5 Character Graphics on the BBC Microcomputer

In all the MODEs except mode 7 each character to be drawn in two colours is made up of a set of eight by eight pixels and may be printed as any foreground colour. Such a block of pixels is known as a *character block*. The BBC micro has a 95 character *standard* set, and also has the capability of displaying *user-defined characters*. Both types of characters may be placed on the screen by the PRINT command, so we must look closely at this operation.

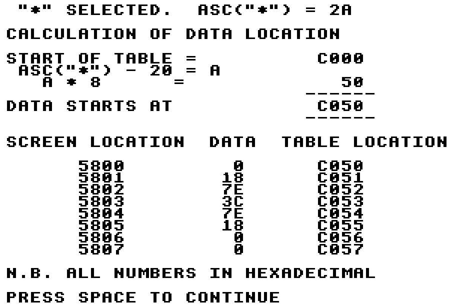
PRINT allows us to place characters at any text position on the screen. Such a position is specified by column and row numbers. The column number lies between 0 on the left of the screen, and on the right either 19 (in modes 2 and 5: sixteen-colour modes), 39 (modes 1, 4 and 6: four-colour modes) or 79 (modes 0 and 3: two-colour modes). The row or line position on this screen is one of the 32 lines (or 25 lines in modes 3 and 6), from top (0) to bottom (31 or 24). For each character position on the screen there are corresponding (mode-dependent) locations in the screen memory: 8 for two-colour modes, 16 for four-colour modes and 32 for sixteen-colour modes. In the two-colour mode each 8-pixel line of an 8 by 8 character block on the screen corresponds to an 8-bit binary memory location, one bit per pixel. if the binary digit is a 1 then the corresponding pixel will be drawn in the foreground colour, and if the digit is a 0 it will be drawn in the background colour. There is a table of data stored in the memory which defines the shape of each character. For any given character the PRINT command finds the corresponding eight 8-bit values from the table and copies them into the appropriate screen memory locations. This has the effect of displaying the character on the screen. Obviously if we are using four-colour or sixteen-colour modes and two-colour characters (background and foreground) then this method will not work; however PRINT automatically takes the 8 binary numbers from the table and converts them to the 16 or 32 numbers required for the multi-colour modes.

The Standard Character Set

The table of data for the standard character set is stored in ROM, the permanent Read Only Memory of the computer. There are eight pieces of data for each of the 95 characters, thus the table consists of 760 (95 * 8) consecutive locations and starts at &C000. Each character has a unique ASCII code number, see the user manual. The table contains the data for each of the characters in turn starting with the space character (ASCII code 32) and ending with the pound symbol £ (ASCII code 126). In order to PRINT a character the command requires the eight pieces of data for that character. To find the location in the table it subtracts 32 decimal (20 hexadecimal) from the ASCII code of the character, multiplies the result by eight and finally adds &C000. If a two-colour mode is in use with default colours then PRINT simply copies the data (foreground logical colour 1 and background 0) into the screen memory and the character appears on the screen. (We can invert the colours by setting the background to 1 and the foreground to 0.) However if a four-colour mode is heing used then PRINT must calculate the two eight-bit values from each of the eight pieces of table data in order to produce eight pixels in the appropriate colours. In the sixteen-colour mode the PRINT command must translate each piece of data into four eight-bit values, but more of this in a moment.

Example 5.1

First run listing 5.1 which demonstrates how this process works in a two-colour mode with foreground 1 and background 0 by showing the calculations as they are performed. A detailed explanation of how the screen memory locations are arranged is given later in this chapter, but for the moment we will limit ourselves to block (0, 0). Figure 5.1 is an example run of this program using ‘*’ as INPUT data (see the user guide for information about indirection operators). Note that this program makes no permanent change to the characters in the table, and also that all numbers are given in hexadecimal notation.



```
 "*" SELECTED.  ASC("*") = 2A
CALCULATION OF DATA LOCATION
START OF TABLE =          C000
 ASC("*") - 20 = A
   A * 8      =             50
                         ------
DATA STARTS AT            C050
                         ------
SCREEN LOCATION  DATA  TABLE LOCATION
      5800          0     C050
      5801         18     C051
      5802         7E     C052
      5803         3C     C053
      5804         7E     C054
      5805         18     C055
      5806          0     C056
      5807          0     C057
N.B. ALL NUMBERS IN HEXADECIMAL
PRESS SPACE TO CONTINUE
```

Listing 5.1

```
 10 MODE 4
 20 BORED=FALSE
 30 REPEAT : CLS
 40 REPEAT
 50 PRINT TAB(0,2);SPC(30);TAB(0,2);
 60 INPUT "Which character ? "A$
 70 UNTIL LEN(A$)=1
 80 CLS
 90 A=ASC(A$) : A$=""""+A$+""""
100 PRINT TAB(1,4);A$;" SELECTED.  ASC(";A$;") = ";~A'
110 PRINT"CALCULATION OF DATA LOCATION"'
120 PRINT"START OF TABLE = ",~&C000
130 B=A-ASC(" ") : C=B*8
140 PRINT" ASC(";A$;") - 20 = ";~B
150 PRINT"   ";~B;" * 8      = ",~C
160 D=&C000+C
170 PRINTSPC(25);"------"
180 PRINT"DATA STARTS AT ",~D
190 PRINTSPC(25);"------"'
200 PRINT"SCREEN LOCATION  DATA  TABLE LOCATION"'
210 S=HIMEM
219 REM transfer eight pieces of data from table to screen
220 FOR I%=0 TO 7
230 DAT=D?I%
239 REM change value of DAT here for solution of exercise 5.1
240 S?I%=DAT
250 PRINT~(S+I%),~DAT,~(D+I%)
260 NEXT I%
270 PRINT'"N.B. ALL NUMBERS IN HEXADECIMAL"
280 PRINT'"PRESS SPACE TO CONTINUE"
290 REPEAT UNTIL INKEY(-99)
300 UNTIL BORED
```

Exercise 5.1

Rewrite listing 5.1 so that it will give you the option of accepting or replacing by INPUT each of the eight data values before it is stored in the screen memory. Experiment by changing one or two values from a standard character.

We now give a method for taking values from the character table and drawing these characters on the screen for multi-colour modes. We start by defining a value n that equals 1, 2 or 4 for the two-colour, four-colour or sixteen-colour modes respectively, and we let $m = 8/n$. Suppose f and b are the n-bit binary numbers that represent the logical foreground and background colours respectively. We calculate a new value F, a foreground mask, by replacing each digit in f by m occurrences of the same digit (for example if $n = 2$ and f = 01, then $m = 4$ and F = 00001111). The background mask B can be created in the same way from b. For each 8-bit number V from the character table, we need to transform it into n 8-bit numbers S_i, $(1 \leq i \leq n)$ so that one line of a character may be drawn in two colours (f and b) on the screen. First we break V into n m-bit pieces p_i, $(1 \leq i \leq n)$, that is $V = p_1 \ldots p_n$. Then we create n 8-bit numbers A_i =

$(p_i)^n$, which means that the m digits of p_1 are repeated n times, for example, $(01)^4$ = 01010101. From there we calculate the S_i by:

S_i = (A_i AND F) OR (A_i AND B) $1 \le i \le n$

where A_i, = A_i EOR 11111111. We give three examples.

(a) Mode 2: n = 4 then m = 2.
Suppose f = 0001 , b = 0100 and V = 00010111 , then

F = 00000011 and B = 00110000

p_l = 00 A1 = 00000000 $\overline{A}_1$ = 11111111

p_2 = 01 A2 = 01010101 $\overline{A}_2$ = 10101010

p_3 = 0l A3 = 01010101 $\overline{A}_3$ = 10101010

p_4 = 11 A4 = 11111111 $\overline{A}_4$ = 00000000

S_1 = 00000000 OR 00110000 = 00110000

S_2 = 00000001 OR 00100000 = 00100001

S_3 = 00000001 OR 00100000 = 00100001

S_4 = 00000011 OR 00000000 = 00000011

Bits 7, 5, 3 and 1 represent the first colour from the word and bits 6, 4, 2, 0 the second: see chapter 1 . Thus using / to represent the relative position of colours and f for foreground and b for background:

S_1 = 00110000 → 0100/0100, that is b/b

S_2 = 00100001 → 0100/0001, that is b/f

S_3 = 00100001 → 0100/0001, that is b/f

S_4 = 00000011 → 0001/0001, that is f/f

Thus the four values combine to give the eight colours b/b/b/f/b/f/f/f which correspond directly to our original line 00010111.

