The Dragon 32

Dragon Companion

M. Jarvis

CONTENTS

3	FOREWORD
6	LOW RESOLUTION MODE 4
8	LOW RESOLUTION MODE 6
10	LOW RESOLUTION MODE 8
12	LOW RESOLUTION MODE 12
14	LOW RESOLUTION MODE 24
17	64 BY 64 FOUR COLOUR
18	128 BY 64 TWO COLOUR
19	128 BY 64 FOUR COLOUR
20	128 BY 96 TWO COLOUR
21	128 BY 96 FOUR COLOUR
22	128 BY 192 TWO COLOUR
23	128 BY 192 FOUR COLOUR
24	256 BY 192 TWO COLOUR
25	VDG MEMORY MAPPING AND MODE SELECTION
26	VIDEO MEMORY BASE-PAGE LOCATION
27	PROCESSOR SPEEDS
28	LOADING MORE THAN ONE PROGRAM FROM TAPE
29	A 68Ø9 DISASSEMBLER
33	USEFUL ROUTINES IN THE BASIC ROM
35	BASIC STORAGE
36	EASIER INPUT OF MACHINE CODE
37	ADDING COMMANDS TO BASIC
39	TOKEN TABLE 1
40	TOKEN TABLE 2
41	CHARACTER TABLE
43	NOTES ON CHARACTER TABLE
44	EXTRA INFO ON SAM CONTROL BIT AREA
45	PIAs
47	CARTRIDGE EDGE CONNECTOR PIN DESCRIPTION
48	SOUND OUTPUT SELECTION
49	NOTES 1. GRAPHICS MODES
49	2. PROCESSOR SPEEDS
50	MEMORY MAP

Foreword

The Dragon 32 is much more versatile and powerful than the manual supplied with the machine would suggest. I was amazed to find extra graphics modes hidden within the computer which appear to have been completely ignored by the manufacturers. It is a simple task to access these extra modes, but far from simple to use them. In order to utilise them fully a set of graphics commands would have to be written (preferably in machine code) which would provide the same sort of commands already in Basic eg CIRCLE, SET etc. As you can imagine this is not a trivial exercise and would no doubt form the basis of another book.

If speed is not important then the Basic POKE and PEEK commands can be used to give a little variety to your graphics programs using the simple driver routines included in this book. Nothing like moving graphics will be possible though because of the time involved in working through a whole screen full of data (try running the Low Resolution Mode 24 routine).

This situation can be improved slightly by increasing the processor clock rate as described later in the Dragon Companion Book. (Try the same Low Resolution Mode 24 routine again but this time at the faster clock rate).

When you have finished experimenting with graphics using Basic you might like to look at the way the Basic interpreter is written. The Disassembler program gives you the means of seeing what a really large machine language program looks like. With it you can disassemble the Basic rom and search for the many useful subroutines which must be hiding in there but have not yet been discovered. It may be possible, for instance, to find some graphics primitives which would greatly improve the speed of the extra graphics modes already described.

When you have found these routines or written them yourself, you can add commands to Basic and so extend the language in whichever way you like. The possibilities are bounded only by your imagination and expertise.

For the less adventurous there is a simple way of building up a library of useful subroutines which can be stored on tape and added to programs when needed. The method described here in this book enables you to add a routine from tape to the end of a program already in memory, ie the use of CLOAD need not destroy what is already there in the machine.

I hope this book provides you with the sort of information you need and that you enjoy reading it—I have certainly enjoyed researching and writing it.

Extra Graphics Modes

The Dragon uses a Motorola 6847 video-display-generator to control the colour graphics and text screens. It is a versatile chip capable of some amazing displays but unfortunately is not fully utilised by the version of Basic on the machine. According to the manual supplied with the Dragon there are five graphics modes Plus the text and low resolution graphics modes available to the user giving seven in all. There are in fact another seven modes hidden in the machine which are really quite easy to manipulate even without the sophisticated commands provided through Basic.

We can take control of the VDG by poking values into the SAM bits (synchronous address multiplexer) towards the top of the Dragon's memory. The mode is selected through location 65314 and the memory mapping for that mode is governed by locations 65472 to 65477. The mode selection procedure is summarised at the end of this section.

The following pages describe each mode in some detail and can be used as reference guides.

TEXT MODE

There is a one to one correspondence between text screen locations and video memory locations which means that characters can be poked onto the screen. The character set is built into the VDG and has one or two peculiarities. Firstly, there are no lower case characters (except when using a Printer)—you choose between inverted and non-inverted text. Secondly, the code used to represent the characters is a modified version of the ASCII standard. The characters and their codes are shown in character table 1. Each character is represented by an eight bit byte the most significant bit of which is always zero. The second most significant bit governs whether the character is inverted or not.

The text screen is divided into 512 locations (16 rows of 32 columns) and its normal address space is from 1Ø24 to 1535.

LOW RESOLUTION GRAPHICS MODES

I have used the name 'low resolution' to signify that these modes rely on dividing each character position on the screen into a number of elements called pixels (picture elements). Each mode is numbered according to the number of pixels to each character position. Only one low resolution graphics mode is implemented in the Microsoft Basic on the Dragon but there are in fact five modes available.

LOW RESOLUTION MODE 4

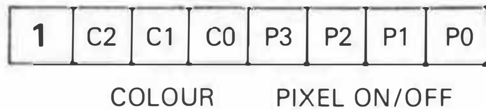
This is the only low resolution mode supported by Basic and is normally manipulated by SET, RESET etc. Text and graphics characters can be freely mixed on the screen in this mode. Graphics characters are distinguished from text characters by setting the most significant bit to one. In this mode any byte with bit 7 set to a one is interpreted by the VDG as a graphics character.

Each byte is divided into a number of sections which govern the colour for the 'on' pixels and also which pixels will be 'on'.

ONE CHARACTER

PIXEL 3	PIXEL 2
PIXEL 1	PIXEL 0

ONE BYTE



A one in any of the lower four bits will light that pixel on the screen to the colour indicated by the code in bits 4, 5 and 6.

All eight colours can be specified ('off' pixels are always black). The colours are selected using the following code:—

000 GREEN
001 YELLOW
010 BLUE
011 RED
100 BUFF
101 CYAN
110 MAGENTA
111 ORANGE

There are 512 character positions on the screen and as each character can be divided into 4 pixels this mode has a resolution of 64 by 32.

This, together with the text mode, is the default one which is selected whenever a return to Basic command level is made. Should you wish to select the mode other than through the commands provided in Basic the example program shows how to go about it.

```
1000 REM
1010 REM LOW RES. MODE 4
1020 REM
1030 REM SET UP VIDEO MEMORY MAPPING FOR MODE4
1040 POKE 65472,1   POKE 65474,1   POKE 65476,1
1050 REM SELECT THE MODE
1060 POKE 65314,0
1070 FOR I=0 TO 511
1080 X=128+RND(127) ' RANDOM VALUE FOR PIXELS AND COLOURS
1090 REM PUT THE PIXELS RANDOMLY ON THE SCREEN
1100 POKE 1024+RND(511),X
1110 NEXT I
1120 FOR T=1 TO 500   NEXT T ' SMALL DELAY
1130 FOR I=0 TO 15 ' NUMBER OF ROWS
1140 FOR J=0 TO 31 ' NUMBER OF COLUMNS
1150 REM HORIZONTAL LINES
1160 POKE 1024+I*32+J,(128 + RND(127))
1170 NEXT J,I
1180 FOR T=1 TO 500   NEXT T ' SMALL DELAY
1190 FOR I=0 TO 15 ' ROWS
1200 X=128+RND(127) ' VALUE FOR PIXELS AND COLOURS
1210 FOR J=0 TO 31 ' COLUMNS
1220 REM THIS TIME THE WHOLE ROW WILL BE THE SAME CHARACTER
1230 POKE 1024+I*32+J,X
1240 NEXT J,I
1250 FOR T=1 TO 500   NEXT T ' SMALL DELAY
1260 REM THIS TIME FILL THE SCREEN WITH VERTICAL LINES
1270 FOR J=0 TO 31 ' COLUMNS
1280 FOR I=0 TO 15 ' ROWS
1290 X=128+RND(127) ' VALUE FOR PIXELS AND COLOURS
1300 POKE 1024+I*32+J,X ' PUT IT ON THE SCREEN
1310 NEXT I,J
1320 FOR T=1 TO 500   NEXT T ' LAST DELAY
1330 GOTO1070
```

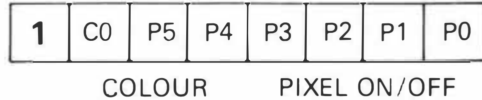
LOW RESOLUTION MODE 6

Each character position in this mode is divided into six pixels and one byte of memory is used to govern both colour and whether they are lit. The increase in resolution without using more memory than mode 4 is paid for by cutting down on the number of colours available.

Each byte is divided as follows:—

ONE CHARACTER

PIXEL 5	PIXEL 4
PIXEL 3	PIXEL 2
PIXEL 1	PIXEL 0



Notice that the colour defining field is bit 6. This means that only 2 colours can be selected but to make things a bit better the possible colours are divided into two groups. Each group is selected by toggling bit 4 in address 65314. A zero in this position selects blue or red while a one selects magenta or orange.

