CHAPTER NINE

SCREEN DUMPS

Modern dot-matrix printers can produce high-resolution graphics far in excess of the maximum resolution of the BBC Micro. It would seem sensible then to have a means of making a 'hard copy' of the entire contents of the screen. Such a copy of the screen is called a SCREEN DUMP.

There are a number of screen dump programs around for various printers; however, they all tend to be slow. Many of them claim to be able to handle dumps of any size and of any mode with one program. This is fine, but to get a program to do this means sacrificing speed and performance. In this chapter we are going to look at a few examples of specialised screen dump programs. The advantage is that the top speed of the printer can be used.

The programs in this chapter are designed to be used with an Epson dot-matrix printer such as the FX80 or RX80. The MX range of printers will not handle the very-high-resolution dumps, but will handle, with only very slight modifications, the first few programs in this chapter. It should be possible to adapt some of the programs to run on other makes of printer. Before reading on, you might like to read the relevant section of your printer manual.

A simple BASIC dump

The first thing we should look at is a simple example − a Mode 4 screen dump about 4.5in by 3.5in.

We can consider the printer head as a vertical column of eight dots. The head is moved from left to right across the paper to produce eight rows of dots which make up a horizontal band. To print this band of dots, we need to send a series of codes to the printer to tell it what vertical spacing between each

band to use and how many dots make up a horizontal row. Then we send one byte for each vertical column of eight dots where each bit of that byte represents a dot (bit 7 is at the top, bit 0 is at the bottom). But this immediately presents us with a problem as the Mode 4 screen is stored with each byte representing a HORIZONTAL row of eight pixels (one bit represents one pixel in Mode 4).

To solve this, let's consider how we would dump one character (eight by eight pixels) to the printer. Let's assume that the character is placed at the top left-hand corner of the screen. We can use a BASIC program for this as it is only an experiment.

First we need to go into Mode 4 and turn off the cursor. Next we need to print the character (say, an A) at the top left-hand corner of the screen.

```
10 MODE4
20 VDU23,1,0;0;0;0;
30 P."A"
```

Now we must set up the printer. The first thing is that we don't want the printer to automatically print a line feed after every carriage return. This is because line-feed feeds the paper upwards by more than eight dots and we want each band (eight dots high) to butt up against the previous one. To do this we use *FX6,10 to disable the printer line-feed. Note that some printers are set up so that they automatically print a line-feed whenever a carriage return is sent. If your printer is set up to do this then the setting must be altered. This is done on Epson printers by changing the position of a small switch inside the printer. The printer manual gives details on how to do this. If you don't do this then the printer will split up your screen dumps into bands with gaps between them.

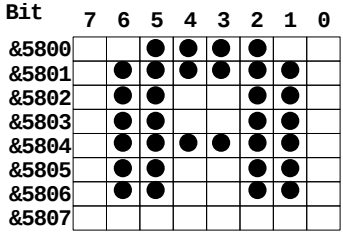
We then need to turn the printer on with VDU2. While we are doing this we can also tell the printer to go into a graphics mode where dots are printed horizontally 1/80 of an inch apart. As the dots are automatically printer 1/72 of an inch apart vertically in all graphics modes this will make our eight-by-eight pixel character come out approximately

square. We also need to tell the printer that we are going to send it eight bytes of graphics. The codes to do this are 27,42,4,8,0. Here, the first three numbers put the printer into the desired graphics mode and the last two are the number of bytes of graphics that we are sending to it. Remember that it is up to you to find out the correct control sequences for your make, model and mark of printer. To resume:

```
40 *FX6,10
50 VDU2,1,27,1,42,1,4,1,8,1,0
```

Notice that we use VDU1,X so that the data doesn't appear on the screen. Now we are ready to dump the character. Our letter A is stored on the screen as eight horizontal bytes (stored in addresses &5800 to &5807).

The pattern for A



To suit the printer, we need to code the A into eight vertical bytes. To do this we must set up a bit mask which, using AND, will first mask out everything but bit 7, the leftmost bit, of each byte, then bit 6, and so on until all eight columns have been printed. We can store the current value of this mask in the variable A% − the initial value must be 128 to mask out everything but bit 7. We can use this mask on each of the eight bytes of the character in turn, to extract the leftmost bits, making these into a vertical byte to send to the printer. Then we can set the mask to 64 to extract the next column, and so on until all eight columns have been printed.

For each column we first set a variable, say B%, to zero. We then check the relevant bit of &5800 using the mask. If it is set then we add 1 to B%. We can then check the relevant bit of &5801, and so on.

We want the top bit of the column (from &5800) to be stored in bit 7 or B%, and so on. If, before each byte is checked with the mask, we shift B% left one bit by multiplying it by two, and then we add 1 if the bit is set, then, after all the eight bytes have been checked the first bit we extracted (from &5800) will be in bit 7; the next bit (from &5801) will be in bit 6; and so on.

When we have done this to get the first column, we send B% to the printer and divide A% by two to move the mask right one pixel for the next column. We go on doing this until A% has reached zero and we have thus printed eight columns. Finally, we must send a carriage return to the printer and turn the printer off.

```
 45 P%=&5800
 60 A%=128
 70 B%=0
 80 FORY%=0TO7
 90 B%=B%*2
100 IFP%?Y%ANDA%THENB%=B%+1
110 NEXT
120 VDU1,B%
130 A%=A%/2
140 IFA%>0THEN70
150 VDU1,13,3
```

This will dump one character from the screen. Notice that, because we have used a separate variable to P% to store the address of the first byte of the character (line 45), we can print any character anywhere on the screen by simply changing P%. So, to print a whole band (which corresponds to a line of text on the screen) we need only alter the initial setting of the printer to say that we are going to send 40 characters of eight bytes each, or &140 bytes (line 50 below), add eight to P% after each character has been sent, and repeat this 40 times.

```
 50 VDU2,1,27,1,42,1,4,1,&40,1,&1
 55 FORX%=1TO40
143 P%=P%+8
147 NEXT
```

To make a complete screen dump we now need to feed the paper upwards by eight dots (each dot takes up 1/72 of an inch vertically) to print the next band. The Epson printers provide the command that will feed the paper upwards by n/216 of an inch. We want to feed by 8/72 so n has to be 24. Because of the way the screen is laid out, P% already points to the start of the next band when we get to line 150; so, if P% is less than &8000 (the end of the screen), then we can go back to line 50 and print the next band down.

```
150 VDU1,13,1,27,1,74,1,24
160 IFP%<&8000THEN50
170 VDU3
```

This program can then be added to an existing program, or a screen can be *SAVEd on disc and this routine can be used as a separate dump program by *LOADing the screen at the beginning.