(b) Mode 1 : n = 2 then m = 4.
Suppose f = 01 , b = 11 and V = 000l0111, then

F = 00001111 and B = 11111111

p_l = 0001 A_1 = 00010001 $\overline{A}_1$ = 11101110

p_2 = 0111 A_2 = 01110111 $\overline{A}_2$ = 10001000

S_1 = 00000001 OR 11101110 = 11101111

S2 = 00000111 OR 10001000 = 10001111

Bits 7 and 3, 6 and 2, 5 and 1, and 4 and 0 give the four colours from each byte.

S_1 = 1110111l → 11/11/11/01, that is b/b/b/f

S_2 = 10001111 → 11/01/01/0l, that is b/f/f/f

Thus the two values combine to give the eight colours b/b/b/f/b/f/f/f which correspond directly to our original line 00010111.

(c) Mode 0: $n = 1$ then $m = 8$.

Suppose f = 0, b = 1 and V = 00010111, then

F = 00000000   and   B = 11111111

p_l = 00010111 A_1=00010111 $\overline{A}_1$= 11101000

S_1 = 00000000 OR 11101000 = 11101000

Bits correspond directly to the eight pixels from the byte.

S_1 = 11101000 → 1/1/1/0/1/0/0/0,that is b/b/b/f/b/f/f/f.

Thus the value gives the eight colours b/b/b/f/b/f/f/f which correspond directly to our original line 00010111.

This logic is programmed in listing 5.2. At this early stage you need not make a great effort to understand the effect of each logical operation, unless you intend to study in detail the assembly language programs in this book.

Listing 5.2

```
 10 BORED=FALSE : REPEAT 20 INPUT "Which mode ",M : MODE M
 30 REPEAT
 40 PRINT TAB(0,2);SPC(30);TAB(0,2);
 50 INPUT"Which character ",A$
 60 UNTIL LEN(A$)=1
 69 REM same calculation as in 5.1
 70 A=ASC(A$)
 80 B=A-ASC(" ")
 90 C=B*8
100 D=&C000+C
110 S=HIMEM
120 INPUT"Which background ",BC
130 INPUT"Which foreground ",FC
140 ON M+1 GOTO 150,230,380,150,150,230,150
150 REM"section for 2_colour modes
160 COLF=&FF*FC : COLB=&FF*BC
170 FOR I%=0 TO 7
180 FMASK=D?I%
190 BMASK=FMASK EOR &FF
200 S?I%=(COLF AND FMASK) OR (COLB AND BMASK)
210 NEXT I%
220 GOTO 570
230 REM section for 4_colour modes
```

```
240 COLF=(FC AND 2)/2*&F0 + (FC AND 1)*&0F
250 COLB=(BC AND 2)/2*&F0 + (BC AND 1)*&0F
260 FOR I%=0 TO 7 : N%=D?I%
270 FMASK1=0 : FMASK2=0
279 REM check each bit of N% and put double-bit in the
        foreground  mask for that half if on.
280 FOR J%=0 TO 3
290 IF (N% AND 2^(J%+4)) THEN FMASK1=FMASK1 + &11*2^J%
300 IF (N% AND 2^J%) THEN FMASK2=FMASK2+ &11*2^J%
310 NEXT J%
320 BMASK1=FMASK1 EOR &FF : BMASK2=FMASK2 EOR &FF
329 REM mix the colours with masks to find two values and put
        2nd value eight bytes on i.e. next to first value.
330 I8%=I%+8
340 S?I%=(COLF AND FMASK1) OR (COLB AND BMASK1)
350 S?I8%=(COLF AND FMASK2) OR (COLB AND BMASK2)
360 NEXT I%
370 GOTO 570
380 REM"section for 16_colour mode
390 COLF=(FC AND 8)/8*&C0 + (FC AND 4)/4*&30+(FC AND 2)/2*&0C +
(FC AND 1)*&03
400 COLB=(BC AND 8)/8*&C0 + (BC AND 4)/4*&30+(BC AND 2)/2*&0C +
(BC AND 1)*&03
410 FOR I%=0 TO 7 : N%=D?I%
420 FMASK1=0:FMASK2=0 :FMASK3=0:FMASK4=0
429 REM check each bit of N% and put quadruple-bit in the mask
        for that quarter if on.
430 FOR J%=0 TO 1
440 IF (N% AND 2^(J%+6)) THEN FMASK1=FMASK1 + &55*2^J%
450 IF (N% AND 2^(J%+4)) THEN FMASK2=FMASK2 + &55*2^J%
460 IF (N% AND 2^(J%+2)) THEN FMASK3=FMASK3 + &55*2^J%
470 IF (N% AND 2^J%) THEN FMASK4=FMASK4 + &55*2^J%
480 NEXT J%
490 BMASK1=FMASK1 EOR &FF : BMASK2=FMASK2 EOR &FF
500 BMASK3=FMASK3 EOR &FF : BMASK4=FMASK4 EOR &FF
509 REM mix the colours with masks to find four values and put
each
        value eight bytes on i.e. next to the previous value.
510 I8%=I%+8 : I16%=I%+16 : I24%=I%+24
520 S?I%=(COLF AND FMASK1) OR (COLB AND BMASK1)
530 S?I8%=(COLF AND FMASK2) OR (COLB AND BMASK2)
540 S?I16%=(COLF AND FMASK3) OR (COLB AND BMASK3)
550 S?I24%=(COLF AND FMASK4) OR (COLB AND BMASK4)
560 NEXT I%
570 UNTIL BORED
```

Exercise 5.2

Write a program that does the inverse of this process and can recognise a character that is printed at block (0, 0) in any foreground and background colours, in modes 0, 1 or 2. This will mean calculating the eight pieces of data used to construct the character and comparing them with each set of eight in the character table until you find a match. Note that you will need to know the logical colour of the background, which will be equivalent to zero bits in our eight numbers, everything else must be foreground and ‘ones' . Remember that when using the OSBYTE call with A% = &87 (see the user guide) it is essential to ensure that the background COLOUR is set to the appropriate value (as is done in the worm game in chapter 1).

User-defined characters: VDU 23

This second type of character is treated in exactly the same way as the ordinary character set, but the table that holds the data for these characters is in RAM. When the machine is turned on, there is an area reserved for a table that defines the 32 characters with ASCII codes from 128 to 159. This table consists of the 256 bytes stored in memory from &C00 to &CFF. From OS 1.0 onwards the red user-definable function keys at the top of the keyboard (the *soft* keys) will produce some of these characters when used with the shift and/or control keys (see the user guide). The data in the table for these characters can be changed by the user, so that any required character can be typed into a BASIC PRINT text string directly from the keyboard. The VDU 23 command is used to redefine these keys, and is entered in the format

VDU 23, ASCII code, eight (8-bit) numbers

Example 5.2
The program in listing 5.3 allows you to redefine a character with ASCII code between 128 and 159, by typing all eight 8-bit binary numbers for a character. After redefining a character, the program also prints out the data in a hexadecimal form for inclusion as DATA statements in future programs (for example, listing 5.4). This information can be copied into a program by using the COPY key. The 'display' procedure may also be used to print out the data for a character in the ASCII range 128 to 159 in binary notation.

Exercise 5.3
Use listing 5.3 to generate a character that consists of a chequer-board pattern of pixels (for example, 85, 170, 85 etc.). Experiment with this program by using various background and foreground colours in order to make new colours (for example, red and yellow give orange).

Exercise 5.4
Use the procedures from listing 5.3 to make an elementary character editor. Move the text cursor around the 'display'ed data and recognise each binary digit, altering it where desired. When you wish to replace the character you will need a method of reconstructing a decimal or hexadecimal number from the binary strings for each line so that they can be stored in the memory.

