c460,&cc60,&d460,&dc60,&e460,&ec60,&f460,&fc60
dw &c4b0,&ccb0,&d4b0,&dcb0,&e4b0,&ecb0,&f4b0,&fcb0
dw &c500,&cd00,&d500,&dd00,&e500,&ed00,&f500,&fd00
dw &c550,&cd50,&d550,&dd50,&e550,&ed50,&f550,&fd50
dw &c5a0,&cda0,&d5a0,&dda0,&e5a0,&eda0,&f5a0,&fda0
dw &c5f0,&cdf0,&d5f0,&ddf0,&e5f0,&edf0,&f5f0,&fdf0
dw &c640,&ce40,&d640,&de40,&e640,&ee40,&f640,&fe40
dw &c690,&ce90,&d690,&de90,&e690,&ee90,&f690,&fe90
dw &c6e0,&cee0,&d6e0,&dee0,&e6e0,&eee0,&f6e0,&fee0
dw &c730,&cf30,&d730,&df30,&e730,&ef30,&f730,&ff30
dw &c780,&cf80,&d780,&df80,&e780,&ef80,&f780,&ff80

 .graphics_table
db 8,2
db 255,255,255,255,255,255,255,255
db 255,255,255,255,255,255,255,255
```