Colour codes for the field in the video memory bytes are:—

Ø MAGENTA or BLUE
1 ORANGE or RED

Mode 6 has a resolution of 64 by 48.

```
1000 REM
1010 REM LOW RES. MODE 6
1020 REM
1030 INPUT "COLOUR SET (1/2) "; S
1040 IF S=l THEN C=8 ELSE IF S=2 THEN C=0 ELSE GOTO 1030
1050 REM SET UP VIDEO MEMORY MAPPING
1060 POKE 65472,1 : POKE 65474,1 : POKE 65476,1
1070 REM SELECT THE MODE
1080 POKE 65314,16+C
1090 REM HORIZONTAL LINES
1100 FOR I=0 TO 15 STEP 2
1110 LI=32*I
1120 FOR J=0 TO 31 STEP 2
1130 POKE 1024+LI+J,25+128
1140 POKE 1024+LI +J+1, 25+128+64
1150 NEXT J, I
1160 FOR I=1 TO 15 STEP 2
1170 LI = 32*1
1180 FOR J=0 TO 31 STEP 2
1190 POKE 1024+LI+J,38+128
1200 POKE 1024+LI +J+1, 38+128+64
1210 NEXT J, I
1220 FOR T=1 TO 500 : NEXT T: ' SMALL DELAY
1230 REM VERTICAL LINES
1240 FOR J=0 TO 31
1250 FOR I=0 TO 15
1260 LI=32*I
1270 POKE 1024+LI+J,128+RND(127)
1280 NEXT I,J
1290 FOR T=l TO 500 NEXT T:' SMALL DELAY
1300 GOTO 1100
```

LOW RESOLUTION MODE 8

The space corresponding to one character position on the text screen is divided into eight pixels in this mode. It is easier to visualise the effects of mode 8 if we now think about rows and columns. The resolution in this mode is 64 by 64 pixels with each byte of video memory controlling two adjacent pixels on the same row. This requires 2048 bytes to control the whole screen and enables all eight colours to be specified (the two pixels controlled by the same byte obviously being the same colour).

The fields within each byte are: —

ONE CHARACTER

PIXEL 7	PIXEL 6
PIXEL 5	PIXEL 4
PIXEL 3	PIXEL 2
PIXEL 1	PIXEL 0

FOUR BYTES

1	C2	C1	C0	P7	P6	XX	XX
1	C2	C1	C0	P5	P4	XX	XX
1	C2	C1	C0	XX	XX	P3	P2
1	C2	C1	C0	XX	XX	P1	P0

COLOUR　　PIXEL ON/OFF

Colour codes are the same as for mode 4, ie.

000 GREEN
001 YELLOW
010 BLUE
011 RED
100 BUFF
101 CYAN
110 MAGENTA
111 ORANGE

```
1000 REM
1010 REM LOW RES. MODE 8
1020 REM
1030 REM SET UP VIDEO MEMORY MAPPING
1040 POKE 65472,1 : POKE 65475,1 : POKE 65476,1
1050 REM SELECT THE MODE
1060 POKE 65314,0
1070 REM FILL SCREEN IN 4 PASSES
1080 FOR I=0 TO 63 STEP 4
1090 ROW = I*32
1100 FOR J=0 TO 31
1110 C = RND(7)*16
1120 POKE 1024+ROW+J,128+C+8
1130 NEXT J,I
1140 FOR I=1 TO 63 STEP 4
1150 ROW = I*32
1160 FOR J=0 TO 31
1170 C = RND(7)*16
1180 POKE 1024+ROW+J,128+C+4
1190 NEXT J,I
1200 FOR I=2 TO 63 STEP 4
1210 ROW = I*32
1220 FOR J=0 TO 31
1230 C = RND(7)*16
1240 POKE 1024+ROW+J,128+C+2
1250 NEXT J,I
1260 FOR I=3 TO 63 STEP 4
1270 ROW = I*32
1280 FOR J=0 TO 31
1290 C = RND(7)*16
1300 POKE 1024+ROW+J,128+C+1
1310 NEXT J,I
1320 GOTO1320
```

LOW RESOLUTION MODE 12

Each character position in this mode is divided into twelve pixels and, like mode 8, each row of two pixels within a character position is represented by one byte in video memory. Consecutive bytes in video memory represent consecutive double pixels across the screen (again like mode 8) so that the second row within a character position is governed by the byte 32 further on in video memory.

A character position is made up as follows:—

ONE CHARACTER

PIXEL 11	PIXEL 10
PIXEL 9	PIXEL 8
PIXEL 7	PIXEL 6
PIXEL 5	PIXEL 4
PIXEL 3	PIXEL 2
PIXEL 1	PIXEL 0

SIX BYTES

1	C2	C1	C0	P11	P10	XX	XX
1	C2	C1	C0	P9	P8	XX	XX
1	C2	C1	C0	P7	P6	XX	XX
1	C2	C1	C0	XX	XX	P5	P4
1	C2	C1	C0	XX	XX	P3	P2
1	C2	C1	C0	XX	XX	P1	P0

COLOUR PIXEL ON/OFF

As can be seen, it take six bytes to control a single character position. This mode therefore requires 512 × 6 = 3072 bytes of video memory. The colour codes are the same as for modes 8 and 4 giving an eight colour 64 by 96 mode, ie.

000 GREEN
001 YELLOW
010 BLUE
011 RED
100 BUFF
101 CYAN
110 MAGENTA
111 ORANGE

```
1000 REM
1010 REM LOW RES MODE 12
1020 REM
1030 REM SET UP VIDEO MEMORY MAPPING
1040 POKE 65472, 1 : POKE 65474, 1 : POKE 65477, 1
1050 REM SELECT THE MODE
1060 POKE 65314, 0
1070 REM FILL SCREEN IN 6 PASSES
1080 FOR I=0 TO 5
1090 FOR J=I TO 95 STEP 6
1100 ROW = J*32
1110 FOR K=0 TO 31
1120 C = RND(7)*16
1130 IF I<3 THEM N = RND(2)+6 ELSE N = RND(2) 1140 POKE 1024+ROW+K, 128+C+N
1150 NEXT K,J, I
1160 FOR T=l TO 500 NEXT T: ' SMALL DELAY
1170 REM VERTICAL LINES
1180 FOR J=0 TO 31
1190 FOR I=0 TO 95
1200 POKE 1024+1*32+J, 128+RND(127)
1210 NEXT I,J
1220 FOR T=l TO 500 NEXT T:' SMALL DELAY
1230 GOTO 1080
```

LOW RESOLUTION MODE 24

This is the last of the low resolution graphics modes and it divides the character position into twenty four pixels in the usual way ie. 2 by 12. It is an eight colour mode and is very similar to modes 12 and 8 where each row in a character position is represented by bytes thirty two addresses apart. The usual table follows:—

ONE CHARACTER

PIXEL 23	PIXEL 22
PIXEL 21	PIXEL 20
PIXEL 19	PIXEL 18
PIXEL 17	PIXEL 16
PIXEL 15	PIXEL 14
PIXEL 13	PIXEL 12
PIXEL 11	PIXEL 10
PIXEL 9	PIXEL 8
PIXEL 7	PIXEL 6
PIXEL 5	PIXEL 4
PIXEL 3	PIXEL 2
PIXEL 1	PIXEL 0

TWELVE BYTES

1	C2	C1	C0	P23	P22	XX	XX
1	C2	C1	C0	P21	P20	XX	XX
1	C2	C1	C0	P19	P18	XX	XX
1	C2	C1	C0	P17	P16	XX	XX
1	C2	C1	C0	P15	P14	XX	XX
1	C2	C1	C0	P13	P12	XX	XX
1	C2	C1	C0	XX	XX	P11	P10
1	C2	C1	C0	XX	XX	P9	P8
1	C2	C1	C0	XX	XX	P7	P6
1	C2	C1	C0	XX	XX	P5	P4
1	C2	C1	C0	XX	XX	P3	P2
1	C2	C1	C0	XX	XX	P1	P0

COLOUR PIXEL ON/OFF

Here we have an eight colour 64 by 192 mode which requires 6144 bytes of video memory. The usual colour codes apply:—

000 GREEN
001 YELLOW
010 BLUE
011 RED
100 BUFF
101 CYAN
110 MAGENTA
111 ORANGE

```
1000 REM
1010 REM LOW RES. MODE 24
1020 REM
1030 REM SET UP VIDEO MEMORY MAPPING
1040 POKE 65472,1 : POKE 65475,1 : POKE 65477,1
1050 REM SELECT THE MODE
1060 POKE 65314,0
1070 REM FILL SCREEN IN 12 PASSES
1080 FOR I=0 TO 11
1090 FOR J=I TO 191 STEP 12
1100 ROW = J*32
1110 FOR K=0 TO 31
1120 C = RND(7)*16
1130 IF I<6 THEN N = RND(2)+6 ELSE N = RND(2)
1140 POKE 1024+ROW+K,128+C+N
1150 NEXT K,J,I
1160 FOR T=1 TO 500 : NEXT T ' SMALL DELAY
1170 REM FILL SCREEN WITH VERTICAL LINES
1180 FOR J=0 TO 31
1190 FOR I=0 TO 191
1200 POKE 1024+I*32+J,128+RND(127)
1210 NEXT I,J
1220 FOR T=1 TO 500 : NEXT T ' SMALL DELAY
1230 GOTO 1080
```

MIXING TEXT AND GRAPHICS IN THE LOW RESOLUTION MODES

Text can be mixed with modes 8, 12 and 24. The method is quite simply to poke the required ASCII value into *all* bytes governing a particular character position on the screen. This means that 4, 6 and 12 pokes respectively for a single character. The process is slow in Basic but would be acceptable in machine code.

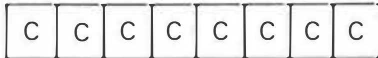
Try the following short example:

```
1000 REM SELECT MODE 24
1010 POKE65472,1:POKE65475,1:POKE65477,1:POKE65314,0
1020 REM SELECT START OF A CHARACTER POSITION
1030 FOR S=&HA0A TO &HA1A
1040 REM POKE SAME VALUE TO ALL BYTES IN THIS CHARACTER POSITION
1050 FOR I=0 TO 32*11 STEP 32
1060 POKE S+I,&H45
1070 NEXT I,S
1080 GOTO 1080
```

The remaining eight modes can be thought of as the true graphics modes in that we no longer think in character positions. Only five of the eight are implemented in Basic. I will describe each mode in turn indicating which are supported by Basic and also how to control all of them without using the special Basic commands.