Example 5.3
Listing 5.4, which contains DATA generated by listing 5.3, can be used to create 'thin' numbers for modes 1 and 2 and to store them as characters that are equivalent to the ASCII codes 128 to 137 and 144 to 146. Hence shift and the soft keys f0 to f9 give the 'thin' digits, and control with f0, fl and f2 give 'thin' '+', '–' and '.'. It also contains a procedure to print a string of 'thin' digits on the screen. These characters will be particularly useful for labelling data diagrams

Listing 5.3

```
 10 MODE 4
 20 DIM A(7)
 30 REPEAT
 40 PRINTTAB(0,1);SPC(39);TAB(0,1);
 50 INPUT "Which ASCII code ",CHAR
 60 UNTIL CHAR>127 AND CHAR<160
 70 PRINT"TYPE IN EIGHT, 8-BIT BINARY NUMBERS"
 80 PRINT'" --------"
 90 FOR I%=0 TO 7
100 REPEAT
110 PRINTTAB(0,I%+5);SPC(39);TAB(0,I%+5);
120 INPUTA$ : A=FNT(A$)
130 UNTIL A>=0 AND A<=255
140 A(I%)=A
150 NEXT I%
160 VDU23,CHAR
170 FORI%=0TO7
180 VDU A(I%)
190 NEXT
200 PRINT''"ASCII CODE ";CHAR;" NOW REDEFINED";
              '"TO REPRESENT ";CHR$(CHAR)
210 PRINT'"DATA ";
220 FOR I%=0 TO 7
230 PRINT"&";~A(I%);",";
240 NEXT I%
249 REM 'delete' to erase last comma
250 VDU 127
260 PRINT
270 END
300 REM base ten to binary conversion
310 DEF FNB(N)
320 LOCAL I%,B$
330 B$="":I%=N
340 REPEAT
350 IFI% DIV 2=I%/2 THEN B$="0"+B$ ELSE B$="1"+B$
360 I%=I% DIV 2 : UNTIL I%=0
370 =B$
400 REM binary to base ten conversion
410 DEF FNT(B$)
420 LOCAL T,I%
430 T=0 : I%=1 : REPEAT
440 T=T*2+VAL(MID$(B$,I%,1))
450 I%=I%+1 : UNTIL I%>LEN(B$)
460 =T
500 REM display
510 DEF PROCdisplay(C)
519 REM print binary values defining a character
        (see also OSWORD call with A=&0A)
520 IF C<128 OR C>159 THEN ENDPROC
530 D=(C-128)*8+&C00
540 FOR I%=0 TO 7
550 K=D?I% : B$=FNB(K) : PRINT LEFT$("00000000",8-LEN(B$));B$
560 NEXT I%
570 ENDPROC
```

in mode 1 and for printing the score in games in mode 2. Note how in this case the PRINT procedure starts at a text block position but subsequently places the characters by using the graphics cursor.

Listing 5.4

```
 10 MODE 1
 20 FOR I%=128 TO 137
 30 PROCread(I%)
 40 NEXT I%
 50 DATA &40,&A0,&A0,&A0,&A0,&A0,&40,0
 60 DATA &40,&C0,&40,&40,&40,&40,&E0,0
 70 DATA &40,&A0,&20,&40,&80,&80,&E0,0
 80 DATA &40,&A0,&20,&40,&20,&A0,&40,0
 90 DATA &20,&60,&A0,&A0,&E0,&20,&20,0
100 DATA &E0,&80,&C0,&20,&20,&A0,&40,0
110 DATA &40,&A0,&80,&C0,&A0,&A0,&40,0
120 DATA &E0,&20,&20,&40,&40,&80,&80,0
130 DATA &40,&A0,&A0,&40,&A0,&A0,&40,0
140 DATA &40,&A0,&A0,&60,&20,&A0,&40,0
150 FOR I%=144 TO 146
160 PROCread(I%)
170 NEXT I%
180 DATA &0,&40,&40,&E0,&40,&40,&0,&0
190 DATA &0,&0,&0,&E0,&0,&0,&0,&0
200 DATA &0,&0,&0,&40,&40,&0,&0,&0
209 REM test 'thin' with a string
210 PROCthin(1,10,10,"0.123+456-789")
220 END
230 REM read
240 DEF PROCread(I%)
249 REM redefine character I% using eight pieces of data
250 VDU 23,I%
260 FOR J%=1 TO 8
270 READ K : VDU K
280 NEXT J%
290 ENDPROC
300 REM thin
310 DEF PROCthin(M,X,Y,A$)
319 REM this only gives correct size for modes < 3
320 A%=2^(4+M)
329 REM use graphics cursor to place each character of string
330 VDU 5
340 MOVE X*A%,1023-Y*32
350 FOR I%=1 TO LEN(A$) : P$=MID$(A$,I%,1)
360 IF P$>="0" AND P$<="9" THEN VDU (80+ASC(P$)) : GOTO 380
369 REM" deal with special symbols
370 IF P$="+" THEN VDU 144 ELSE IF P$="-" THEN VDU 145 ELSE VDU 146
380 MOVE (X+I%/2)*A%,1023-Y*32
390 NEXT I%
399 REM separate text and graphics
400 VDU 4
410 ENDPROC
```

Example 5.4
Defined characters can be directly incorporated into your programs to enhance the speed of display construction. The equivalent character could be constructed by a series of DRAW and PLOT commands but this would take longer in most cases. For instance we can build up tessellated patterns on the screen with characters in a very quick time compared to the length of time it would take to PLOT or DRAW the same pattern. Dsting 5 .5 shows how tMs can be done by using combinations of the user-defined characters with ASCIIcodes 128 to 131 that were constructed by using listing 5.3.

Listing 5.5

```
  9 REM tessellation pattern
 10 MODE 4
 20 CLS : INPUT "TWO ACTUAL COLOURS ",FC,BC : CLS
 30 GCOL 0,128 : GCOL 0,1 : MOVE 0,1023
 40 VDU 19,0,BC,0,0,0 : VDU 19,1,FC,0,0,0
 49 REM use graphics cursor as text text cusor so that display
        doesn't scroll
 50 VDU 5
 60 A=128 : B=129 : C=130 : D=131
 70 FOR Y%=0 TO 31
 80 FOR X%=0 TO 39 STEP 4
 90 VDU A,B,C,D
100 NEXT X%
109 REM shift characters around
110 T=A : A=B : B=C : C=D : D=T
120 NEXT Y%
130 VDU 4
140 REPEAT UNTIL INKEY(-99)
```

Exercise 5.5
Experiment with various possible symmetries of characters and patterns for placement of characters on the screen (that is, the order in which groups of characters are drawn on the screen). Alter the program so that it tries all the possible combinations of foreground and background colours within two nested FOR. . .NEXT loops.

We have seen that the amount of work that goes into constructing even one character is enough to imply that the construction of a complete new set of characters would be a very arduous task. So we need a program that will simplify this task, and that will also allow us to alter characters that have already been created. Thd following program is such a character generation and editing program for ASCII codes from 32 to 255, and was designed for use in the development of graphical display programs.