A machine code equivalent

The next thing we need to do is to convert this BASIC into machine code. As in previous chapters, the above program has been purposely written to code into machine code easily.

The first thing we should do in the machine code version is disable all interrupts. This will help to speed up the program slightly. We can then disable the line feeds.

```
10000 DEFPROCass
10010 DIMmc%150
10020 oswrch=&FFEE
10030 osbyte=&FFF4
10040 FORpass%=0TO2STEP2
10050 P%=mc%
10060 [OPTpass%
10070 .dump    SEI
10080          LDA #6
10090          LDX #10
10100          LDY #0
10110          JSR osbyte
```

Before we carry on any further, we should notice

that almost every VDU command in the BASIC program was in the form VDU1,X. To save space it would be sensible to write a subroutine which outputs first a 1 then the byte we want to send to the printer.

```
20000 .out      PHA
20010           LDA #1
20020           JSR oswrch
20030           PLA
20040           JMP oswrch
```

This routine should be entered with the byte to be sent to the printer in the accumulator.

Going back to the main routine, we can set up P% in &70 and &71 and turn on the printer.

```
10120           LDA #0
10130           STA &70
10140           LDA #&58
10150           STA &71
10160           LDA #2
10170           JSR oswrch
```

Now we must initialise the printer for each band.

```
10180 .band     LDA #27
10190           JSR out
10200           LDA #42
10210           JSR out
10220           LDA #4
10230           JSR out
10240           LDA #&40
10250           JSR out
10260           LDA #&1
10270           JSR out
```

We can use the X register to count the 40 characters that make up a horizontal band. We can use &72 to store the bit mask, A%; &73 to store the byte sent to the printer, B%; and the Y register for the loop, Y%.

```
10280           LDX #40
10290 .char     LDA #128
10300           STA &72
```

```
10310 .column  LDA #0
10320          STA &73
10330          LDY #0
```

Next we need to shift &73 (B%) left. Then we must load a byte from the screen and mask it with &72. If there is a pixel in this position then we must set bit zero of &73. Then we can repeat the loop until Y is eight.

```
10340 .pixel   ASL &73
10350          LDA (&70),Y
10360          AND &72
10370          BEQ not
10380          LDA &73
10390          ORA #1
10400          STA &73
10410 .not     INY
10420          CPY #8
10430          BNE pixel
```

We can then send the column of eight pixels stored at &73 to the pointer.

```
10440          LDA &73
10450          JSR out
```

Next we must shift the mask right one bit and repeat until we have dumped eight columns or a complete text character.

```
10460          LSR &72
10470          BNE column
```

Now we must add eight to &70 and &71 for the next character position and repeat back to the character routine 40 times to print one complete band.

```
10480          LDA &70
10490          CLC
10500          ADC #8
10510          STA &70
10520          BCC skip
10530          INC &71
10540 .skip    DEX
```

```
10550           BNE char
```

Lastly we need to feed the paper upwards and repeat the band routine until the content of &71 is &80 or bigger. Then we can turn the printer off, enable interrupts and end the subroutine.

```
10560           LDA #13
10570           JSR out
10580           LDA #27
10590           JSR out
10600           LDA #74
10610           JSR out
10620           LDA #24
10630           JSR out
10640           LDA &71
10650           BPL band
10660           LDA #3
10670           JSR oswrch
10680           CLI
10690           RTS
10750 ]
10760 NEXT
10770 ENDPROC
```

And that's it.

As an example, try adding the following lines at the beginning of the assembly code:

```
  10  MODE4
  20  VDU23,1,0;0;0;0;29,640;512;
  30  PROCass
  40  S=1.1
  50  P%=4
  60  FORA=0TOPI*21STEPPI/20
  70  PLOTP%,636*SIN(A),508*COS(A*S)
  80  P%=5
  90  NEXT
 100  CALLdump
 110  END
```

Mode 4 is very similar to Mode 0. To convert this program to run in Mode 0 only six lines need to be changed.

```
   10 MODE 0
10140          LDA #&30
10220          LDA #1
10240          LDA #&80
10260          LDA #2
10280          LDX #80
```

Notice that the Epson does not provide a graphics mode that has 160 pixels to the inch horizontally, so we have to use the closest mode; this gives only 120 pixels to the inch. This has the effect of stretching the screen dump slightly horizontally.

A colour-as-tone dump

So far we have only looked at two-colour dumps. What happens if we want to dump a 16-colour, Mode 2 screen? Most people don't own expensive colour printers, so let's find a way of representing colours on a black-and-white printer. It is easiest to do this with shades. Unfortunately, a dot-matrix printer won't automatically print greys for us. To get the illusion of shades we need to print a pattern of dots with more dots per square inch for the darker colours than for the lighter ones. For dumping in Mode 2 we will first ignore flashing colours and then represent each pixel by an imaginary box containing 12 dots arranged as two rows (height) of six columns (width). This arrangement is chosen because the pixels in Mode 2 are not square but rectangular − they are thin horizontal dashes rather than dots − so the printer must represent each pixel by a pattern of dots that is rectangular.