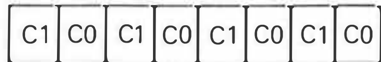
Bit mapping of the display follows the same conventions for each of the eight modes I am about to describe. The modes are either two or four colour and so each pixel can be represented by one or two bits respectively. This means that a single byte can represent eight or four consecutive pixels on the same screen row, depending on whether the mode is two or four colour, eg.

ONE BYTE
TWO COLOUR DISPLAY



Each bit represents a single pixel in one of two colours.

ONE BYTE
FOUR COLOUR DISPLAY



Each set of two bits represents a single pixel in one of four colours.

The colour sets are:

Two-colour. . .BLACK and GREEN or BLACK and BUFF

Four-colour. . .GREEN, YELLOW, BLUE and RED, or BUFF, CYAN, MAGENTA and ORANGE

64 BY 64 FOUR COLOUR MODE

(not supported by Basic)

This mode requires 1024 bytes of video memory to display a full screen. The sample program gives all the information you need in order to select the 64 by 64 four colour mode.

```
1000 REM
1010 REM 64 X 64 FOUR COLOUR MODE
1020 REM
1030 REM SELECT CORRECT VIDEO MEMORY MAPPING
1040 POKE 65473,1 : POKE 65474,1 : POKE 65476,1
1050 REM SELECT 64 X 64 MODE
1060 REM COLOUR SET IS SELECTED BY MAKING C = 0 OR 8
1070 C=0
1080 REM CHANGE COLOUR SET
1090 IF C=0 THEN C=8 ELSE C=0
1100 POKE 65314,128+C
1110 FOR I=0 TO 1023
1120 POKE 1024+I,RND(255)
1130 NEXT I
1140 FOR I=1 TO 1000 : NEXT I
1150 FOR T=1 TO 500 : NEXT T ' SMALL DELAY
1160 REM VERTICAL LINES NOW
1170 FOR J=0 TO 31 STEP 2
1180 FOR I=0 TO 31
1190 REM RANDOM COLOURS
1200 X=RND(255)
1210 REM POKE THEM ONTO THE SCREEN
1220 POKE 1024+I*32+J,X
1230 NEXT I,J
1240 FOR T=1 TO 500 : NEXT T ' ANOTHER DELAY
1250 GOTO 1090
```

128 BY 64 TWO COLOUR MODE

(not supported by Basic)

This mode uses 1024 bytes of video memory and again the example program gives all the information necessary.

```
1000 REM
1010 REM 128 X 64 TWO COLOUR MODE
1020 REM
1030 REM SELECT CORRECT VIDEO MEMORY MAPPING
1040 POKE 65473,1 : POKE 65474,1 : POKE 65476,1
1050 REM SELECT MODE
1060 REM COLOUR SET IS SELECTED BY MAKING C = 0 OR 8
1070 C=0
1080 REM CHANGE COLOUR SET
1090 IF C=0 THEN C=8 ELSE C=0
1100 POKE 65314,128+16+C
1110 FOR I=0 TO 1023
1120 POKE 1024+I,RND(255)
1130 NEXT I
1140 FOR T=1 TO 500 : NEXT T ' SMALL DELAY
1150 REM VERTICAL LINES
1160 FOR J=0 TO 15
1170 FOR I=0 TO 63
1180 REM RANDOM COLOURS
1190 X=RND(255)
1200 POKE 1024+I*16+J,X
1210 NEXT I,J
1230 FOR T=1 TO 500 : NEXT T ' ANOTHER DELAY
1230 GOTO1090
```

128 BY 64 FOUR COLOUR MODE

(not supported by Basic)

Double the number of colours over the 128 by 64 two colour mode means that this mode must have 2048 bytes for a full video screen. A similar demonstration program gives all the information on selecting the mode.

```
1000 REM
1010 REM 128 X 64 FOUR COLOUR MODE
1020 REM
1030 REM SELECT CORRECT VIDEO MEMORY MAPPING
1040 POKE 65472,1 : POKE 65475,1 : POKE 65476,1
1050 REM SELECT MODE
1060 REM COLOUR SET IS SELECTED BY MAKING C = 0 OR 8
1070 C=0
1080 REM CHANGE COLOUR SET
1090 IF C=0 THEN C=8 ELSE C=0
1100 POKE 65314,128+32+C
1110 FOR I=0 TO 2047
1120 POKE 1024+I,RND(255)
1130 NEXT I
1140 FOR T=1 TO 500 : NEXT T ' SMALL DELAY
1150 REM VERTICAL LINES
1160 FOR J=0 TO 31
1170 FOR I=0 TO 63
1180 REM RANDOM COLOURS
1190 X=RND(255)
1200 REM POKE THEM ONTO THE SCREEN
1210 POKE 1024+I*32+J,X
1220 NEXT I,J
1230 FOR T=1 TO 500 : NEXT T ' ANOTHER DELAY
1240 GOTO 1090
```

128 BY 96 TWO COLOUR MODE

(Basic — PMODE Ø)

Memory requirements here are 1536 bytes per screen. This mode is fully supported by Basic with PSET etc. but in order to make the information in this book complete I will give the 'manual' method of selection.

```
1000 REM
1010 REM 128 X 96 TWO COLOUR MODE
1020 REM
1030 REM SELECT CORRECT VIDEO MEMORY MAPPING
1040 POKE 65473,1 : POKE 65475,1 : POKE 65476,1
1050 REM SELECT MODE
1060 REM COLOUR SET IS SELECTED BY MAKING C = 0 OR 8
1070 C=0
1080 REM CHANGE COLOUR SET
1090 IF C=0 THEN C=8 ELSE C=0
1100 POKE 65314,128+32+16+C
1110 FOR I=0 TO 1535
1120 POKE 1024+I,RND(255)
1130 NEXT I
1140 FOR T=1 TO 500 : NEXT T ' SMALL DELAY
1150 REM VERTICAL LINES
1160 FOR J=0 TO 15
1170 FOR I=0 TO 95
1180 REM RANDOM COLOURS
1190 X=RND(255)
1200 REM POKE THEM ONTO THE SCREEN
1210 POKE 1024+I*16+J,X
1220 NEXT I,J
1230 FOR T=1 TO 500 : NEXT T ' ANOTHER DELAY
1240 GOTO 1090
```

128 BY 96 FOUR COLOUR MODE

(Basic — PMODE 1)

3072 bytes per screen are required in this mode because of the doubling of the colours available over the 128 by 96 two colour mode. The same simple program is used to demonstrate calling the mode.

```
1000 REM
1010 REM 128 X 96 FOUR COLOUR MODE
1020 REM
1030 REM SELECT CORRECT VIDEO MEMORY MAPPING
1040 POKE 65472,1 : POKE 65474,1 : POKE 65477,1
1050 REM SELECT MODE
1060 REM COLOUR SET IS SELECTED BY MAKING C = 0 OR 8
1070 C=0
1080 REM CHANGE COLOUR SET
1090 IF C=0 THEN C=8 ELSE C=0
1100 POKE 65314,128+64+C
1110 FOR I=0 TO 3071
1120 POKE 1024+I,RND(255)
1130 NEXT I
1140 FOR T=1 TO 500 : NEXT T ' SMALL DELAY
1150 REM VERTICAL LINES
1160 FOR J=0 TO 31
1170 FOR I=0 TO 95
1180 REM RANDOM COLOURS
1190 X=RND(255)
1200 REM POKE THEM ONTO THE SCREEN
1210 POKE 1024+I*32+J,X
1220 NEXT I,J
1230 FOR T=1 TO 500 : NEXT T ' ANOTHER DELAY
1240 GOTO 1090
```

128 BY 192 TWO COLOUR MODE

(Basic — PMODE 2)

Half the colours available but double the vertical resolution means that this mode has the same memory requirements as the 128 by 96 four colour mode.

```
1000 REM
1010 REM 128 X 192 TWO COLOUR MODE
1020 REM
1030 REM SELECT CORRECT VIDEO MEMORY MAPPING
1040 POKE 65473,1 : POKE 65474,1 : POKE 65477,1
1050 REM SELECT MODE
1060 REM COLOUR SET IS SELECTED BY MAKING C = 0 OR 8
1070 C=0
1080 REM CHANGE COLOUR SET
1090 IF C=0 THEN C=8 ELSE C=0
1100 POKE 65314,128+64+16+C
1110 FOR I=0 TO 3071
1120 POKE 1024+I,RND(255)
1130 NEXT I
1140 FOR T=1 TO 500 : NEXT T ' SMALL DELAY
1150 REM VERTICAL LINES
1160 FOR J=0 TO 15
1170 FOR I=0 TO 191
1180 REM RANDOM COLOURS
1190 X=RND(255)
1200 REM POKE THEM ONTO THE SCREEN
1210 POKE 1024+I*16+J,X
1220 NEXT I,J
1230 FOR T=1 TO 500 : NEXT T ' ANOTHER DELAY
1240 GOTO 1090
```

128 BY 192 FOUR COLOUR MODE

(Basic — PMODE 3)

Double the number of colours available over the previous mode makes a full screen use 6144 bytes.

```
1000 REM
1010 REM 128 X 192 FOUR COLOUR MODE
1020 REM
1030 REM SELECT CORRECT VIDEO MEMORY MAPPING
1040 POKE 65472,1 : POKE 65475,1 : POKE 65477,1
1050 REM SELECT MODE
1060 REM COLOUR SET IS SELECTED BY MAKING C = 0 OR 8
1070 C=0
1080 REM CHANGE COLOUR SET
1090 IF C=0 THEN C=8 ELSE C=0
1100 POKE 65314,128+64+32+C
1110 FOR I=0 TO 6143
1120 POKE 1024+I,RND(255)
1130 NEXT I
1140 FOR T=1 TO 500 : NEXT T ' SMALL DELAY
1150 REM VERTICAL LINES
1160 FOR J=0 TO 31
1170 FOR I=0 TO 191
1180 REM RANDOM COLOURS
1190 X=RND(255)
1200 REM POKE THEM ONTO THE SCREEN
1210 POKE 1024+I*32+J,X
1220 NEXT I,J
1230 FOR T=1 TO 500 : NEXT T ' ANOTHER DELAY
1240 GOTO 1090
```