Listing 5.6

```
 10 MODE 4
 19 REM ensure all blocks can be used
 20 *FX20,6
 30 DIM B(8),A(7),S(7,7),R(7,7)
 40 DIM M$(6)
 50 FOR I%=1 TO 6 : READ M$(I%) : NEXT I%
 60 DATA 1  ... DISPLAY CHARACTERS,2  ... DISPLAY ALL CHARACTERS
        ,3  ... EDITOR,4  ... SAVE CHARACTERS,5  ... LOAD CHARACTERS
        ,6  ... TEST PROGRAM
 70 FOR I%=1 TO 7 : READ A(I%)
 80 IF A(I%)<&C00 THEN A(I%)=A(I%)+PAGE-&600
 90 NEXT I%
100 DATA &300,&400,&500,&C00,&0,&100,&200
 110 QUIT=FALSE
 120 REPEAT : PROCmenu : UNTIL QUIT
 130 PROCtestprog
 140 END

 200 REM menu
 210 DEF PROCmenu : VDU 19,1,2,0,0,0
 220 COLOUR 128 : CLS : COLOUR 1
 229 REM display options and wait for selection
 230 PRINT TAB(6,3);"*** CHARACTER GENERATOR ***"''
 240 FOR I%=1 TO 6
 250 PRINT SPC(6);M$(I%)''
 260 NEXT I%
 270 PRINT "Which option ? ";
 280 REPEAT : A$=GET$ : UNTIL A$>"0" AND A$<"7"
 290 VDU19,1,7,0,0,0
 300 IF A$="1" THEN PROCdisplay
 310 IF A$="2" THEN PROCdisall
 320 IF A$="3" THEN PROCedit
 330 IF A$="4" THEN PROCfile("SAVE")
 340 IF A$="5" THEN PROCfile("LOAD")
 350 IF A$="6" THEN QUIT=TRUE : ENDPROC
 360 PRINT TAB(0,31);"PRESS SPACE TO RETURN TO MENU";
 370 REPEAT UNTIL INKEY(-99)
 380 ENDPROC

 400 REM display OPTION 1
 410 DEF PROCdisplay
 419 REM display one block of 32 characters with their codes.
 420 COLOUR 129 : CLS : COLOUR 0
 430 PRINT TAB(0,1);"Which block ?";
 440 REPEAT
 450 B$=GET$ : B=VAL(B$)
 460 UNTIL B<8 AND B>0
 470 PRINT B : C=32*B
 480 FOR I%=0 TO 31
 490 PRINT TAB((I% MOD 2)*20,I% DIV 2+6);
 499 REM remember 127 is 'delete'
 500 PRINTI%+C;"= ";: IF I%<>127 THEN VDU I%+C
 510 NEXT I%
 520 ENDPROC

 600 REM disall OPTION 2
 610 DEF PROCdisall
 620 COLOUR 129 : CLS : COLOUR 0
 630 PRINT''
 639 REM show all characters except 'delete'
 640 FOR I%=32 TO 255
 650 IF I%<> 127 THEN VDU 32,I%
 660 NEXT I%
 670 ENDPROC

 700 REM edit OPTION 3
 710 DEF PROCedit
 719 REM set cursor keys for use
 720 *FX4,1
 729 REM give submenu options and draw grid for editing
 730 COLOUR 129 : CLS : COLOUR 0
 740 PRINT'"    1) COLOUR 1    0) COLOUR 0"
 750 PRINT"    <SPACE> TO COLOUR SQUARE"
 760 PRINT"    CURSOR KEYS MOVE THE CROSS"
 770 PRINT"    X)-AXIS     Y)-AXIS     R)OTATE"
 780 PRINT"    U)NPACK     P)ACK       M)ERGE"
 790 FOR I=0 TO 7 : FOR J=0 TO 7
     : PROCsquare(I,J,0) : NEXT J : NEXT I
 800 LX=320 : RX=LX+640 : BY=128 : TY=BY+640
 810 GCOL0,0 : MOVE LX,BY
     : DRAW LX,TY : DRAW RX,TY : DRAW RX,BY : DRAW LX,BY
 820 I=0 : J=0 : C=0 : PROCmark(I,J)
 829 REM remove cursor cross and scan keyboard for commands
 830 REPEAT : PROCmark(I,J)
 840 IF INKEY(-42) AND J>0 THEN J=J-1
 850 IF INKEY(-58) AND J<7 THEN J=J+1
 860 IF INKEY(-26) AND I>0 THEN I=I-1
 870 IF INKEY(-122) AND I<7 THEN I=I+1
 880 IF INKEY(-99) THEN PROCsquare(I,J,C)
 889 REM replace cross and check for rest of commands
 890 PROCmark(I,J)
 900 A$=INKEY$(0)
 910 IF A$="1" OR A$="0" THEN C=VAL(A$) ELSE 930
 920 COLOUR 0 : PRINT TAB(0,0);"COLOUR = ";C;
     : COLOUR C:PRINT" ** " : COLOUR 0
 930 IF A$="R" THEN PROCrotate
 940 IF A$="X" THEN PROCx_axis
 950 IF A$="Y" THEN PROCy_axis
 959 REM all final three options use an ASCII value so input it
 960 IF NOT (A$="U" OR A$="P" OR A$="M") THEN 1040
 970 REPEAT
 980 PRINT TAB(0,30);SPC(39);TAB(0,30);
 990 INPUT "Which character ",CHAR
1000 UNTIL CHAR<256 AND CHAR>31
1010 IF A$="U" THEN PROCunpack : PROCr_to_s : PROCmark(I,J)
1020 IF A$="M" THEN PROCmerge : PROCmark(I,J)
1030 IF A$="P" THEN PROCpack
1040 UNTIL A$="P"
1049 REM return cursor keys to normal
1050 *FX4,0
1060 ENDPROC

1100 REM square
1110 DEF PROCsquare(X,Y,IN) : GCOL0,IN
1119 REM colour square in on grid
1120 LX=640+80*(X-4)+4 : RX=LX+72 : BY=Y*80+132 : TY=BY+72
1130 MOVE LX,BY : MOVE LX,TY : PLOT 85,RX,BY : PLOT 85,RX,TY
1139 REM plot point on small character
1140 S(X,Y)=IN : PLOT 69,1027+X*4,320+Y*4
1150 ENDPROC
```

```
1200 REM mark
1210 DEF PROCmark(X,Y) : GCOL 3,7
1219 REM eor cross onto a square for use as cursor
1220 LX=650+80*(X-4) : RX=LX+60 : BY=Y*80+138 : TY=BY+60
1230 MOVE LX,BY : DRAW RX,TY : MOVE RX,BY : DRAW LX,TY
1240 ENDPROC

1300 REM pack
1310 DEF PROCpack
1319 REM redefine character using data direct from screen
1320 VDU 23,CHAR
1330 FOR I%=0 TO 7
1340 VDU (I%?&7340)
1350 NEXT
1360 ENDPROC

1400 REM unpack
1410 DEF PROCunpack
1419 REM place character on screen and read data directly
     from its position.
1420 COLOUR 128 : COLOUR 1 : PRINTTAB(32,21); : VDU CHAR
1430 FOR I%=0 TO 7 : B(I%)=I%?&7340 : NEXT I%
1439 REM convert bytes in B() to bits in R(,)
1440 FOR B=0 TO 7
1450 Y=7-(B MOD 8)
1460 FOR X=0 TO 7
1470 IF B(B) AND 2^(7-X) THEN R(X,Y)=1 ELSE R(X,Y)=0
1480 NEXT X : NEXT B
1490 COLOUR 129 : COLOUR 0
1500 ENDPROC

1600 REM rotate
1610 DEF PROCrotate
1619 REM shift bits around into R
1620 FOR I%=0 TO 7
1630 FOR J%=0 TO 7
1640 R(I%,J%)=S(J%,7-I%)
1650 NEXT J% : NEXT I%
1660 PROCr_to_s
1670 ENDPROC

1700 REM r_to_s
1710 DEF PROCr_to_s
1719 REM put bits from R into S and update screen display
1720 FOR I%=0 TO 7
1730 FOR J%=0 TO 7
1740 S(I%,J%)=R(I%,J%)
1750 PROCsquare(I%,J%,S(I%,J%))
1760 NEXT J% : NEXT I%
1770 ENDPROC

1800 REM y_axis
1810 DEF PROCy_axis
1819 REM flip bits around into R
1820 FOR I%=0 TO 7
1830 FOR J%=0 TO 7
1840 R(I%,J%)=S(7-I%,J%)
1850 NEXT J% : NEXT I%
1860 PROCr_to_s
1870 ENDPROC
```

```
1900 REM x_axis
1910 DEF PROCx_axis
1919 REM flip bits around into R
1920 FOR I%=0 TO 7
1930 FOR J%=0 TO 7
1940 R(I%,J%)=S(I%,7-J%)
1950 NEXT J% : NEXT I%
1960 PROCr_to_s
1970 ENDPROC

2000 REM file OPTIONS 4 AND 5
2010 DEF PROCfile(B$)
2020 CLS : INPUT "WHAT FILE NAME",A$ : IF A$="" THEN ENDPROC
2030 PRINT"WHAT BLOCK ?";
2040 REPEAT: C$=GET$ : UNTIL C$>"0" AND C$<"8"
2050 C=VAL(C$) : VDU22,7
2059 REM change to mode 7 and print command into memory
2060 PRINTB$;" ";A$;" ";~A(C); : IF B$="SAVE" THEN PRINT " +100";
2069 REM put a carriage return into the memory to finish command
2070 PRINT:?&7C28=&D
2079 REM set pointers to screen and use CLI to execute command
2080 X%=0:Y%=&7C:CALL&FFF7
2089 REM switch back to mode 4
2090 VDU22,4
2100 ENDPROC

2200 REM merge
2210 DEF PROCmerge
2220 PROCunpack
2229 REM mix incoming character in R with existing one in S
2230 FOR I%=0 TO 7
2240 FOR J%=0 TO 7
2250 R(I%,J%)=S(I%,J%) OR R(I%,J%)
2260 NEXT J% : NEXT I%
2270 PROCr_to_s
2280 COLOUR 129 : COLOUR 0
2290 ENDPROC

2300 REM tessellation testprog
2310 DEF PROCtestprog
2320 CLS : INPUT "TWO ACTUAL COLOURS ",FC,BC : CLS
2330 GCOL 0,128 : GCOL 0,1 : MOVE 0,1023
2340 VDU 19,0,BC,0,0,0 : VDU 19,1,FC,0,0,0
2349 REM use graphics cursor as text cursor so that display
         doesn't scroll
2350 VDU 5
2360 A=128 : B=129 : C=130 : D=131
2370 FOR Y%=0 TO 31
2380 FOR X%=0 TO 39 STEP 4
2390 VDU A,B,C,D
2400 NEXT X%
2409 REM shift characters around
2410 T=A : A=B : B=C : C=D : D=T
2420 NEXT Y%
2430 VDU 4
2440 REPEAT UNTIL INKEY(-99)
2450 ENDPROC
```

The CHARACTER GENERATOR 1 (listing 5.6) procedures are intended to take all the hard work out of preparing and using defined characters: they allow you to edit and to manipulate characters, to save and to reload defined characters, and to use them with your own programs. The characters are split into seven groups each of 32 characters, that is codes 32 to 63, 64 to 95 etc. (see the user guide for a discussion of OSHWM, and *FX20, 1: although with some systems it is necessary to use *FX20,6 (!)). Because so much space is needed for the characters it is necessary to move the BASIC prograins along the store. You must set the PAGE value to &600 greater than normal; that is, for an OS 0.1 type PAGE = &1400, and for OS 1.0 and above (from now on referred to as OS ≥ 1.0) PAGE = &1F00.