The pattern of dots for each colour (chosen to look as close to a solid block of colour as possible) is as follows:

This means that, as a Mode 2 screen is 160 pixels by 256 pixels, we will dump 960 (=6*160) by 512 (=2*256) dots on the printer.

Before we can do anything, we must know how a Mode 2 screen is laid out in the memory. As Mode 2 is a 16-colour mode, the computer must use four bits to store the colour of each pixel on the screen. Thus it can store two pixels (2x4=8 bits) in one byte. If there are 40960 (160x256) pixels on the screen,

Black
Blue
Red
Magenta
Green
Cyan
Yellow
White

'Colour dot patterns'

this means that to store the complete screen we need 20480 bytes or 20K.

The two pixels stored in each byte are always adjacent to each other and appear in a horizontal line on the screen. We would expect to find that the first 80 consecutive bytes (=160 pixels) of the screen memory make up the top row of pixels. However, this unfortunately is not so. The screen memory is organised in terms of character positions. One character is made up of 64 (8x8) pixels. This means that 32 bytes of screen memory are needed to make up each character. Horizontally, these are split up into four columns, each two pixels (or one byte) wide. Each column is then made up vertically of eight consecutive bytes.

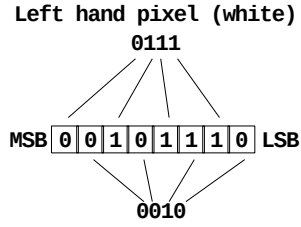
0	8	16	24
1	9	17	25
2	10	18	26
3	11	19	27
4	12	20	28
5	13	21	29
6	14	22	30
7	15	23	31

The order of the bytes that make up one text character.

The set of 32-byte characters is laid out, as would be expected, in 32 rows of 20 characters. So, for our purposes, we can look at the screen as divided into 32 rows. Each row is stored as 640 bytes and consists of 80 columns, each column being eight bytes tall. The first byte of the screen is stored at &3000 and the last at &7FFF. Of course, all this changes for different modes.

&3000	&3008		&3270	&3278
&3001	&3009		&3271	&3279
&3002	&300A		&3272	&327A
&3003	&300B		&3273	&327B
&3004	&300C		&3274	&327C
&3005	&300D		&3275	&327D
&3006	&300E		&3276	&327E
&3007	&300F		&3277	&327F
&3280	&3288		&34F0	&34F8
&3281	&3289		&34F1	&34F9
&3282	&328A		&34F2	&34FA
&3283	&328B		&34F3	&34FB
&3284	&328C		&34F4	&34FC
&3285	&328D		&34F5	&34FD
&3286	&328E		&34F6	&34FE
&3287	&328F		&34F7	&34FF

Finally, we must examine how the two Mode 2 pixels are stored in each screen memory byte. Again, we would expect the most significant four bits of the byte to hold the colour of the first pixel and the least significant four bits to hold the colour of the second pixel. Yet again, things are slightly more complex! The four two-bit numbers which represent the colours of the two pixels are ‘interleaved’ − first a bit from pixel one (the left-hand pixel) then a bit from pixel two (the right-hand pixel) then a bit from pixel one again, and so on.



Mode 2 pixels: layout

It is obvious from this diagram that ours is not an easy task. Remember that we need to send graphics to the printer in bands of eight vertical dots (corresponding to the eight vertical dots on the printer head). This means that we need to group four screen pixels (each two dots high) above each other. Let's first of all assume that we have found the colours of each of these pixels (ignoring the most significant bit to exclude flashing colours) and we have stored the four of them, from top to bottom, in addresses &74 to &77. Now let's try and write a routine to dump these four pixels to the printer.

As often with machine code, the easiest way to do this is with a look-up table. This will contain data on the pattern of dots for each colour. However, we need to think carefully about how to arrange this information in a table. As the printer deals in vertical columns, the sensible way to split up the 12 dots per pixel is into six columns, devoting one byte of table for each column. Although we are wasting memory by only using two bits in each byte of the table, it will speed up the program if we use one byte of table for each column.

We now have to decide how to order these columns. The obvious way is to store them as six bytes for black followed by six bytes for red, etcetera. However, we must consider how we are going to address this table. To print the four vertical pixels we are going to send six bytes to the printer. The first byte will be made up of the first columns of each of the relevant colour patterns, the second byte will be made up of the second column of patterns, and so on. So, to send the first byte to the printer we are going to need access to the eight bytes that

contain the first columns of each of the eight colour patterns. Then for the second byte we will need the eight bytes of the second columns and so on. Thus it makes sense to store the eight bytes that contain the first columns of each of the colour patterns, grouped together in the table.

To make the program easier we can store the data in groups according to the colours but just read them into the table in our preferred order.

```
10000DEFPROCass
10010DIMmc%300,D%47
10020RESTORE20000
10030FORA%=0TO7
10040FORB%=0TO5
10050READD%?(A%+B%*8)
10060NEXT,

20000 DATA3,3,3,3,3,3:REM Black
20010 DATA3,2,1,3,1,2:REM Red
20020 DATA0,1,1,0,2,2:REM Green
20030 DATA0,0,1,0,0,0:REM Yellow
20040 DATA3,3,1,3,3,2:REM Blue
20050 DATA1,2,1,2,1,2:REM Magenta
20060 DATA0,1,0,0,2,0:REM Cyan
20070 DATA0,0,0,0,0,0:REM White
```

Now the routine to print four vertical pixels should be relatively easy. First we are going to need, from the previous program, the routine for sending a byte to the printer.

```
10070 oswrch=&FFEE

14000  .out     PHA
14010           LDA #1
14020           JSR oswrch
14030           PLA
14040           JMP oswrch
```

Now we can start. The first job is to set up the address of the start of the table in &7A and &7B. Then we can use post-indexed indirect addressing to look up the bytes for the first column of each pixel by setting the Y register to the colour number. Then

when we have sent this byte to the printer we can add eight to the contents of &7A and &7B so that, taken together, they point to the address of the eight second-column bytes. We can use the X register to count the number of the bytes we have sent to the printer.

```
13000 .print   LDA #D%MOD256
13010          STA &7A
13020          LDA #D%DIV256
13030          STA &7B
13040          LDX #6
```