256 BY 192 TWO COLOUR MODE

(Basic—PMODE 4)

This is the last mode available in the Dragon and requires 6144 bytes of video memory for a full screen.

```
1000 REM
1010 REM 256 X 192 TWO COLOUR MODE
1020 REM
1030 REM SELECT CORRECT VIDEO MEMORY MAPPING
1040 POKE 65472,1 : POKE 65475,1 : POKE 65477,1
1050 REM SELECT MODE
1060 REM COLOUR SET IS SELECTED BY MAKING C = 0 OR 8
1070 C=0
1080 REM CHANGE COLOUR SET
1090 IF C=0 THEN C=8 ELSE C=0
1100 POKE 65314,128+64+32+C
1110 FOR I=0 TO 6143
1120 POKE 1024+I,RND(255)
1130 NEXT I
1140 FOR T=1 TO 500 : NEXT T ' SMALL DELAY
1150 REM VERTICAL LINES
1160 FOR J=0 TO 31
1170 FOR I=0 TO 191
1180 REM RANDOM COLOURS
1190 X=RND(255)
1200 REM POKE THEM ONTO THE SCREEN
1210 POKE 1024+I*32+J,X
1220 NEXT I,J
1230 FOR T=1 TO 500 : NEXT T ' ANOTHER DELAY
1240 GOTO 1090
```

VDG MEMORY MAPPING AND MODE SELECTION

The following tables summarise the information contained in the example programs so far.

Decimal address	Hex address	Function
65472	FFCØ	CLEAR BIT Ø
65473	FFC1	SET BIT Ø
65474	FFC2	CLEAR BIT 1
65475	FFC3	SET BIT 1
65476	FFC4	CLEAR BIT 2
65477	FFC5	SET BIT 2

Table 1.—The six locations from 65472 to 65477 control the memory mapping mode of the VDG. Table 2 explains the memory maps selected by the various bit patterns.

VDG MEM. MODE	BIT PATTERN	BYTES PER
Ø	Ø Ø Ø	Ø512
1	Ø Ø 1	1Ø24
2	Ø 1 Ø	2Ø48
3	Ø 1 1	1536
4	1 Ø Ø	3Ø72
5	1 Ø 1	3Ø72
6	1 1 Ø	6144

Table 2.—The bit patterns above are set up in the SAM bits by pokes into locations 65472 to 65477 as shown in Table 1.

MODE	BIT PATTERN
TEXT / LOW RESOLUTION 4.	Ø Ø Ø Ø Ø Ø Ø Ø
LOW RESOLUTION 6	Ø Ø Ø 1 X Ø Ø Ø
LOW RESOLUTION 8	Ø Ø Ø Ø Ø Ø Ø Ø
LOW RESOLUTION 12.	Ø Ø Ø Ø Ø Ø Ø Ø
LOW RESOLUTION 24.	Ø Ø Ø Ø Ø Ø Ø Ø
64 BY 64 FOUR COL.	1 Ø Ø Ø X Ø Ø Ø
128 BY 64 TWO COL.	1 Ø Ø 1 X Ø Ø Ø
128 BY 64 FOUR COL	1 Ø 1 Ø X Ø Ø Ø
128 BY 96 TWO COL.	1 Ø 1 1 X Ø Ø Ø
128 BY 96 FOUR COL	1 1 Ø Ø X Ø Ø Ø
128 BY 192 TWO COL	1 1 Ø 1 X Ø Ø Ø
128 BY 192 FOUR COL.	1 1 1 Ø X Ø Ø Ø
256 BY 192 TWO COL	1 1 1 1 X Ø Ø Ø

Table 3.—This table shows the bit patterns to be poked into location 65314 for graphic mode selection. The X can be either 1 or Ø and indicates the colour set selected.

VIDEO MEMORY BASE-PAGE LOCATION

Now that we can select any of the fourteen modes provided by the VDG without using Basic graphic commands we are in a position to be able to write our own machine code programs which take advantage of them. There is, however, a very important feature of the Dragon's graphics still to be explained and that is the specification of the start of video ram. This is important because several pictures can be held in memory at the same time and a form of animation achieved by switching rapidly between them.

The Dragon uses locations 65478 to 65491 to control a seven bit value in the SAM bits which when multiplied by 512 gives the address of the start of the video ram. Table 4 explains the functions of the locations.

DECIMAL ADDRESS	HEX ADDRESS	FUNCTION
65478	FFC6.	CLEAR BIT 0
65479	FFC7.	SET BIT 0
65480	FFC8.	CLEAR BIT 1
65481	FFC9.	SET BIT 1
65482	FFCA.	CLEAR BIT 2
65483	FFCB.	SET BIT 2
65484	FFCC.	CLEAR BIT 3
65485	FFCD.	SET BIT 3
65486	FFCE.	CLEAR BIT 4
65487	FFCF.	SET BIT 4
65488	FFD0.	CLEAR BIT 5
65489	FFD1.	SET BIT 5
65490	FFD2.	CLEAR BIT 6
65491	FFD3.	SET BIT 6

Table 4.—The Dragon calculates the video base-page location by multiplying the seven bit number contained in the SAM bits by 512.

The two short example programs give good examples of the power of this feature.

```
1000 REM
1010 REM THIS ROUTINE SHOWS THE DRAGON'S ZERO PAGE WORK AREA
1020 REM
1030 REM PRESS 'BREAK' TO RETURN TO NORMAL
1040 POKE &HFFC8,0
1050 GOTO 1050
```

```
1000 REM
1010 REM VIDEO BASE PAGE SWITCHING USING 64 X 64 FOUR COLOUR MODE
1020 REM
1030 POKE 65473,1 : POKE 65474,1 : POKE 65476,1
1040 POKE 65314,128
1050 FOR I=0 TO 1023 ' SET UP TWO PAGES
1060 POKE 1024+I,&H9B : POKE 2048+I,RND(255)
1070 NEXT I
1080 POKE 65483,1 : POKE 65480,1 ' SWITCH TO SECOND PAGE
1090 FOR I=0 TO 100 : NEXT I
1100 POKE 65481,1 : POKE 65482,1 ' SWITCH BACK TO FIRST PAGE
1110 FOR I=0 TO 100 : NEXT I
1120 GOTO 1080
```

PROCESSOR SPEEDS

Another feature of the Dragon which is very interesting, and indeed can be extremely useful, is the fact that it has a variable processor clock rate. For those of you who do not understand about clock rates a simple explanation is that the central processing unit (in this case the 68Ø9E) receives a regular tick from a timer which tells it to move on to the next stage of obeying an instruction. These ticks are measured in megahertz (millions of ticks per second) and the faster the tick the faster the computer works. In the Dragon's case the clock rate is controlled by SAM bits in locations 65494 to 65497. These four locations control two SAM bits which should give us four clock rates. Table 5 sets out the functions of the locations.

DECIMAL ADDRESS	HEX ADDRESS	FUNCTION
65494	FFD6.	CLEAR BIT 0
65495	FFD7.	SET BIT 0
65496	FFD8.	CLEAR BIT 1
65497	FFD9.	SET BIT 1

Table 5.

Two bits should give four speeds but the Dragon appears only to respond to three. The default speed, set on switching on, is the slowest with both SAM bits cleared (poking any value to 65494 and 65496 achieves the same effect). The next faster speed is set by poking to 65495 and results in the execution of programs being 50% faster. The slowest two speeds are the only ones which can be used and still retain video synchronisation.

The next increase in speed is achieved by setting bit 1 and clearing bit Ø. Execution speeds are 100% faster than the default speed but video synchronisation is lost. This speed would be useful if a program involves large amounts of computation and where video is not important. Video synchronisation can always be regained by slowing the processor down again after the burst of computation.

The final speed should be achieved when both bits are set but as I have said, the Dragon does not appear to respond (see the example program).

One thing to remember about the faster speeds is that the cassette interface will only work at the default speed so make sure you save your programs at the correct speed.

The following routine gives examples of the various speeds. Notice that the SOUND command has constant parameters as does the delay subroutine in line 3ØØØ. Listen to the note change and watch the delay fall by 50% each time. The third and fourth speeds appear to be the same.

```
1000 REM
1010 REM PROCESSOR SPEED DEMO
1020 REM
1030 REM START WITH DEFAULT SPEED SET
1040 GOSUB 2000 : GOSUB 3000 : GOSUB 2000
1050 POKE 65495,1 : GOSUB 2000 : GOSUB 3000 : GOSUB 2000
1060 POKE 65494,1 : POKE 65497,1 : GOSUB 2000 : GOSUB 3000 : GOSUB 2000
1070 POKE 65495,1 : GOSUB 2000 : GOSUB 3000 : GOSUB 2000
1080 POKE 65494,1 : POKE 65496,1 : END
2000 SOUND 10,10 : RETURN
3000 FOR I=1 TO 10000 : NEXT I : RETURN
```

LOADING MORE THAN ONE PROGRAM FROM TAPE

Dragon's Basic has fairly sophisticated and reliable tape handling facilities but there is one limitation—only one Basic program can be loaded at a time. If a second program is loaded it overwrites the one already in memory. As you progress with your programming you will no doubt collect quite a number of routines which could be used in more than one program so to save constant retyping I have included a method of appending programs. The process can be broken down into a number of steps:

1.... Make sure that the line numbers of the program on tape start at a higher value than the highest line number of the program already in memory.