The program offers a choice of six options.

(1) The first option is DISPLAY CHARACTERS. On selecting this option you will be asked which of the seven blocks of 32 consecutive characters you wish to view. The screen will then display the 32 ASCII codes followed by the characters currently defined for those codes in black on a white background.
(2) The second option is DISPLAY ALL CHARACTERS. On selecting this option the screen will display all the characters for ASCIIcodes from 32 to 255 in black on a white background.
(3) The third option is the editor, which is the most complicated option and has a large set of commands that are accessed by typing a single key.

In edit mode a character will be displayed as black blocks in an 8 by 8 grid, with a cross (or edit cursor) initially placed in the bottom left-hand corner. The cursor is controlled by the standard cursor keys either singly or in pairs.

Pressing the space bar will change a square in the character grid to the currently selected colour (1 or 0). White is equivalent to a binary one and black is zero. The current colour is displayed in the top left-hand corner of the screen and may be altered at any time by pressing either 1 or 0.

The next two commands are UNPACK and PACK, which are activated by typing their initial letters. UNPACK takes the piece of data for a character from the memory and converts it into separate binary digits for display in the grid as well as PRINTing it at normal size on the screen alongside the grid. Pack takes the piece of data from the grid and uses it to redefine the character. By plotting the character at normal size during the editing, we are in fact creating the eight bytes of data in the screen memory and so it is simple to read the piece of data from the screen and place it directly into the character table. Should we use this program with an optional extra processor (see the user guide) we would have to replace the PACK procedure with one that calculated the data from the S array by the type of logical functions given in listing 5.4.

The next three commands all specify transformations similar to those that were used to perform on two-dimensional objects in chapter 4. ROTATE turns the character through 90 degrees anticlockwise about its centre. X-AXIS reflects the character about the horizontal axis and Y-AXIS reflects the character about the vertical. These commands can be used to create character sets for any orientation.

Finally we have MERGE. This allows any character to be merged into the grid on top of what is already being edited. This is very useful for creating foreign language sets, for example, to place a slash through an O in the Scandinavian languages, or to add accents to letters in French etc.

(4) and (5) The fourth option allows characters to be SAVEd in blocks of 32 characters, either on disk or on tape and the fifth option allows them to be reLOADed. For both options we must reply to the question ‘WHICH SET’ with a number between 1 and 7, in which case that particular block is SAVEd or reLOADed.

To allow other programs to load and to use alternate sets of characters created in this way, then the ‘file’ procedure must be included in the new program.

(6) The sixth option simply calls up procedure ‘testprog’ which can either contain a short test program or simply end the procedure.

In order to familiarise yourself with the ‘CHARACTER GENERATOR 1’ carry out the following instructions. Create a character like a spidery ink-blot pattern and SAVE it as ASCII code 128. Unpack this character. Rotate and SAVE it as ASCIIcode 129. Edit 129, use X-AXIS reflection and SAVE it as code 130. Edit 130, use Y-AXIS reflection, and SAVE it as 131 . Now use option (6) to run the ‘testprog’ procedure which contains a tessellation program from listing 5.5, and see what pattern emerges.

Exercise 5.6

Write a routine that affects a whole block of characters one after another and uses some of the editor routines from the ‘CHARACTER GENERATOR 1’ to perform transformations on each character. Figure 5.2 shows a listing that was produced with characters that have been ROTATED.

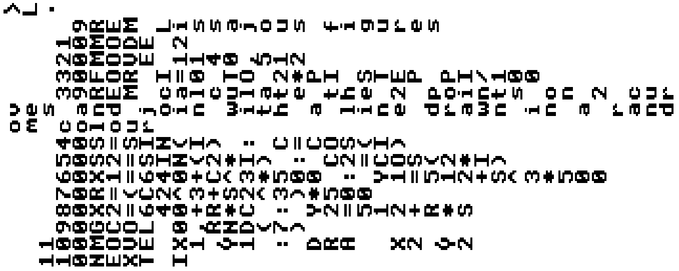
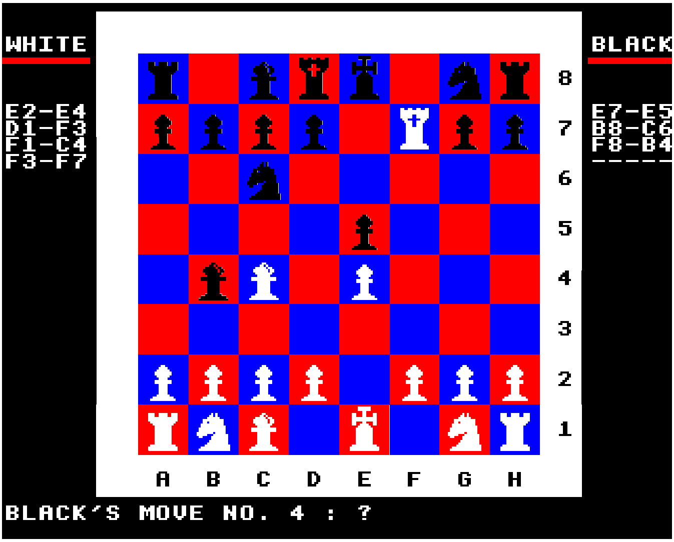


Figure 5.2

Example 5.5
We now give an example of a complete program (listing 5.7) which uses the characters developed with ‘CHARACTER GENERATOR 1’. The following program.(loaded at &1F00 for OS ≥ 1 .0 and &1400 for OS 0.1) simulates a chess board. A picture of a typical display is shown in figure 5.3. Each chess piece is constructed from 9 characters and placed 3 by 3 on the screen. The 56 characters required for the display were created as ASCII codes 160 to 215, subsequently stored as two blocks in locations starting at &1900 and &1A00 (OS > 1 .0) or &E00 and &F00 (OS 0.1), and placed as files CHESSP1 and CHESSP2 on backing store. You may have to strip the REMarks from the program in order to have enough store to run it.



```
WHITE                                  BLACK

E2-E4                                  E7-E5
D1-F3                                  B8-C6
F1-C4                                  F8-B4
F3-F7                                  -----

BLACK'S MOVE NO. 4 : ?
```

Figure 5.3

Exercise 5.7
The program in listing 5.7 simply acts as a chess board and places the last 20 moves for each player on the side of the screen: the side columns scroll if more than 20 moves are made. Adapt the program so that it checks for illegal moves, and add facilities for castling and *en passant* captures. If you have a *lot of time!!* to spare then add routines to make the computer play against you (see Liffick, 1979).