Next we can set Y to the contents of &74 (the top pixel colour) and load in the two bits for the top two dots of the first column. These must eventually be the most significant two bits of the byte so we must shift them left. We shift left the bytes we are calculating, by two bits each time a pixel has been calculated, and so save a lot of effort:

```
13050 .prloop  LDY &74
13060          LDA (&7A),Y
13070          ASL A
13080          ASL A
13090          STA &79
13100          LDY &75
13110          LDA (&7A),Y
13120          ORA &79
13130          ASL A
13140          ASL A
13150          STA &79
13160          LDY &76
13170          LDA (&7A),Y
13180          ORA &79
13190          ASL A
13200          ASL A
13210          STA &79
13220          LDY &77
13230          LDA (&7A),Y
13240          ORA &79
13250          JSR out
```

We have now sent the first of our six bytes to the printer and all that remains is to add eight to &7A

and &7B and go back for the next byte.

```
13260          LDA &74
13270          CLC
13280          ADC #8
13290          STA &7A
13300          BCC skip2
13310          INC &7B
13320 .skip2   DEX
13330          BNE prloop
13340          RTS
```

This is all very well, but we still need to find out the four colours from the screen. We could use the operating system command that is equivalent to the BASIC function POINT, but this is slow and it is faster to work it out ourselves.

Let's now look at the main program and let's assume that we have already written a routine that will dump a whole band of Mode 2 graphics, four pixels high, given the address of the top left-hand corner of the line on the screen stored in &70 and &71.

Firstly, we need to disable interrupts and the printer line-feed, and turn the printer on.

```
10080 FORpass%=0TO2STEP2
10090 P%=mc%
10100 [OPTpass%
10110 .dump    SEI
10120          LDA #6
10130          LDX #10
10140          LDY #0
10150          JSR &FFF4
10160          LDA #2
10170          JSR oswrch
```

Next, we need to store the address of the top left-hand corner of the screen in &70 and &71. Then to print the top half of a text line we need only our BAND routine.

```
10180          LDA #0
10190          STA &70
10200          LDA #&30
```

```
10210          STA &71
10220 .textlin JSR band
```

To print the bottom half we need to add four to the start address and call LINE again.

```
10230          LDA &70
10240          CLC
10250          ADC #4
10260          STA &70
10270          BCC skip
10280          INC &71
10290 .skip    JSR band
```

To move on to the next line we need to add &280 to the address of the previous line, but we have already added four so we only need add &27C. If the answer is less than &8000 then we can go back and dump the next text line. Otherwise we must turn off the printer, re-enable the interrupts and end the routine.

```
10300          LDA &70
10310          CLC
10320          ADC #&7C
10330          STA &70
10340          LDA &71
10350          ADC #2
10360          STA &71
10370          BPL textlin
10380          LDA #3
10390          JSR oswrch
10400          CLI
10410          RTS
```

Now all we have to do is make our assumptions concrete and actually write BAND.

Firstly, we need to make a copy of &70 and &71 in &72 and &73. Then we need to initialise the printer to accept a 960-column band of graphics.

```
11000 .band    LDA &70
11010          STA &72
11020          LDA &71
11030          STA &73
```

```
11040          LDA #27
11050          JSR out
11060          LDA #42
11070          JSR out
11080          LDA #1
11090          JSR out
11100          LDA #&C0
11110          JSR out
11120          LDA #3
11130          JSR out
```

Now we need to count the 80 groups of pixels that we must send to the printer (160 pixels per line and two pixels per byte). As we need both the X and Y registers we use location &79 to hold a count.

```
11140          LDA #80
11150          STA &78
```

Before we carry on, we need two routines that will extract the colours of the two pixels from a byte. Let us deal first with the left-hand pixel stored in the odd-numbered bits. We can arrange for the three bits of interest to us to be in bits 2, 4 and 6 (least significant to most significant) of the accumulator, using ASL once; then we can do two left-shifts to transfer the most significant bit in bit 6 into the carry flag. Then we can ROL the carry flag into a memory byte previously set to zero (&79). This leaves the next bit in bit 6 of the accumulator so we can repeat this a further two times to leave our three bits in bits 0-2 of location &79. If this sounds complicated, try following the code through:

```
12000 .colour1 LDA (&72),Y
12010 .colour  ASL A
12020          INY
12030          LDX #0
12040          STX &79
12050          LDX #3
12060 .coloop  ASL A
12070          ASL A
12080          ROL &79
12090          DEX
12100          BNE coloop
```

```
12110          LDA &79
12120          RTS
```

The INY command loads the next byte down on the next call of the routine. If we want to extract the colour of the other pixel we need only load the byte into the accumulator and shift it left one bit so that the bits we want occupy the same positions as for the other pixel. Then we can jump to the label COLOUR, conveniently placed in the previous routine, to finish the job.

```
12130 .colour2 LDA (&72),Y
12140          ASL A
12150          JMP colour
```

Now we can go back to BAND where we were just about to dump a line of graphics.