2.... PEEK at the contents of locations 25 and 26 (19 and 1A HEX) and note the values. These two locations are used as a pointer to the start of the program in memory.

3.... Execute the following—POKE 25, PEEK (27): POKE 26, PEEK (28)-2. This alters the pointer so that it now points to the end of the program.

4.... Use CLOAD to bring into memory the program from tape.

5.... POKE into locations 25 and 26 the values noted in step 2 above.

The second program is now appended to the first—use LIST to check.

The only thing to watch for in this process is that subtracting 2 from PEEK (28) in step 3 above does not result in a negative number. If this does happen replace step 3 by:—
POKE 25, PEEK (27)—1:POKE 26, 256-PEEK (28)

There is no reason why the above process cannot be repeated any number of times to take advantage of a whole library of subroutines provided there is enough memory available.

A 6809 DISASSEMBLER

This program was developed to aid me in writing this book. I have found it so useful that I have included it for your use.

Screen output is restricted to the address of the instruction being disassembled and any data in hex format followed by the mnemonic representation. I used a book by Lance Leventhal called 6809 ASSEMBLY LANGUAGE PROGRAMMING, published by Osbourne/McGraw-Hill, as a reference for this program.

Printer output includes all of the above plus a further field which unfortunately cannot fit onto the screen at the same time as the rest of the information. This field is the ASCII representation (where it exists) of all the bytes just read. This is useful for deciphering data areas of programs.

On running the program there will be a short delay while data structures are initialised followed by the prompt 'START?'.

Reply to this with the address at which you want disassembly to commence – try &H8000 to see the beginning of Basic. Next the prompt 'PRINTER (Y/N)?' will appear. Type 'Y' if you want printed output. Having replied to these questions the first fifteen lines of disassembled program will appear on the screen. The computer will then wait for a keypress before giving the next fifteen lines. Any portion of the area to be disassembled which is not understood by the program will result in a question mark appearing in the mnemonic field.