Listing 5.7

```
  9 REM data for chesspieces
 10 *LOAD CHESSP1 1900
 20 *LOAD CHESSP2 1A00
 30 MODE 1 : VDU23,1,0;0;0;0;
 40 VDU19,2,4,0,0,0 : *FX20,6
 50 DIM B(8,8),C(8,8),N$(1,100)
 60 PRINTTAB(0,2);"WHITE";SPC(30);"BLACK"
 69 REM"underline WHITE and BLACK with red blocks
 70 GCOL 0,1
 80 MOVE 0,911 : MOVE0,919
 90 PLOT 85,160,911 : PLOT 85,160,919
100 MOVE 1279,911 : MOVE 1279,919
110 PLOT 85,1119,911 : PLOT 85,1119,919
119 REM draw tablecloth under board
120 GCOL0,3
130 MOVE 176,80 : MOVE 1103,80
140 PLOT 85,176,1007 : PLOT 85,1103,1007
150 COLOUR 131 : COLOUR 0
160 FOR I%=1 TO 8
169 REM print letters along bottom and numbers up the side
170 PRINT TAB(I%*3+6,28);CHR$(64+I%)
180 PRINT TAB(33,(8-I%)*3+4);I%
189 REM put pawns on row 2 and make them white
190 B(I%,2)=6 : C(I%,2)=3
199 REM"put pawns on row 7
200 B(I%,7)=6
209 REM read order of pieces for back rows
210 READ A
219 REM"white back row
220 B(I%,1)=A : C(I%,1)=3
229 REM black back row
230 B(I%,8)=A
240 NEXT I%
249 REM"draw each square including any piece on the square
250 FOR I%=1 TO 8
260 FOR J%=1 TO 8
270 PROCsquare(I%,J%,1)
280 NEXT J%
290 NEXT I%
299 REM input players moves in turn
300 FOR N%=1 TO 100
310 PROCinput("WHITE") : N$(0,N%)=N$ : N$(1,N%)="-----"
320 PROClist
330 PROCinput("BLACK") : N$(1,N%)=N$
340 PROClist
350 NEXT N%
360 DATA 1,2,3,4,5,3,2,1
370 END

400 REM flash
409 REM IN : coordinates of start and end of move
        OUT: A$="Y" if move accepted
410 DEF PROCflash(X1,Y1,X2,Y2)
420 COLOUR 128 : COLOUR 3
429 REM"give player a chance to check input and/or reconsider move
430 PRINT TAB(0,30);SPC(39);TAB(10,30);"ACCEPT (Y,N) ? "
439 REM"flash two squares involved while waiting for reply
440 REPEAT
450 PROCsquare(X1,Y1,0) : PROCsquare(X2,Y2,0)
```

```
 460 A$=INKEY$(20) : IF A$<>"" THEN 490
 470 PROCsquare(X1,Y1,1) : PROCsquare(X2,Y2,1)
 480 A$=INKEY$(20)
 490 UNTIL A$<>""
 500 IF A$<> "Y" THEN 530
 509 REM if move is accepted then put  piece in new position and
         erase at old position
 510 B(X2,Y2)=B(X1,Y1) : C(X2,Y2)=C(X1,Y1)
 520 B(X1,Y1)=0 : C(X1,Y1)=0
 530 PROCsquare(X1,Y1,1) : PROCsquare(X2,Y2,1)
 540 ENDPROC

 600 REM square
 609 REM X,Y coordinates of square M<>1 for inverse display
 610 DEF PROCsquare(X,Y,M)
 620 LOCAL I%,J%
 630 P=B(X,Y) : C=C(X,Y)
 640 PRINT TAB(X*3+5,(8-Y)*3+3);
 650 IF M=1 THEN COLOUR 129+((X+Y) MOD 2) ELSE COLOUR 128+C
 660 IF M=1 THEN COLOUR C ELSE COLOUR 1+((X+Y) MOD 2)
 669 REM if square has no piece on it then output blanks
 670 IF P=0 THEN VDU 32,32,32,10,8,8,8,32,32,32,10,8,8,8,32,32,32
     : GOTO 710
 680 J%=&A0+9*(P-1)
 689 REM for each row of square output the 3 characters of piece
         then go down and back three spaces
 690 FOR I%=1 TO 3 : VDU J%,J%+1,J%+2,10,8,8,8
 700 J%=J%+3 : NEXT I%
 710 ENDPROC
 800 REM input
 810 DEF PROCinput(B$)
 819 REM prompts one of players to enter move returns N$ with
         accepted move
 820 REPEAT
 830 COLOUR 128 : COLOUR 3
 840 PRINT TAB(0,30);SPC(39);TAB(0,30);B$;"'S MOVE NO. ";N%;" :
";
 850 OK=TRUE : INPUT N$
 859 REM"check input is valid
 860 IF LEN(N$)<>5 THEN OK=FALSE
 870 F$=LEFT$(N$,1)
     : IF F$<"A" OR F$>"H" THEN OK=FALSE ELSE X1=ASC(F$)-64
 880 F$=MID$(N$,2,1)
     : IF F$<"1" OR F$>"8" THEN OK=FALSE ELSE Y1=VAL(F$)
 890 F$=MID$(N$,4,1)
     : IF F$<"A" OR F$>"H" THEN OK=FALSE ELSE X2=ASC(F$)-64
 900 F$=MID$(N$,5,1)
     : IF F$<"1" OR F$>"8" THEN OK=FALSE ELSE Y2=VAL(F$)
 910 UNTIL OK
 920 N$=LEFT$(N$,2)+"-"+RIGHT$(N$,2)
 929 REM offer move for acceptance
 930 PROCflash(X1,Y1,X2,Y2)
 940 IF A$="Y" THEN ENDPROC ELSE 820

1000 REM list
1010 DEF PROClist
1020 COLOUR 128 : COLOUR 3
1029 REM show last 21 moves made
1030 IF N%<20 THEN J%=0 ELSE J%=N%-20
1040 FOR I%=1 TO 21
1050 PRINT TAB(0,I%+5);N$(0,I%+J%);TAB(35,I%+5);N$(1,I%+J%);
1060 NEXT I%
1070 ENDPROC
```

We have seen how characters created in this way can be printed in different colours, however we can still have only two colours in any one character-sized area. There are obvious advantages if we had the ability to print multi-coloured characters quickly at any position on the screen. To do this we must have a small machine-code routine which takes the data that relate to such a ‘character’ (with multi-colour information already encoded, rather than the two-colour format) and transfers them to the screen. We therefore need to understand the way in which the memory positions of the screen are arranged. As we saw in chapter 1, when a mode is first selected the top left-hand corner of the screen is represented by the first byte of the screen memory. The next seven bytes represent the lines vertically below this initial byte. This makes it very easy to place the data for a two-colour character into the correct display locations. The byte at location HIMEM + 8 is horizontally adjacent to the first byte (that is, HIMEM) and is then underscored by the next seven bytes and so on along the row of characters. When we get to the end of the row the next location is the top of the first character on the second line. The complete screen memory is organised in rows in this way, from top (row 0) to bottom (row 31), and from left to right within each row. In a two-colour mode this means that the display holds the information for the first character followed by the second set of data etc., through all the characters displayed in order. Of course in a four-colour mode two columns of bytes will be required to hold the data for one character (left half and right half) and in the sixteen-colour mode we need four columns (far left, middle left, middle right and far right), but this is still fairly simple to calculate since the next column of bytes is always the next eight bytes in the memory.

Simple! It would be if the screen stayed still; furthermore what we have said above is not strictly true. When the screen scrolls up (or down) by one row some odd things happen. The BBC micro uses *hardware scrolling*. This means that the computer can decide which of the 32 (or so) rows in the memory corresponds to the top of the screen (it need not be row 0), and the rest follow in order down the screen with line zero following line thirty-one. When the display scrolls up (or down) it is a simple matter of redefining which row in the memory is the top line of the screen, and the line that disappears from the top (or bottom) is blanked out by the operating system and reallocated to the bottom (or top). This is much faster than *software scrolling*, where rows on the screen have a fixed correspondence with areas of memory, and a program is needed to move all the data from row 1 to row 0, row 2 to row 1 etc. until all the data from the bottom line has been copied to the line above so that the bottom line is ready for use again.

We shall avoid scrolling the screen since this will only confuse our calculations of positions (unless we carefully count the number of scrolling movements and allow for them), and instead make a new print routine which we shall call ‘prynt’. This routine (listing 5.8) is loading with PAGE = &3000 to keep it out of harm’s way, and the machine code it generates is placed at the locations that follow &2300 and stored in backing store as file PRYNT. If we now ‘prynt’ off the bottom of the screen the information will *wrap around* and reappear at the top. The assembly language program (listing 5.7) uses the memory locations from &2400 onwards to store tables of multi-colour character definitions. The characters will have code numbers between 32 and 127 and may be accessed in a program by referring to the standard character with the same ASCII code. The program is designed to print characters in any of modes 0, 1 or 2, so it must know how many bytes make up the character. This is passed into the routine by the variable A% (which is transferred to the A-register) which should be set to either 8, 16 or 32 for the respective modes. If you use the wrong value of A% for a given mode you can produce half-width or double width characters printing. This can be very useful if you want to print half width ‘thin’ numbers in mode 2 or double-width jet-plane characters for games in mode 1. The ‘prynt’ position for the character is calculated from the values of X% and Y%. These are equivalent to the TAB(X, Y) values of the PRINT command. In order to ‘prynt’ a string of such characters we generate a string (B$ say) of normal ASCII characters with codes corresponding to the characters to be drawn, and then call the routine:

CALLprynt, B$

Note that the values of X% and Y% will not change unless they are reset in subsequent ‘prynt' s.