Firstly, we must set Y to zero so that we are ready to load the top byte. We can then call COLOUR1 to find the colour of the left-hand pixel and store this information in &74 ready for PRINT. We can then do the same for the other four pixels and call PRINT.

```
11160 .column  LDY #0
11170          JSR colour1
11180          STA &74
11190          JSR colour1
11200          STA &75
11210          JSR colour1
11220          STA &76
11230          JSR colour1
11240          STA &77
11250          JSR print
```

We do the same for the four right-hand pixels.

```
11260          LDY #0
11270          JSR colour2
11280          STA &74
11290          JSR colour2
11300          STA &75
11310          JSR colour2
11320          STA &76
```

```
11330          JSR colour2
11340          STA &77
11350          JSR print
```

Now, to print the next set of four bytes, we need to add eight to &72 and &73 and repeat until we have printed all 80 groups of pixels.

```
11360          LDA &72
11370          CLC
11380          ADC #8
11390          STA &72
11400          BCC skip1
11410          INC &73
11420 .skip1   DEC &79
11430          BNE column
```

Lastly we need to feed the paper and return.

```
11440          LDA #13
11450          JSR out
11460          LDA #27
11470          JSR out
11480          LDA #74
11490          JSR out
11500          LDA #24
11510          JMP out

15000 ]
15010 NEXT
15020 ENDPROC
```

At last we have finished. Try the following example to check that the routine works.

```
  10 MODE 2
  20 VDU23,1,0;0;0;0;
  30 PROCass
  40 COLOUR135
  50 CLS
  60 COLOUR128
  70 RESTORE180
  80 P%=4
  90 FORA%=0TO8
 100 READB%
 110 GCOL0,B%
```

```
  120 MOVE0,0
  130 PLOTP%,1000*SIN(A%*PI/16),1000*COS(A%*PI/
16)
  140 P%=85
  150 NEXT
  160 CALLdump
  170 END
  180 DATA0,0,4,1,5,2,6,3,7
```

A miniature dump

So far, we have not printed anything particularly revolutionary. However, with many modern printers it is possible to produce screen dumps that put to shame what we have looked at so far.

Some Epson and compatable printers have a graphics mode in which 240 dots to the inch can be printed horizontally. Unfortunately, adjacent dots cannot be printed as the print heads need time to recover after each dot has been printed. To overcome this we can print each line TWICE, first printing all the even-numbered dots then all the odd-numbered dots. This still leaves us with the vertical resolution − fixed by the pacing of the print in the print head − of 72 dots to the inch. This is more difficult to overcome.

To solve this problem we use another special feature of Epson and compatible printers − they can feed the paper relatively accurately by down to 1/216 of an inch! This means that we can inter-leave three columns of dots at a spacing of 1/72 of an inch. The diagram on page 200 shows this. The pixels printed on the first pass are numbered 1, etcetera. This means that the maximum resolution of the printer is 240 dots per inch horizontally and 216 dots per inch vertically. This makes a staggering 51840 dots per square inch! At this resolution we should be able to dump a full Mode 4 screen in about 1.6 square inches − so that's what we are going to do!

Dot interleaving

```
1 2 1 2 1 2 1 2 1 2
3 4 3 4 3 4 3 4 3 4
5 6 5 6 5 6 5 6 5 6
1 2 1 2 1 2 1 2 1 2
```

```
3  4  3  4  3  4  3  4  3  4
5  6  5  6  5  6  5  6  5  6
1  2  1  2  1  2  1  2  1  2
3  4  3  4  3  4  3  4  3  4
5  6  5  6  5  6  5  6  5  6
1  2  1  2  1  2  1  2  1  2
3  4  3  4  3  4  3  4  3  4
5  6  5  6  5  6  5  6  5  6
1  2  1  2  1  2  1  2  1  2
3  4  3  4  3  4  3  4  3  4
5  6  5  6  5  6  5  6  5  6
1  2  1  2  1  2  1  2  1  2
3  4  3  4  3  4  3  4  3  4
5  6  5  6  5  6  5  6  5  6
1  2  1  2  1  2  1  2  1  2
3  4  3  4  3  4  3  4  3  4
5  6  5  6  5  6  5  6  5  6
1  2  1  2  1  2  1  2  1  2
3  4  3  4  3  4  3  4  3  4
5  6  5  6  5  6  5  6  5  6
```

It is very difficult to try and use direct screen access to dump the screen in this example (though you are welcome to try if you are feeling masochistic) so we will use OSWORD call 9 (the operating system equivalent of POINT).

The first thing we need, then, is a routine to set up a parameter block and call OSWORD 9. Let us state that the X coordinate should be stored in &70 and &71 and the Y coordinate in &72 and &73. This means that our OSWORD parameter block can be taken as starting at &70 and the colour will be returned in &74. The first job is to save the X and Y registers and set up the three registers for the call.

```
20000 .point  STX &76
20010         STY &77
20020         LDA #9
20030         LDY #0
20040         LDX #&70
20050         JSR &FFF1
```

Now we can reload the X and Y registers with their original values (saved at &76 and &77). We also need to check if the point was on the screen. This is

because there are 256 pixels vertically on the screen and we are printing in lots of 24. Now, 24 into 256 doesn't go, so the last line is going to drop off the bottom and when this happens POINT must return black. The OSWORD call we're using will return 255 so we need only check that the colour is positive, and, if not, return zero.

```
20060         LDX &76
20070         LDY &77
20080         LDA &74
20090         BPL point1
20100         LDA #0
20110 .point1 RTS
```

We also need our old faithful routine, OUT.

```
20120 .out    PHA
20130         LDA #1
20140         JSR oswrch
20150         PLA
20160         JMP oswrch
```

Now, the main routine. We reset the graphics windows and origin, as we are using POINT; otherwise, we might end up dumping the wrong part of the screen. Also, we can disable interrupts.

```
10000 DEFPROCass
10010 DIMmc%300
10020 oswrch=&FFEE
10030 FORpass%=0TO2STEP2
10040 P%=mc%
10050 [OPTpass%
10060 .dump    SEI
10070          LDA #26
10080          JSR oswrch
```

Next we must disable printer line-feeds and turn on the printer.

```
10090          LDA #6
10100          LDX #10
10110          LDY #0
10120          JSR &FFF4
```

```
10130           LDA #2
10140           JSR oswrch
```

Now we must set the Y coordinate of the top pixel of the screen in &78 and &79 ready to work down the screen.

```
10150           LDA #&FC
10160           STA &78
10170           LDA #3
10180           STA &79
```

For the moment let's assume that we already have a routine BAND that prints one pass of the print head, i.e. given the coordinates of the top left-hand pixel of a band (320 pixels by 24 pixels) it prints one sixth of the dots on that band. The X coordinate needs to be in &70 and &71 and the Y coordinate in &78 and &79. Also, we need to tell the routine whether it is printing odd-numbered or even-numbered dots, horizontally, on that pass. We can do that by supplying a mask in &7A − either 255 for the even dots or zero for the odd ones. The reason for this will be become obvious in a moment. Armed with this routine, we can finish the main routine.