Should you wish to start disassembly again from another address type 'S' when at the end of a fifteen line screenful. This will give the 'START?' prompt again. Another command is 'C' which cancels the wait between screenfuls ie. you get a continuous disassembly.

```
10 REM 6809 DISASSEMBLER
20 CLEAR 2000
30 GOTO 1480
40 IF PEEK(S)<16 THEN M$=M$+"0"
50 M$=M$+HEX$(PEEK(S)):RETURN
60 Z=PEEK(S):IF Z<16 THEN P$=P$+"0"
70 P$=P$+HEX$(Z)
80 IF Z>&H7F THEN Z=Z-&H80
90 IF Z<&H20 THEN Z=&H20
100 T$=T$+CHR$(Z):RETURN
110 PRINT USING L$;A$,P$,C$,M$
120 IF PR=1 THEN PRINT #-2,USING J$;A$,P$,C$,M$,T$
130 A$="":P$="":C$="":M$="":T$="":S=S+1:GOTO200
140 S=S+1:M$=M$+"$":GOSUB40:GOSUB60:GOTO110
150 S=S+1:M$=M$+"#$":GOSUB40:GOSUB60:GOTO110
160 S=S+1:M$=M$+"#$":GOSUB40:GOSUB60:S=S+1:GOSUB40:GOSUB60:GOTO110
170 S=S+1:M$=M$+"$":GOSUB40:GOSUB60:S=S+1:GOSUB40:GOSUB60:GOTO110
180 CLS:L=0:N=0
190 INPUT"START ";S:INPUT"PRINTER (Y/N) ";I$:IF I$="Y" THEN PR=1 ELSE PR=0
200 N=N+1:IF L=1 THEN 220 ELSE IF N=15 THEN N=0 ELSE 220
210 I$=INKEY$:IF I$="C" THEN L=1 ELSE IF I$="" THEN 210 ELSE IF I$="S" THEN 180
220 X=PEEK(S):A$=HEX$(S):GOSUB60
230 IF MN$(X)="" THEN M$=M$+CHR$(X):C$="?":GOTO110
```

```
240 IF X=&H10 THEN GOTO650
250 IF X=&H11 THEN 920
260 C$=MN$(X)
270 IF X<&H10 THEN 140
280 IF X<&H15 OR X=&H19 OR X=&H1D OR (X>&H38 AND X<&H3C) OR (X>&H3C AND X<&H60)
THEN 110
290 IF X=&H16 OR X=&H17 THEN S=S+1:GOSUB60:S=S+1:GOSUB60:S=S-2:GOTO380
300 IF (X>&H1F AND X<&H30) OR X=&H8D THEN 420
310 IF X=&H1C OR X=&H1A OR X=&H3C OR (X>&H7F AND X<&H83) OR (X>&H83 AND X<&H8C)
OR (X>&HBF AND X<&HC3) OR (X>&HC3 AND X<&HCC) THEN 150
320 IF X=&H1E OR X=&H1F THEN 1030
330 IF X>&H33 AND X<&H38 THEN 530
340 IF (X>&H8F AND X<&HA0) OR (X>&HCF AND X<&HE0) THEN 140
350 IF X=&H83 OR X=&H8C OR X=&H8E OR X=&HC3 OR X=&HCC OR X=&HCE THEN 160
360 IF (X>&H6F AND X<&H80) OR (X>&HAF AND X<&HC0) OR X>&HEF THEN 170
370 IF (X>&H2F AND X<&H34) OR (X>&H5F AND X<&H70) OR (X>&H9F AND X<&HB0) OR (X>&
HDF AND X<&HF0) THEN 460
380 DISPLACEMENT=PEEK(S+1)*256+PEEK(S+2)
390 IF DI>=&H8000 THEN DI=DI-&HFFFF - 1
400 S=S+2:DI=DI+S+1:IF DI>&HFFFF THEN DI=DI-&HFFFF - 1
410 M$=M$+"$"+HEX$(DI):GOTO110
420 DI=PEEK(S+1)
430 IF DI>=&H80 THEN DI=DI-&HFF - 1
440 S=S+1:M$=M$+"$":GOSUB60
450 DI=DI+S+1:M$=M$+HEX$(DI):GOTO110
460 S=S+1:PO=PEEK(S):M$=M$+PO$(PO)
470 GOSUB60:NB=PB(PO)
480 IF NB>0 THEN S=S+1:GOSUB60:NB=NB-1
490 IF NB>0 THEN S=S+1:GOSUB60
500 U=INSTR(1,M$,"MM"):IF U<>0 THEN M$=LEFT$(M$,U-1)+RIGHT$(P$,4)+RIGHT$(M$,LEN(
M$)-U-3):GOTO110
510 U=INSTR(1,M$,"NN"):IF U<>0 THEN M$=LEFT$(M$,U-1)+RIGHT$(P$,2)+RIGHT$(M$,LEN(
M$)-U-1)
520 GOTO110
530 S=S+1:GOSUB60:PO=PEEK(S)
540 IF PO AND &H80 THEN REG$=REG$+"PC,"
550 IF PO AND &H40 THEN REG$=REG$+"U/S,"
560 IF PO AND &H20 THEN REG$=REG$+"Y,"
570 IF PO AND &H10 THEN REG$=REG$+"X,"
580 IF PO AND &H08 THEN REG$=REG$+"DP,"
590 IF PO AND &H04 THEN REG$=REG$+"B,"
600 IF PO AND &H02 THEN REG$=REG$+"A,"
610 IF PO AND &H01 THEN REG$=REG$+"CC"
620 M$=M$+REG$:REG$="":GOTO110
630 S=S+1:GOSUB60:PO=PEEK(S):M$=M$+PO$(PO)
640 NB=PB(PO):GOTO 480
650 S=S+1:X=PEEK(S):GOSUB60
660 IF X>&H20 AND X<&H30 THEN C$=LB$(X-&H20) ELSE 680
670 S=S+1:GOSUB60:S=S+1:GOSUB60:S=S-2:GOTO380
680 IF X=&H3F THEN C$="SWI2":S=S+1:GOTO110
690 IF X=&H83 THEN C$="CMPD":GOTO160
700 IF X=&H8C THEN C$="CMPY":GOTO160
710 IF X=&H8E THEN C$="LDY":GOTO160
720 IF X=&H93 THEN C$="CMPD":GOTO140
730 IF X=&H9C THEN C$="CMPY":GOTO140
740 IF X=&H9E THEN C$="LDY":GOTO140
750 IF X=&H9F THEN C$="STY":GOTO140
760 IF X=&HA3 THEN C$="CMPD":GOTO630
770 IF X=&HAC THEN C$="CMPY":GOTO630
780 IF X=&HAE THEN C$="LDY":GOTO630
790 IF X=&HAF THEN C$="STY":GOTO630
```

```
800 IF X=&HB3 THEN C$="CMPD":GOTO170
810 IF X=&HBC THEN C$="CMPY":GOTO170
820 IF X=&HBE THEN C$="LDY":GOTO170
830 IF X=&HBF THEN C$="STY":GOTO170
840 IF X=&HCE THEN C$="LDS":GOTO160
850 IF X=&HDE THEN C$="LDS":GOTO140
860 IF X=&HDF THEN C$="STS":GOTO140
870 IF X=&HEE THEN C$="LDS":GOTO460
880 IF X=&HEF THEN C$="STS":GOTO460
890 IF X=&HFE THEN C$="LDS":GOTO170
900 IF X=&HFF THEN C$="STS":GOTO170
910 P$="10":C$="?":S=S-1:GOTO110
920 S=S+1:X=PEEK(S):GOSUB60
930 IF X=&H3F THEN C$="SWI3":S=S+1:GOTO140
940 IF X=&H83 THEN C$="CMPU":GOTO160
950 IF X=&H8C THEN C$="CMPS":GOTO160
960 IF X=&H93 THEN C$="CMPU":GOTO140
970 IF X=&H9C THEN C$="CMPS":GOTO140
980 IF X=&HA3 THEN C$="CMPU":GOTO460
990 IF X=&HAC THEN C$="CMPS":GOTO460
1000 IF X=&HB3 THEN C$="CMPU":GOTO170
1010 IF X=&HBC THEN C$="CMPS":GOTO170
1020 P$="11":C$="?":S=S-1:GOTO110
1030 S=S+1:X=PEEK(S):GOSUB60
1040 R1=INT(X/16):R2=X-R1*16
1050 IF R1=0 THEN REG$=REG$+"D,":GOTO1150
1060 IF R1=1 THEN REG$=REG$+"X,":GOTO1150
1070 IF R1=2 THEN REG$=REG$+"Y,":GOTO1150
1080 IF R1=3 THEN REG$=REG$+"U,":GOTO1150
1090 IF R1=4 THEN REG$=REG$+"S,":GOTO1150
1100 IF R1=5 THEN REG$=REG$+"PC,":GOTO1150
1110 IF R1=8 THEN REG$=REG$+"A,":GOTO1150
1120 IF R1=9 THEN REG$=REG$+"B,":GOTO1150
1130 IF R1=10 THEN REG$=REG$+"CCR,":GOTO1150
1140 IF R1=11 THEN REG$=REG$+"DP,"
1150 IF R1=R2 THEN 1160 ELSE R1=R2:GOTO1050
1160 M$=M$+REG$:REG$="":GOTO110
1170 DATA NEG,,,COM,LSR,,ROR,ASR,ASL,ROL,DEC,,INC,TST,JMP,CLR,NUL,NUL,NOP,SYNC,,
,LBRA,LBSR,,DAA
1180 DATA ORC,,ANDCC,SEX,EXG,TFR,BRA,BRN,BHI,BLS,BCC,BCS,BNE,BEQ,BVC,BVS,BPL,BMI
,BGE,BLT,BGT,BLE
1190 DATA LEAX,LEAY,LEAS,LEAU,PSHS,PULS,PSHU,PULU,,RTS,ABX,RTI,CWAI,MUL,,SWI,NEG
A,,,COMA,LSRA,,RORA
1200 DATA ASRA,ASLA,ROLA,DECA,,INCA,TSTA,,CLRA,NEGB,,,COMB,LSRB,,RORB,ASRB,ASLB,
ROLB,DECB,,INCB
1210 DATA TSTB,,CLRB,NEG,,,COM,LSR,,ROR,ASR,ASL,ROL,DEC,,INC,TST,JMP,CLR,NEG,,,C
OM,LSR,,ROR,ASR
1220 DATA ASL,ROL,DEC,,INC,TST,JMP,CLR,SUBA,CMPA,SBCA,SUBD,ANDA,BITA,LDA,,EORA,A
DCA,ORA,ADDA,CMPX
1230 DATA BSR,LDX,,SUBA,CMPA,SBCA,SUBD,ANDA,BITA,LDA,STA,EORA,ADCA,ORA,ADDA,CMPX
,JSR,LDX,STX,SUBA
1240 DATA CMPA,SBCA,SUBD,ANDA,BITA,LDA,STA,EORA,ADCA,ORA,ADDA,CMPX,JSR,LDX,STX,S
UBA,CMPA,SBCA,SUBD
1250 DATA ANDA,BITA,LDA,STA,EORA,ADCA,ORA,ADDA,CMPX,JSR,LDX,STX,SUBB,CMPB,SBCB,A
DDD,ANDB,BITB
1260 DATA LDB,,EORB,ADCB,ORB,ADDB,LDD,,LDU,,SUBB,CMPB,SBCB,ADDD,ANDB,BITB,LDB,ST
B,EORB,ADCB,ORB
1270 DATA ADDB,LDD,STD,LDU,STU,SUBB,CMPB,SBCB,ADDD,ANDB,BITB,LDB,STB,EORB,ADCB,O
RB,ADDB,LDD,STD
1280 DATA LDU,STU,SUBB,CMPB,SBCB,ADDD,ANDB,BITB,LDB,STB,EORB,ADCB,ORB,ADDB,LDD,S
TD,LDU,STU
```

```
1290 DATA 0.X,1.X,2.X,3.X,4.X,5.X,6.X,7.X,8.X,9.X,10.X,11.X,12.X,13.X,14.X,15.X,
-16.X,-15.X,-14.X
1300 DATA -13.X,-12.X,-11.X,-10.X,-9.X,-8.X,-7.X,-6.X,-5.X,-4.X,-3.X,-2.X,-1.X,0
.Y,1.Y,2.Y
1310 DATA 3.Y,4.Y,5.Y,6.Y,7.Y,8.Y,9.Y,10.Y,11.Y,12.Y,13.Y,14.Y,15.Y,-16.Y,-15.Y,
-14.Y,-13.Y,-12.Y
1320 DATA -11.Y,-10.Y,-9.Y,-8.Y,-7.Y,-6.Y,-5.Y,-4.Y,-3.Y,-2.Y,-1.Y,0.U,1.U,2.U,3
.U,4.U,5.U,6.U
1330 DATA 7.U,8.U,9.U,10.U,11.U,12.U,13.U,14.U,15.U,-16.U,-15.U,-14,U,-13.U,-12.
U,-11.U,-10.U
1340 DATA -9.U,-8.U,-7.U,-6.U,-5.U,-4.U,-3.U,-2.U,-1.U,0.S,1.S,2.S,3.S,4.S,5.S,6
.S,7.S,8.S,9.S
1350 DATA 10.S,11.S,12.S,13.S,14.S,15.S,-16.S,-15.S,-14.S,-13.S,-12.S,-11.S,-10.
S,-9.S,-8.S,-7.S
1360 DATA -6.S,-5.S,-4.S,-3.S,-2.S,-1.S,.X+,.X++,.-X,.--X,.X,B.X,A.X,,NN.X,MMNN.
X,,D.X,NN.PC
1370 DATA MMNN.PC,,,,[.X++],,[.--X],[.X],[B.X],[A.X],,[NN.X],[MMNN.X],,[D.X],[NN
.PC],[MMNN.PC]
1380 DATA ,[MMNN],.Y+,.Y++,.-Y,.--Y,.Y,B.Y,A.Y,,NN.Y,MMNN.Y,,D.Y,NN.PC,MMNN.PC,,
,,[.Y++],,[.--Y]
1390 DATA [.Y],[B.Y],[A.Y],,[NN.Y],[MMNN.Y],,[D.Y],[NN.PC],[MMNN.PC],,[MMNN],.U+
,.U++,.-U,.--U
1400 DATA .U,B.U,A.U,,NN.U,MMNN.U,,D.U,NN.PC,MMNN.PC,,,,[.U++],,[.--U],[.U],[B.U
],[A.U],,[NN.U]
1410 DATA [MMNN.U],,[D.U],[NN.PC],[MMNN.PC],,[MMNN],.S+,.S++,.-S,.--S,.S,B.S,A.S
,,NN.S,MMNN.S,,D.S
1420 DATA NN.PC,MMNN.PC,,,,[.S++],,[.--S],[.S],[B.S],[A.S],,[NN.S],[MMNN.S],,[D.
S],[NN.PC],[MMNN.PC],,[MMNN]
1430 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0
1440 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1450 DATA 0,0,0,0,0,0,0,0,1,2,0,0,1,2,0,0,0,0,0,0,0,0,0,1,2,0,0,1,2,0,2,0,0,0,
0,0,0,0,0,1,2,0,0,1,2,0,0,0,0,0,0,0,0,0,1,2,0,0,1,2,0,2,0,0,0,0,0,0,0,0,1,2,0,
0
1460 DATA 1,2,0,0,0,0,0,0,0,0,0,0,1,2,0,0,1,2,0,2,0,0,0,0,0,0,0,0,1,2,0,0,1,2,0,
0,0,0,0,0,0,0,0,0,1,2,0,0,1,2,0,2
1470 DATA LBRN,LBHI,LBLS,LBHS,LBLO,LBNE,LBEQ,LBVC,LBVS,LBPL,LBMI,LBGE,LBLT,LBGT,
LBLE
1480 DIM MN$(&HFF),PO$(&HFF),LB$(&HF),PB(&HFF)
1490 FOR I=0 TO &HFF:READ MN$(I):NEXT I
1500 FOR I=0 TO &HFF:READ PO$(I):NEXT I
1510 FOR I=0 TO &HFF:READ PB(I):NEXT I
1520 FOR I=1 TO &HF:READ LB$(I):NEXT I
1530 L$="%  % %          % %    % %          %"
1540 J$="%  % %          % %    % %                         % %    %"
1550 GOTO180
```

USEFUL ROUTINES IN THE BASIC ROM

When you start writing your own machine code programs you will want at some stage to use routines which can be found within the Basic rom. For instance, it would be a waste of time to write a routine for reading the keyboard when such a routine already exists. I have included specifications of the ones I have managed to find—there are probably many more.

All numbers in this section are given in hexadecimal form.

The names used for all registers and condition code flags are the standard 68Ø9 ones a full explanation of which can be found in any reference book.

Some zero-page locations are used by these routines the more important of which are:—

6F—Device specification for character output (Ø to the screen and -2 to the printer).

7C—Block type for tape routines (Ø = file header, 1 = data, FF = end of file).

7D—Block length for tape routines (less than or equal to FF).

7E/7F—Buffer address for tape routines.

Locations AØØØ to AØØD contain the addresses of the following routines. They can be called from machine code programs by an indirect jump instruction.

eg. JSR [AØØØ]

1. . . . [AØØØ] = POLKBD

This routine looks at the keyboard for a keypress and if there is one its value is returned in the accumulator. It can be thought of as the same as INKEYS$ in Basic.

Entry conditions—None.