Listing 5.8

```
 10 REM assembly code for prynt
        assign names to locations for use by the routine
 19 REM table of screen line starts
 20 HI=&23E0
 29 REM data pointer for characters
 30 DLO=&80 : DHI=&81
 39 REM screen pointer for position
 40 SLO=&82 : SHI=&83
 49 REM pointer to string info
 50 PLO=&84 : PHI=&85
 59 REM address of string
 60 ALO=&86 : AHI=&87
 69 REM no. of chars. : temp y store
 70 NUM=&88 : TY=&89
 79 REM print position for string
 80 TABX=&8A : TABY=&8B
 89 REM size of character in bytes
 90 SIZE=&8C
 99 REM data table address hi byte
100 TABLE=&8D
109 REM length of line in chars.
110 WIDE=&8E
120 VDU 14
130 FOR O%=0 TO 3 STEP 3
140 P%=&2300
```

```
150 [
160 OPT O%    ;Set char size
170 STA SIZE  ;from A% and
180 STX TABX  ;print position
190 STY TABY  ;from X% and Y%
200 LDA &600  ;Check only one
210 CMP #1    ;parameter in
220 BNE BAD   ;call or error.
230 LDA &603  ;Check string
240 CMP #129  ;variable
250 BNE BAD   ;or error.
260 LDA &601  ;Copy pointer
270 STA PLO   ;to variable
280 LDA &602  ;into zero page
290 STA PHI   ;pointer.
300 LDY #0    ;Initialise Y
310 STY TY    ;for indirects.
320 LDA (PLO),Y
330 STA ALO   ;Copy address
340 INY       ;of string from
350 LDA (PLO),Y
360 STA AHI   ;variable data.
370 INY       ;Move pointer
380 INY       ;to get length
390 LDA (PLO),Y
400 STA NUM   ;of string.
410 JSR START ;Initialise mode
420 .OUT
430 LDY TY    ;output loop
440 LDA (ALO),Y
450 INY       ;move pointer
460 STY TY    ;store counter
470 JSR CHAR  ;prynt character
480 DEC NUM   ;last character?
490 BNE OUT   ;no, do next one
500 RTS       ;end of prynt.
510 .BAD
520 BRK       ;Error section
530 ]
540 ?P%=99:$(P%+1)="Misprynt":P%=P%+10
550 [OPT O%
560 BRK       ;end of error.
570 .CHAR
580 LDX #0    ;prynt one char.
590 STX SHI   ;use SHI and SLO
600 STA SLO   ;multiply ascii
610 JSR MULT  ;to get offset
620 LDA SHI   ;move result
630 CLC       ;to data pointer
640 ADC TABLE ;adding in
650 STA DHI   ;start of table
660 LDA SLO   ;to complete
670 STA DLO   ;data address
680 LDA TABX  ;multiply x
690 STX SHI   ;to get offset
700 STA SLO   ;from start of
710 JSR MULT  ;line.
720 LDX TABY  ;add lo-byte
730 TXA       ;for screen line
740 ROR A     ;odd lines have
750 BCC EVEN  ;&80 extra for
760 LDA SLO   ;their address
```

```
 770 CLC        ;mod 256.
 780 ADC #&80   ;even line is
 790 STA SLO    ;zero lo-byte
 800 .EVEN
 810 LDA HI,X   ;Get hi-byte
 820 ADC SHI    ;of screen line
 830 STA SHI    ;= screen point
 840 LDX SIZE   ;Move X bytes
 850 LDY #0     ;of data to
 860 .TRANS
 870 LDA (DLO),Y
 880 STA (SLO),Y
 890 INY        ;screen in
 900 DEX        ;loop.
 910 BNE TRANS ;When done add
 920 INC TABX   ;one to x pos.
 930 LDA TABX   ;and check for
 940 CMP WIDE   ;end of line
 950 BCC OK     ;If gone over
 960 STX TABX   ;zero x pos.
 970 INC TABY   ;and add to
 980 LDA TABY   ;y pos. If y
 990 CMP #32    ;is off bottom
1000 BCC OK     ;then put it
1010 STX TABY   ;back to top.
1020 .OK
1030 RTS        ;end of char.
1040 .MULT
1050 LDA SIZE   ;Multiply by
1060 CMP #8     ;appropriate
1070 BEQ M8     ;amount for
1080 CMP #16    ;size of data
1090 BEQ M16    ;block in use
1100 ASL SLO    ;Multiply
1110 ROL SHI    ;shi,slo by 2
1120 .M16
1130 ASL SLO    ;times 2 again
1140 ROL SHI
1150 .M8
1160 ASL SLO    ;times 2
1170 ROL SHI    ;three more
1180 ASL SLO    ;times
1190 ROL SHI
1200 ASL SLO
1210 ROL SHI
1220 RTS        ;end of mult.
1230 .START
1240 LDA SIZE   ;get values of
1250 LDX #&2C   ;table hi-byte
1260 LDY #80    ;line length
1270 CMP #8     ;which are
1280 BEQ NSTAR ;appropriate
1290 LDX #&28   ;to size of
1300 LDY #40    ;character
1310 CMP #16    ;being used
1320 BEQ NSTAR ;and store
1330 LDX #&20   ;for future
1340 LDY #20    ;reference
1350 .NSTAR
1360 STX TABLE ;table hi-byte
1370 STY WIDE   ;width of line
1380 RTS        ;end of start.
```

```
1390 ]
1400 NEXT O%
1409 REM construct table of hi-bytes for screen line adresses
1410 M%=&3000
1420 FOR I%=0 TO 31
1430 M=M%+640*I%
1440 HI?I%=M DIV 256
1450 NEXT I%
1459 REM save code after assembly
1460 *SAVE PRYNT 2300 +100
```

We now give ‘CHARACTER GENERATOR 2’ (listing 5.9), which is an elementary character editor that uses routine ‘prynt’ from file PRYNT. 1t has been deliberately kept short so that it can operate normally on all operating systems, however its size is such that it has to be loaded after the PAGE has been reset to &1100. The selection of a mode automatically sets the screen to display a grid of the correct relative dimensions for that mode, and gives you options equivalent to those of ‘CHARACTER GENERATOR 1’. The available conimands, which are called by typing their initial letter, are detailed below.

Pack and Unpack: as before these will display or store a character.
Save and Load: these are the same as options (4) and (5) above, but are now used to save or load the complete character data area (&2400 to &2FFF).

The choice of colours is made by typing the single hexadecimal digit of the required logical colour (for example, press A for colour 10). We are naturally limited to the number of logical colours that are available in any one mode, However there is nothing to stop us reassigning the logical-actual colour relationships within a program that uses these characters.

As before the cursor keys are used to move around the grid and the space bar is used to colour in squares.

From the assembler listing 5.8 we can see that the table of characters apparently starts from either &2000 (for the 32-byte long characters), &2800 for the 16-byte characters) or &2C00 (for the 8-byte characters). 1n fact these are the positions where the data for the character with code 0 would have been found, although the real table does not start until the data for the character that is coded 32 is reached. 1n this way it is not necessary to subtract 32 from the ASCII-equivalent code for each character in order to find its position in the table.

Example 5.6

Listing 5.10 will allow you to type in redeHned characters directly from the keyboard (stricly speaking you type the ASCII-equivalent characters) and ‘prynt’ them on the screen. First you need to create some characters in order to use this program. The two parts of figue 5.4 (a and b) are ‘painting-by-numbers’ charts (in hexadecimal) for two characters in mode 2. Use ‘CHARACTER GENERATOR 2’ to create the equivalent multi-coloured characters. Save