The first job, as we are going to print three interlaced passes vertically, is to set the X register to count these.

```
10190 .loop1    LDX #3
```

Next, we need to set up the X coordinate as zero and the mask as 255 for the even pixels.

```
10200 .loop2    LDA #0
10210           STA &70
10220           STA &71
10230           LDA #255
10240           STA &7A
10250           JSR band
```

Next, for the odd pixels we need to set the X coordinate to four and the mask to zero.

```
10260           LDA #4
```

```
10270           STA &70
10280           LDA #0
10290           STA &71
10300           STA &7A
10310           JSR band
```

Now we must feed the paper 1/216 of an inch ready for the next interleaved set of pixels. We also need to move the Y coordinate down one pixel and then repeat the whole process.

```
10320           LDA #27
10330           JSR out
10340           LDA #74
10350           JSR out
10360           LDA #1
10370           JSR out
10380           LDA &78
10390           SEC
10400           SBC #4
10410           STA &78
10420           BCS skip3
10430           DEC &79
10440 .skip3    DEX
10450           BNE loop2
```

We have now printed a whole band 24 dots high and need to feed the paper onward 24 dots (less the three we have already fed it).

```
10460           LDA #27
10470           JSR out
10480           LDA #74
10490           JSR out
10500           LDA #21
10510           JSR out
```

We then need to move the Y coordinate down 24 pixels (less the three we have already moved it); and, if Y coordinate is still on the screen, we must go back to dump the next band.

```
10520           LDA &78
10530           SEC
10540           SBC #84
10550           STA &78
```

```
10560          LDA &79
10570          SBC #0
10580          STA &79
10590          BPL loop1
```

All that remains is to disable the printer, re-enable the interrupts again and exit.

```
10600          LDA #3
10610          JSR oswrch
10620          CLI
10630          RTS
```

Now we need to write BAND. The first thing this routine needs to do is to set up the printer to receive 320 bytes of graphics.

```
10640 .band    LDA #127
10650          JSR out
10660          LDA #42
10670          JSR out
10680          LDA #3
10690          JSR out
10700          LDA #&40
10710          JSR out
10720          LDA #1
10730          JSR out
```

Next we need to make a temporary working copy of the Y coordinate in &72 and &73 for POINT to use.

```
10740 .line1   LDA &78
10750          STA &72
10760          LDA &79
10770          STA &73
```

We now need to work out a byte. Again, we are going to use the technique of shifting a byte of memory left while setting bit one to the colour of pixel and repeating this eight times. So first we need to set the byte (&75) to zero. Then we need the Y register to count the eight times. The first job inside the loop is to shift &75 left a bit. Then we can use POINT to return the colour of the pixel. If it is zero then we can leave &75 alone as we have

already set all the bits to zero. Otherwise, we need to set bit zero to one. As this will always be zero to start with we can simply use the command INC to set it to one.

```
10780          LDA #0
10790          STA &75
10800          LDY #8
10810 .line2   ASL &75
10820          JSR point
10830          BEQ skip1
10840          INC &75
10850 .skip1
```

Next we need to move down three pixels to allow for the interleave, and repeat the process.

```
10850 .skip1   LDA &72
10860          SEC
10870          SBC #12
10880          STA &72
10890          BCS skip2
10900          DEC &73
10910 .skip2   DEY
10920          BNE line2
```

We are now ready to send a byte to the printer. However, if &7A is set to 255, then we want to send the byte followed by a zero; if it is set to zero, then we want to send a zero followed by the byte. So, as the first byte to send to the printer, we can use (&75 AND &7A). For the second byte we can use (&75 AND (&7A EOR 255)).

```
10930          LDA &7A
10940          AND &75
```

Now all that remains to do is move on to the next-pixel-but-one horizontally by adding eight to the X coordinate; and then, if the answer is less than 1280 or &500, go back and send the next pair of bytes. If the end of the line has been reached then we must send a carriage return and exit from the routine.

```
10950          JSR out
```

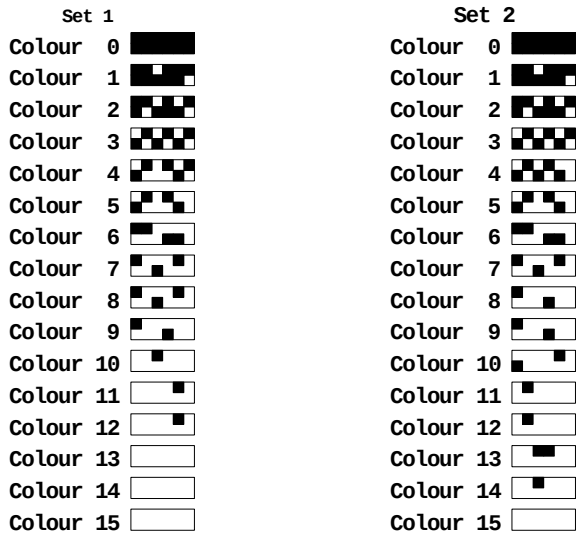
```
10960           LDA &7A
10970           EOR #255
10980           AND &75
10990           JSR out
11000           LDA &70
11010           CLC
11020           ADC #8
11030           STA &70
11040           LDA &71
11050           ADC #0
11060           STA &71
11070           CMP #5
11080           BNE line1
11090           LDA #13
11100           JMP out

25000 ]
25010 NEXT
25020 ENDPROC
```

We now have a finished program. To try it out you can use the example program from the large Mode 4 dump routine.