Exit conditions. —Z = 1, A = Ø if no key pressed.
........................Z = Ø, A = key code if key pressed.
........................All registers except B and X are modified.

2. . . . [AØØ2] = OPCHR

This routine outputs the character whose code is in the accumulator to the device specified by location 6F (Ø to the screen, —2 to the printer).

Entry conditions—Character to be output is in the accumulator.

Exit conditions.—Only the condition code register is modified.

3. . . . [AØØ4] = CASSON

A call to CASSON starts the cassette motor and gets the machine into synchronisation ready for data transfer.

Entry conditions—None.

Exit conditions.—All registers except U and Y are changed.

........................FIRQ and IRQ are masked.

4.... [AØØ6] = RDBLK
RDBLK reads a block of data from the tape.

Entry conditions—CASSON must have been called and the buffer address (7E) must have been initialised.

Exit conditions.—Location 7C (block type) will contain the code for the data just read.
........................Location 7D (block length) will contain a count of the number of bytes in this block.
........................Z = 1, A = Ø—no errors.
........................Z = Ø, a = 1—checksum error.
........................Z = Ø, A = 2—memory error.
........................If a memory error occurs then the X register points to just after the bad address, otherwise the X register contains the buffer address plus the block length.
........................All registers except U and Y are modified.
........................Interupts are disabled.

5.... [AØØ8] = WRBLK
WRBLK writes a block of data to tape.

Entry conditions—Block type, block length and buffer address should have been initialised.

........................If this is the first block to be written then it should have been preceded by WRLEAD (7).

Exit conditions. —All registers are changed.
........................Interupts are disabled.
........................X register = buffer address + block length.

6.... [AØØA] = JOYSTK
JOYSTK examines the joystick ports and stores the values read in locations 15A to 15D.
Right joystick 15A—right/left
Right joystick 15B—up/down

Left joystick 15C—right/left
Left joystick 15D—up/down

Entry conditions—None

Exit conditions.—All registers except Y are modified.

7.... [AØØC] = WRLEAD
WRLEAD turns the cassette motor on and writes a leader of 55's to the tape.

Entry conditions—None.

Exit conditions—None.

The first few bytes of the basic rom contain a number of jump instructions. They point to entry points for various subroutines the more useful of which can be incorporated in machine code programs using a subroutine call:

JSR $8006	POLLS THE KEYBOARD (SEE PREVIOUS EXPLANATION OF POLKBD)
JSR $8009	BLINK THE CURSOR
JSR $800C	OUTPUT A CHARACTER (SEE PREVIOUS EXPLANATION OF OPCHR)
JSR $8015	TURN ON CASSETTE MOTOR
JSR $8018	TURN OFF CASSETTE MOTOR
JSR $801B	PREPARE CASSETTE FOR WRITING
JSR $801E	OUTPUT CHARACTER IN ACCUMULATOR 'A' TO TAPE
JSR $8021	PREPARE CASSETTE FOR INPUT
JSR $8024	INPUT NEXT 8 BITS FROM TAPE TO ACCUMULATOR 'A'
JSR $8027	GET NEXT BIT FROM TAPE TO CARRY FLAG

BASIC STORAGE

When you write a program in Basic on the Dragon you are in fact using an editor which takes each line as you type it and converts it to a form which can be understood by another program called the interpreter. All the reserved words in Basic are converted into tokens which take up either one or two bytes of memory instead of a byte for each letter of the word. This obviously saves space but much more importantly it greatly increases the speed at which the interpreter can execute programs. A list of the tokens used by the Dragon follows.

To get an idea of what a program looks like in memory type in and run the following short routine.

```
10 X=PEEK(25)*256+PEEK(26)
20 FOR I=0 TO 79
30 Z=PEEK(X+I)
50 PRINT HEX$(Z);" ";
60 NEXT I
```

When you run the program the screen of the Dragon will fill the numbers:—

```
1E 19 0 A 58 CB FF 8C 28 32 35 29 C5 32 35 36 C3 FF 8C 28 32 36 29 0 1E 28 0 14
80 20 49 CB 30 20 BC 20 37 39 0 1E 36 0 1E 5A CB FF 8C 28 58 C3 49 29 0 1E 47 0
32 87 20 FF 95 28 5A 29 3B 22 20 22 3B 0 1E 4F 0 3C 8B 20 49 0 0 0
```

The first number is a zero—this is a line delimiter in the Basic program. The next two numbers together are used to point to the start of the next line in memory. Two more bytes follow which taken together give the number of that line. It is not until after this that the program proper starts.

The first line of our program starts with 'X' which is stored as an ASCII character—in this case 58 hex. Next comes '=' which happens to be a Basic reserved word and so is converted into a token—CB hex. 'PEEK' is the next word and it too is a reserved word so it is converted into a token—this time FF 8C hex. The next four characters in the program are '(25)' and as these are not reserved words they are stored in their ASCII formats.

We can continue this process until we reach the end of the line—signified by a zero—at which point the whole thing repeats for the second line. The program end is signified by three consecutive zeros.

Notice that the tokens can be split into two groups—single byte and double byte. The reason for this is that the tables of tokens are stored in different parts of the Basic rom and the interpreter needs to know which table to use. The FF hex prefix to the token proper (which is always greater than 7F hex) indicates the second table. The first table begins at address 8Ø33 hex and extends to 8153 hex. The second begins at 81CA hex and ends at 824F hex.

The interpreter uses the token's position in the table as an index into a further table—this time of addresses of routines to carry out the command indicated by the reserved word. These tables of addresses follow the token tables.

EASIER INPUT OF MACHINE CODE

The more you program the Dragon the more you will want to write your own machine code programs. Without the facilities of a good assembler this can be a tedious if not impossible task. The following program helps to make the entering of short routines a little easier but it is obviously no substitute for the real thing.

The program starts by asking for the address at which you want to begin entering your code. It then outputs the address on a new line together with the present contents of that location. It then waits for your input.

You can enter one or two bytes of data at a time. When you have done this and pressed return the address is updated to the next free location and the process repeated.

If you do not wish to alter a particular location enter a negative number in response to the prompt.

```
1000 REM
1010 REM EASY MACHINE CODE INPUT
1020 REM
1030 INPUT "START ";S
1040 PRINTHEX$(S);" ";HEX$(PEEK(S));
1050 INPUT N
1060 IF N<0THENS=S+1:GOTO1040
1070 IF N>255 THEN POKE S,INT(N/256) : N=N-INT(N/256)*256 : S=S+1
1080 POKE S,N : S=S+1
1090 GOTO1040
```

ADDING COMMANDS TO BASIC

The version of Basic supplied with the Dragon is a very comprehensive package but no language can be all things to all men. It is often desirable to extend the language if possible with customised commands which suit particular applications. I will describe one way of doing this and give an example of a command which I personally feel is missing from the Dragon.

Basic uses a self-modifying routine starting at address 9F hex which is used to load the accumulator of the 6809 with the next byte to be processed. This byte usually comes from one of two areas—the program storage space or the input buffer area.

The routine has the form: —

```
0002 009F 0CA7                 INC $00A7
0003 00A1 2602                 BNE $00A5
0004 00A3 0CA6                 INC $00A6
0005 00A5 B61E5D               LDA $1E5D
0006 00A8 7EBB26               JMP $BB26
```

The instruction at address A5 hex does the loading of the accumulator while the code immediately before it alters the address from where the loading takes place. Once loaded, control is returned to Basic by a jump instruction at A8 hex. As this whole routine is in ram we can alter it to suit ourselves and if instead of returning control directly to Basic we make a slight detour to a routine of our own then we have effectively extended the language.

The steps in the process are: —

1. . . . Develop a routine which you would like to see added to Basic and enter it into a safe area of memory (use the CLEAR command if necessary).

ii. . . . Alter the JMP instruction at A8 hex to point to the new routine which should eventually return control to Basic with a JMP to the original location pointed to by addresses A9 and AA hex.

The following example can be entered using the EASY ENTRY program described earlier.

The effect of the program is to reverse the actions of the NEW command—that is if you type NEW and then decide that you want to regain the old program just type in '!' (an exclamation mark). Provided that the old program has not been corrupted for any reason then it will be restored.

I have chosen to have this command operate only in the direct mode, ie, an exclamation mark in a program is treated as just another character. I have achieved this by looking at location A6 hex. If this contains a 2 then the character just read came from the keyboard buffer (and so must have come from the keyboard). This shows that a program is not running and control is then passed to the routine to restore the program.

If you study the program in conjunction with the section on how Basic is stored you will see what is happening. A full explanation of the zero-page locations used can be found in the memory map at the end of this book.

The routine first of all sets up a pointer in the X-register which indicates the start of Basic storage. This value is always contained in addresses 19 and 1A hex. It then skips through memory until it finds the first zero after ensuring that the X-register points to the location following the initial zeros of Basic's area. When found, the zero indicates the end of the first line and the address now in the X-register is the start of the second line. This value is stored in the first pointer position of the Basic program. The final thing to do is to update locations 1B and 1C hex which indicate the end of the Basic program. Three consecutive zeros mark the end of the program so the routine skips through memory again until it finds them.

Once the routine is finished, control is restored to Basic by a jump to the start of the self-modifying routine so that the next character can be read.