Listing 5.9

```
 10 DIM B(32),T(2),S(7,7),R(7,7) : T(0)=&2C00 : T(1)=&2800 :
                                   T(2)=&2000
 20 DIM H(23):FOR I=0 TO 23:READ H(I) : NEXT I
 30 DATA 0,1,4,5,16,17,20,21,64,65,68,69,80,81,84,85
         ,&88,&44,&22,&11,0,1,16,17
 40 INPUT "MODE ",M : MODE M
 50 A%=2^(M+3) : B%=2^(2^M) : D%=2^(M+1) : M%=2^(3-M)
 60 SX=20*D%
 70 COLOUR 135 : CLS
 80 FOR I=0 TO 7 : FOR J=0 TO 7
 90 PROCsquare(I,J,0)
100 NEXT J : NEXT I
110 LX=640-4*SX-D%:RX=LX+SX*8:BY=196:TY=838
120 GCOL 0,0 : MOVE LX,BY : DRAW LX,TY : DRAW RX,TY
    : DRAW RX,BY : DRAW LX,BY
130 I=0 : J=0 : C=0 : PROCmark(I,J)
140 *FX4,1
150 REPEAT : PROCmark(I,J)
160 IF INKEY(-42) AND J>0 THEN J=J-1
170 IF INKEY(-58) AND J<7 THEN J=J+1
180 IF INKEY(-26) AND I>0 THEN I=I-1
190 IF INKEY(-122) AND I<7 THEN I=I+1
200 IF INKEY(-99) THEN PROCsquare(I,J,C)
210 PROCmark(I,J)
220 COLOUR 0
230 A$=INKEY$(0) : IF A$="" THEN 350
240 IF VAL(A$)=0 AND A$<>"0" AND (A$<"A" OR A$>"F") THEN 270
250 C=EVAL("&"+A$)
260 PRINT TAB(0,0);"COLOUR ";~C; : COLOUR C : PRINT" **"
270 IF A$="S" THEN PROCfile("SAVE")
280 IF A$="L" THEN PROCfile("LOAD")
290 IF A$<>"P" AND A$<>"U" THEN 350
300 REPEAT : PRINT TAB(0,29);SPC(39);TAB(0,29);
310 INPUT"Which character ",CHAR
320 UNTIL CHAR>31 AND CHAR <127
330 IF A$="U" THEN PROCunpack : PROCmark(I,J)
340 IF A$="P" THEN PROCpack
350 UNTIL A$="Q"
360 *FX4,0
370 END

400 REM square
410 DEF PROCsquare(X,Y,IN) : GCOL 0,IN
420 LX=640+SX*(X-4):RX=LX+SX-D%*2:BY=Y*80+200:TY=BY+72
430 MOVE LX,BY : MOVE LX,TY : PLOT 85,RX,BY : PLOT 85,RX,TY
440 S(X,Y)=IN : PLOT 69,1031+D%*X,960+4*Y
450 ENDPROC

500 REM mark
510 DEF PROCmark(X,Y) : GCOL 3,7
520 LX=650+SX*(X-4) : RX=LX+SX-30 : BY=Y*80+210 : TY=BY+52
530 MOVE LX,BY : DRAW RX,TY : MOVE RX,BY : DRAW LX,TY
540 ENDPROC

600 REM pack
610 DEF PROCpack
620 FOR X=0TO7 : FORY=0TO7
```

```
 630 B=8*(X DIV M%)+7-Y
 640 ON M+1 GOTO 650,660,690
 650 N%=2^(7-X) : B(B)=B(B) AND (255 EOR N%) OR N%*S(X,Y) : GOTO
700
 660 HI=X MOD 4 : B(B)=B(B) AND (255 EOR H(HI+16))
 670 B(B)=B(B) OR H(S(X,Y)+20)*2^(3-HI)
 680 GOTO 700
 690 HI=2-(X MOD 2):B(B)=B(B) AND (&AA/HI) OR (H(S(X,Y))*HI)
 700 NEXT Y : NEXT X
 710 MEM=T(M)+CHAR*A%
 720 FOR I%=0 TO A%-1 : MEM?I%=B(I%) : NEXT I%
 730 ENDPROC

 800 REM unpack
 810 DEF PROCunpack
 820 MEM=T(M)+CHAR*A%
 830 FOR I%=0 TO A%-1 : B(I%)=MEM?I% : NEXT I%
 840 FOR B=0 TO A%-1
 850 Y=7-(B MOD 8)
 860 ON M+1 GOTO 870,890,920
 870 FOR X=0 TO 7 : IF B(B) AND 2^(7-X) THEN S(X,Y)=1 ELSE S(X,Y)=0
 880 NEXT X : GOTO 950
 890 FOR I%=0 TO 3 : X=I%+4*(B DIV 8)
     : HI=(B(B) AND H(16+I%))/2^(3-I%)
 900 S(X,Y)=(HI AND &F0)/8+(HI AND &F)
 910 NEXT I% : GOTO 950
 920 FOR I%=1 TO 2 : X=2-I%+2*(B DIV 8) : HI=(B(B) AND (&55*I%))/I%
 930 J%=-1 : REPEAT : J%=J%+1 : UNTIL H(J%)=HI
 940 S(X,Y)=J% : NEXT I%
 950 NEXT B
 960 FOR I%=0 TO 7
 970 FOR J%=0 TO 7
 980 PROCsquare(I%,J%,S(I%,J%))
 990 NEXT J% : NEXT I%
1000 ENDPROC

1100 REM file
1110 DEF PROCfile(B$)
1120 PRINT TAB(0,29);
1130 INPUT "WHAT FILE NAME",A$:IFA$=""THEN ENDPROC
1140 A$=B$+" "+A$+" 2400" : IF B$="SAVE"THEN A$=A$+" 3000"
1150 $&3000=A$ : X%=0 : Y%=&30 : CALL&FFF7
1160 ENDPROC
```

these (and any other characters) on tape or disk, and then use the program from listing 5.9 to reload the file and display the characters on the screen. If you are feeling fit and healthy you can fill the screen with the callisthenic character from figure 5.4a and then join in.

You will find that this type of character is used in the video game in chapter 15.

F	0	0	7	7	0	0	F
0	F	0	7	7	0	F	0
8	8	7	7	7	7	8	8
0	0	7	7	7	7	0	0
0	0	7	7	7	7	0	0
0	0	7	0	0	7	0	0
0	F	8	0	0	8	F	0
F	0	8	0	0	8	0	F

0	0	0	0	0	0	0	0
0	0	6	6	6	6	0	0
0	6	8	F	8	F	6	0
6	6	6	6	6	6	6	6
6	6	6	9	E	6	6	6
6	6	6	6	6	6	6	6
6	6	6	6	6	6	6	6
6	0	6	0	6	0	6	0

(a) (b)

Listing 5.10

```
   9 REM program to test PRYNT routine
  10 *LOAD PRYNT 2300
  20 INPUT"MODE",M : MODE M : A%=2^(3+M)
  30 WIDE=80/2^M : PRYNT=&2300 : HIMEM=&2300
  39 REM" load in a set of characters
  40 PROCfile("LOAD")  : X%=0 : Y%=0
  49 REM PRYNT any characters typed and alter prynt position
  50 REPEAT : A$=GET$
  60 CALL PRYNT,A$
  70 X%=X%+1 : IF X%=WIDE THEN Y%=(Y%+1) MOD 32 : X%=0
  80 UNTIL FALSE

1100 REM file
1110 DEF PROCfile(B$)
1120 PRINT TAB(0,29);
1130 INPUT "WHAT FILE NAME",A$ : IF A$="" THEN ENDPROC
1140 A$=B$+" "+A$+" 2400" : IF B$="SAVE"THEN A$=A$+" 3000"
1150 $&3000=A$ : X%=0 : Y%=&30 : CALL&FFF7
1160 ENDPROC
```

In the next chapter we shall consider how character graphics and our knowledge of two-dimensional geometry can be combined to form data displays for use in the office or the laboratory.

Complete Programs

I Listing 5.1. Data required: any character. Try #.

II Listing 5.2. Data required: a data mode (try 1), a character (#) and background and foreground colours (try 1 and 2).

III Listing 5.3. Data required: an ASCII code (try 128) and eight 8-bit binary numbers (try 10101010, 01010101 , 10101010, 01010101, 10101010, 01010101, 10101010 and 01010101).

IV Listing 5.4. No data required.

V Listing 5.5. Use listing 5.3 to create characters with ASCIIcodes 128, 129, and 131 before running the program. 1t requires actual background and foreground colours (try 1 and 2).

VI Listing 5.6 and PAGE set to &1400 for OS 0.1 and &1F00 for OS > 1.0. Use option (3) of the CHARACTER GENERATOR 1 to create characters 128, 129, 130 and 131 and run option (6) (listing 5.5). Type 1 (to specify that you wish to draw in white) and move the cross around the screen with the cursor keys: the space bar will colour in a pixel (type 0 to plot pixels in black). Pack the characters in codes 128 etc.

VII Listing 5.7 loaded at PAGE = &1400 (OS 0.1) or &1F00 (OS ≥ 1 .0). You may have to strip it of REMarks. Characters must be created in blocks CHESSP1 and CHESSP2 on backing store. The chess program needs each move of the game to be specified (such as E2 to E4) and the question

Accept? to be answered as Yes or No.

VIII Listing 5.8 and 5.9. Run listing 5.8 with PAGE = &3000, which saves a file PRYNT for future use. Listing 5.9, PAGE = &1100, calls for a mode (try 2); then move the cross about the screen with the cursor keys, add colours with the space bar, and change colour by typing a hexadecimal digit (0 to F). Pack the new characters into locations equivalent to the ASC1l codes 32 to 126. When you have generated the required symbols Save a file called TESTC. Try placing new characters in locations equivalent to ASCII 32 (normally a space), 33 (!), 34 (") and 35 (#).

IX Listing 5.10 with PAGE = &1100. This needs to load PRYNT. Then type on the keyboard 1, # etc. and watch the equivalent character being drawn.