Now that we have discovered the maximum resolution of the printer it is well worth going back to the Mode 2 dump. Let us consider using pixels represented as six dots by three dots. If we dump a Mode 2 screen at the highest resolution using this system then it will be about 4 inches by 3.5 inches. Also, we should be able to make a grey scale that will handle 16 levels of brightness. This means that we can use each of the 16 colours that the BBC Micro will handle in Mode 2. Obviously we can't represent flashing colours so the best way to make use of this is not to try and print exactly what is on the screen but to represent each colour from 0 to 15 as a shade with 0 darkest and 15 brightest.

It turns out that with six-by-three dots at the highest resolution the third row can always be left blank and we still get a good range of colours. This speeds printing up as we will only have to print four interleaved lines. But if we try to print the same pattern for each pixel, one above the other, they will tend to produce visible vertical lines. So we will use TWO sets of patterns and use alternate sets for

alternate rows of pixels. The patterns we will use are as below:



As we are going to dump all the top rows then all the bottom rows, it would seem sensible to block all the top rows in one group and all the bottom rows in one group. Within each group we can group the entries into all the column ones then all the column twos, etcetera. We can store the two alternate sets of patterns, in this format, one after the other. We are going to build up the bytes to send to the printer by shifting them left and ORing them with 1 or 0. For this reason it will be easiest if we use one whole byte of a table to store each of the dot patterns. These bytes will either be 1 for a dot or 0 for no dot. The data looks as set out below.

```
20000 DATA1,1,1,0,0,0,1,1
20010 DATA1,1,0,0,0,0,0,0
20020 DATA1,1,1,1,1,1,1,0
20030 DATA0,0,0,0,0,0,0,0
20040 DATA1,0,0,0,0,0,0,0
20050 DATA0,0,1,0,0,0,0,0
20060 DATA1,1,1,1,1,1,0,0
20070 DATA0,0,0,1,0,0,0,0
20080 DATA1,1,0,0,0,0,0,1
```

```
20090 DATA1,0,0,1,1,0,0,0
20100 DATA1,1,1,1,1,0,0,0
20110 DATA0,0,0,0,0,0,0,0
20120 DATA1,1,1,1,1,1,0,0
20130 DATA0,0,0,0,0,0,0,0
20140 DATA1,1,0,0,0,0,0,0
20150 DATA0,0,0,0,0,0,0,0
20160 DATA1,1,1,1,0,0,0,1
20170 DATA1,0,0,0,0,0,0,0
20180 DATA1,1,1,0,0,0,1,0
20190 DATA0,1,0,0,0,0,0,0
20200 DATA1,1,1,1,1,1,1,0
20210 DATA0,0,0,0,0,0,0,0
20220 DATA1,0,0,0,0,0,0,0
20230 DATA0,0,0,0,0,0,0,0
20040 DATA1,1,1,0,0,0,1,1
20250 DATA1,1,0,0,0,0,0,0
20260 DATA1,1,1,1,1,1,1,0
20270 DATA0,0,0,1,1,0,0,0
20280 DATA1,0,0,0,0,0,0,0
20290 DATA0,0,0,0,0,1,1,0
20300 DATA1,1,1,1,1,1,0,0
20310 DATA0,0,0,0,0,1,0,0
20320 DATA1,1,0,0,0,0,0,1
20330 DATA0,0,1,0,0,0,0,0
20340 DATA1,1,1,1,0,0,0,0
20350 DATA0,0,0,0,0,0,0,0
20360 DATA1,1,1,1,1,1,0,0
20370 DATA1,1,0,0,0,0,0,0
20380 DATA1,1,0,0,0,0,0,0
20390 DATA0,0,0,0,0,0,0,0
20400 DATA1,1,1,1,1,0,0,1
20410 DATA0,0,0,0,0,0,0,0
20420 DATA1,1,1,0,0,0,1,0
20430 DATA1,1,0,0,0,0,0,0
20440 DATA1,1,1,1,1,1,1,0
20450 DATA0,0,0,0,0,0,0,0
20460 DATA1,0,0,0,0,0,0,0
20470 DATA0,0,0,0,0,0,0,0
```

So our first job is to read this into a reserved area of memory.

```
10000 DEFPROCass
10010 DIMmc%1000,D%383
10020 oswrch=&FFEE
10030 FOR A%=0TO383
```

```
10040 READD%?A%
10050 NEXT
10060 FORpass%=0TO2STEP2
10070 P%=mc%
10080 [OPTpass%
```

We will again need our trust routines, POINT and OUT. This time we don't need to worry about the point being off the screen as we are printing at eight pixels per band and eight goes into 256 exactly.

```
18000 .point   STX &76
18010          STY &77
18020          LDA #9
18030          LDY #0
18040          LDX #&70
18050          JSR &FFF1
18060          LDX &76
18070          LDY &77
18080          LDA &74
18090          RTS
19000 .out     PHA
19010          LDA #1
19020          JSR oswrch
19030          PLA
19040          JMP oswrch
19050 ]
19060 NEXT
19070 ENDPROC
```

Now for the main routine. As with the previous routine we need first of all to disable the interrupts, reset windows, disable printer line-feeds and turn the printer on.

```
10090 .dump    SEI
10100          LDA #26
10110          JSR oswrch
10120          LDA #6
10130          LDX #10
10140          LDY #0
10150          JSR &FFF4
10160          LDA #2
10170          JSR oswrch
```

Next we need to set up the Y coordinate of the top of the screen in &78 and &79.

```
10180          LDA #&FC
10190          STA &78
10200          LDA #3
10210          STA &79
```

Before we carry on, we need a routine to print one pass of the printer head. We can specify that on entry the Y coordinate of the top left-hand corner of the band is in &78 and &79; that the contents of &7B are zero for the top row and 96 for the bottom row (this allows us to add this to the table address to take care of which row we are printing); and that &7A contains a mask which is 255 for the even-numbered dots and zero for the odd numbered ones.

The first job, as always, is to set the printer to the right graphics mode − here, quadruple-density with 960 (&3C0) dots across.

```
15000 .band    LDA #27
15010          JSR out
15020          LDA #42
15030          JSR out
15040          LDA #3
15050          JSR out
15060          LDA #&C0
15070          JSR out
15080          LDA #3
15090          JSR out
```

Next we must set the X coordinate to zero.

```
15100          LDA #0
15110          STA &70
15120          STA &71
```

Next, for each column of pixels we send to the printer, we need to set up the address of the table in &7C and &7D. This will be D% plus the contents of &7B.

```
15130 .column  LDA #D%MOD256
15140          CLC
```

```
15150           ADC &7B
15160           STA &7C
15170           LDA #D%DIV256
15180           ADC #0
15190           STA &7D
```

Now we are ready to send six bytes to the printer. We can use the X register to count the bytes. For each byte we need to make a temporary working copy of the contents of &78 and &79 in &72 and &73.

```
15200           LDX #6
15210 .byte     LDA &7B
15220           STA &72
15230           LDA &79
15240           STA &73
```

We are going to work on the byte in &75 so we need to set it to zero for starters. Then we can count the bits we have worked on, with the Y register.

```
15250           LDA #0
15260           STA &75
15270           LDY #8
```

Now for every bit we calculate we first need to shift &75 left a bit. Then we must find the colour of the pixel.

```
15280 .bit      ASL &75
15290           JSR point
```

We are going to use the Y register to point into the table so we need to save the Y register at &76 until we have finished with the table. We have the colour of the pixel in the accumulator, so, by transferring it to the Y register we can use it directly to point into the table. However, if we are calculating an odd-numbered bit then we need to use the second set of shade patterns; these are 192 bytes further on in the table. We need to check bit 0 of the bit count which we have temporarily stored at &76; if it is one, we must add 192 to the Y register. Then we can load a byte from the table. At this point we have finished

with the Y register and we can reload its original bit count value.

```
15300          STY &78
15310          TAY
15320          LDA #1
15330          BIT &76
15340          BEQ skip1
15350          TYA
15360          CLC
15370          ADC #192
15380          TAY
15390 .skip1   LDA (&7C),Y
15400          LDY &76
```

This byte we need to OR with the byte we are calculating. We then need to move down a pixel and, if we haven't already finished the byte, go back and calculate the next bit.

```
15410          ORA &75
15420          STA &75
15430          LDA &72
15440          SEC
15450          SBC #4
15460          STA &72
15470          BCS skip2
15480          DEC &73
15490 .skip2   DEY
15500          BNE bit
```

Now we can send the byte to the printer. However, we must only send alternate bytes so we must mask it with &7A and reverse the mask in &7A ready for the next byte.

```
15510          LDA &75
15520          AND &7A
15530          JSR out
15540          LDA &7A
15550          EOR #255
15560          STA &7A
```

Next we must move the start of the table to the next column. Then, unless we have printed all six, we

must go back and print the next column.

```
15570          LDA &7C
15580          CLC
15590          ADC #16
15600          STA &7C
15610          BCC skip3
15620          INC &7D
15630 .skip3   DEX
15640          BNE byte
```

We have now dumped a column of eight pixels and can move on to the next column. If we have printed a whole line then we can send a carriage return and exit the routine.

```
15650          LDA &70
15660          CLC
15670          ADC #8
15680          STA &70
15690          LDA &71
15700          ADC #0
15710          STA &71
15720          CMP #5
15730          BNE column
15740          LDA #13
15750          JMP out
```

We can now finish off the main routine. For every text line, we must first set &7B to zero for the top row. Then, for each row, we must make two pases: first with the mask set to 255, then with the mask set to zero.

```
10220 .textlin LDA #0
10230          STA &7B
10240 .row     LDA #255
10250          STA &7A
10260          JSR band
10270          LDA #0
10280          STA &7A
10290          JSR band
```

Next we must feed the paper a fraction and set &7B to 96 for the second row. By EXCLUSIVE Oring &7B

with 96 we can use this to count the two rows.

```
10300          LDA #27
10310          JSR out
10320          LDA #74
10330          JSR out
10340          LDA #1
10350          JSR out
10360          LDA &7B
10370          EOR #96
10380          STA &7B
10390          BNE row
```

Now we can feed the paper up the rest of the line. Because the printer will not feed accurately at 1/216 of an inch it tends to feed more than this, so the two feeds we have performed will have fed closer to 3/216 of an inch. To correct this we need to feed only a further 21/216 of an inch. This may not be necessary on some printers, so you should experiment.

```
10400          LDA #27
10410          JSR out
10420          LDA #74
10430          JSR out
10440          LDA #21
10450          JSR out
```

All that remains is to move down to the next text line and repeat until we have dumped the whole screen; then we turn off the printer, re-enable the interrupts, and exit in the usual fashion.

```
10460          LDA &78
10470          SEC
10480          SBC #32
10490          STA &78
10500          LDA &79
10510          SBC #0
10520          STA &79
10530          BPL textlin
10540          LDA #3
10550          JSR oswrch
10560          CLI
10570          RTS
```

To try out the dump routine, add the following lines to the assembly code:

```
 10 MODE2
 20 VDU23,1,0;0;0;0;
 30 PROCass
 40 COLOUR135
 50 CLS
 60 COLOUR128
 70 P%=4
 80 FORA%=0TO16
 90 GCOL0,A%-1
100 MOVE0,0
110 PLOTP%,1000*SIN(A%*PI/32),1000*COS
    (A%*PI/32)
120 P%=85
130 NEXT
140 CALLdump
150 END
```

If you want to produce larger pictures it is relatively easy to join two or more dumps together. We have only looked at a selection of the screen dumps that can be written but these should give you an idea of how to go about writing any others you may need.