The routine becomes operable as soon as EXEC &H7000 is typed in.

```
0002 7000 CC7006            LDD #$7006
0003 7003 DDA9              STD $00A9
0004 7005 39                RTS
0005 7006 8121              CMPA #$21
0006 7008 2608              BNE $7012
0007 700A 96A6              LDA $00A6
0008 700C 8102              CMPA #$02
0009 700E 2705              BEQ $7015
0010 7010 8621              LDA #$21
0011 7012 7EBB26            JMP $BB26
0012 7015 3410              PSHS X
0013 7017 9E19              LDX $0019
0014 7019 3003              LEAX 3,X
0015 701B A680              LDA ,X+
0016 701D 26FC              BNE $701B
0017 701F AF9F0019          STX [$0019]
0018 7023 A680              LDA ,X+
0019 7025 26FC              BNE $7023
0020 7027 A680              LDA ,X+
0021 7029 26F8              BNE $7023
0022 702B A680              LDA ,X+
0023 702D 26F4              BNE $7023
0024 702F 9F1B              STX $001B
0025 7031 3510              PULS X
0026 7033 0E9F              JMP $009F
```

TOKEN TABLE 1

Res. Word	Token
FOR	80
GO	81
REM	82
'	83
ELSE	84
IF	85
DATA	86
PRINT	87
ON	88
INPUT	89
END	8A
NEXT	8B
DIM	8C
READ	8D
LET	8E
RUN	8F
RESTORE	9Ø
RETURN	91
STOP	92
POKE	93
CONT	94
LIST	95
CLEAR	96
NEW	97
DEF	98
CLOAD	99
CSAVE	9A
OPEN	9B
CLOSE	9C
LLIST	9D
SET	9E
RESET	9F
CLS	AØ
MOTOR	A1
SOUND	A2
AUDIO	A3
EXEC	A4
SKIPF	A5
DEL	A6

Res. Word	Token
EDIT	A7
TRON	A8
TROFF	A9
LINE	AA
PCLS	AB
PSET	AC
PRESET	AD
SCREEN	AE
PCLEAR	AF
COLOR	BØ
CIRCLE	B1
PAINT	B2
GET	B3
PUT	B4
DRAW	B5
PCOPY	B6
PMODE	B7
PLAY	B8
DLOAD	B9
RENUM	BA
TAB(	BB
TO	BC
SUB	BD
FN	BE
THEN	BF
NOT	CØ
STEP	C1
OFF	C2
+	C3
—	C4
*	C5
/	C6
^	C7
AND	C8
OR	C9
>	CA
=	CB
<	CC
USING	CD

TOKEN TABLE 2

Res. Word	Token
SGN	FF 8Ø
INT	FF 81
ABS	FF 82
POS	FF 83
RND	FF 84
SQR	FF 85
LOG	FF 86
EXP	FF 87
SIN	FF 88
COS	FF 89
TAN	FF 8A
ATN	FF 8B
PEEK	FF 8C
LEN	FF 8D
STR$	FF 8E
VAL	FF 8F
ASC	FF 9Ø
CHR$	FF 91
EOF	FF 92
JOYSTK	FF 93
FIX	FF 94
HEX$	FF 95
LEFT$	FF 96
RIGHT$	FF 97
MID$	FF 98
POINT	FF 99
INKEY$	FF 9A
MEM	FF 9B
VARPTR	FF 9C
INSTR	FF 9D
TIMER	FF 9E
PPOINT	FF 9F
STRING$	FFAØ
USR	FFA1

CHARACTER TABLE ONE
LOWER CASE

CHARACTER	POKE DEC	POKE HEX	CHR$ DEC	CHR$ HEX	CHARACTER	POKE DEC	POKE HEX	CHR$ DEC	CHR$ HEX
@	0	00			SPACE	32	20		
A	1	01	97	61	!	33	21	N	
B	2	02	98	62	"	34	22	O	
C	3	03	99	63	#	35	23		
D	4	04	100	64	$	36	24	L	
E	5	05	101	65	%	37	25	O	
F	6	06	102	66	&	38	26	W	
G	7	07	103	67	'	39	27	E	
H	8	08	104	68	(	40	28	R	
I	9	09	105	69	)	41	29		
J	10	0A	106	6A	*	42	2A	C	
K	11	0B	107	6B	+	43	2B	A	
L	12	0C	108	6C	,	44	2C	S	
M	13	0D	109	6D	-	45	2D	E	
N	14	0E	110	6E	.	46	2E		
O	15	0F	111	6F	/	47	2F	A	
P	16	10	112	70	0	48	30	S	
Q	17	11	113	71	1	49	31	C	
R	18	12	114	72	2	50	32	I	
S	19	13	115	73	3	51	33	I	
T	20	14	116	74	4	52	34		
U	21	15	117	75	5	53	35	E	
V	22	16	118	76	6	54	36	Q	
W	23	17	119	77	7	55	37	U	
X	24	18	120	78	8	56	38	I	
Y	25	19	121	79	9	57	39	V	
Z	26	1A	122	7A	:	58	3A	A	
[	27	1B	123	7B	;	59	3B	L	
\	28	1C	124	7C	<	60	3C	E	
]	29	1D	125	7D	=	61	3D	N	
↑	30	1E	126	7E	>	62	3E	T	
←	31	1F	127	7F	?	63	3F		

CHARACTER TABLE ONE
UPPER CASE

	POKE		CHR$			POKE		CHR$	
CHARACTER	DEC	HEX	DEC	HEX	CHARACTER	DEC	HEX	DEC	HEX
@	64	40	64	40	SPACE	96	60	32	20
A	65	41	65	41	!	97	61	33	21
B	66	42	66	42	"	98	62	34	22
C	67	43	67	43	#	99	63	35	23
D	68	44	68	44	$	100	64	36	24
E	69	45	69	45	%	101	65	37	25
F	70	46	70	46	&	102	66	38	26
G	71	47	71	47	'	103	67	39	27
H	72	48	72	48	(	104	68	40	28
I	73	49	73	49	)	105	69	41	29
J	74	4A	74	4A	*	106	6A	42	2A
K	75	4B	75	4B	+	107	6B	43	2B
L	76	4C	76	4C	,	108	6C	44	2C
M	77	4D	77	4D	-	109	6D	45	2D
N	78	4E	78	4E	.	110	6E	46	2E
O	79	4F	79	4F	/	111	6F	47	2F
P	80	50	80	50	0	112	70	48	30
Q	81	51	81	51	1	113	71	49	31
R	82	52	82	52	2	114	72	50	32
S	83	53	83	53	3	115	73	51	33
T	84	54	84	54	4	116	74	52	34
U	85	55	85	55	5	117	75	53	35
V	86	56	86	56	6	118	76	54	36
W	87	57	87	57	7	119	77	55	37
X	88	58	88	58	8	120	78	56	38
Y	89	59	89	59	9	121	79	57	39
Z	90	5A	90	5A	:	122	7A	58	3A
[	91	5B	91	5B	;	123	7B	59	3B
\	92	5C	92	5C	<	124	7C	60	3C
]	93	5D	93	5D	=	125	7D	61	3D
↑	94	5E	94	5E	>	126	7E	62	3D
←	95	5F	95	5F	?	127	7F	63	3F

NOTES ON CHARACTER TABLE ONE

As mentioned earlier in the book the video display generator uses a modified ASCII character set. This set is shown in the columns headed 'POKE'—a character can be placed on the Dragon screen by poking the appropriate value to screen ram.

Basic strings are stored as true ASCII values and the CHR$ function demands true ASCII. What this means is that the true ASCII value is converted by the Basic interpreter before a character is displayed on the screen. The true ASCII values handled by Basic are shown in the columns headed 'CHR$'.

The two simple routines below will show the characters produced by various values.

```
10 REM CHR$ VALUES
20 CLS
30 PRINT@0, "VALUE";:INPUT V
40 PRINT@016, CHR$(V);:PRINT@8, "      ";
50 GOTO30
```

```
10 REM VDG CHARACTER SET
20 CLS
30 PRINT@0. "VALUE";:INPUT V
40 POKE1038, V:PRINT@8, "      ";
50 GOTO30
```

EXTRA INFORMATION ON SAM CONTROL BIT AREA

PAGE SELECT (paged memory when implemented)

FFD4 (clear) FFD5 (set)	normally 0 for page 1

MEMORY SIZE

	4k	16k	32/64k	?
FFDA (clear bit zero) FFDB (set bit zero)	0	1	0	1
FFDC (clear bit one) FFDD (set bit one)	0	0	1	1

MAP TYPE (when implemented—0 = ram + rom, 1 = all ram)

FFDE (clear) FFDF (set)	normally 0

PIAs

The Dragon uses 2 6821 Parallel Interface Adaptors for the control of its I/0 functions. Each PIA occupies 4 memory locations in the section of the Dragon memory map reserved as the I/O area.

The first uses locations FF00 - FF03 hex.

Details of the individual bits within these locations follows:

Locations	bit	functions
FF00	0	keyboard row 1 and joystick switch
	1	keyboard row 2 and joystick switch
	2	keyboard row 3
	3	keyboard row 4
	4	keyboard row 5
	5	keyboard row 6
	6	keyboard row 7
	7	input for joystick comparison
FF01	0	)
	1	) Rapid IRQ
	2	usually set to 1 (governs function of FF00)
	3	least significant bit of multiplexer select lines
	4	set to 1 always
	5	set to 1 always
	6	
	7	
FF02	0	keyboard column 1
	7	keyboard column 8
FF03	0	)
	1	) Slow IRQ, Timer, Play, etc.
	2	usually set to 1 (governs functions of FF02)
	3	most significant bit of multiplexer select lines
	4	)
	5	) always set to 1
	6	
	7	

The second PIA occupies FF20 - FF23 hex.

Details of the individual bits:

Location	bit	function
FF20	0	cassette input
	1	printer strobe
	2	D/A least significant bit
	3	
	7	D/A most significant bit
FF21	0	
	1	
	2	usually set to 1 (governs function of FF20)
	3	cassetts motor control
	4	)
	5	) always set to 1
	6	
	7	
FF22	0	
	1	
	2	ram size
	3	VDG control
	4	VDG control
	5	VDG control
	6	VDG control
	7	VDG control
FF23	0	)
	1	) cartridge FIRQ
	2	usually set to 1 (governs function of FF22)
	3	six bit sound enable (TV)
	4	)
	5	) always set to 1
	6	
	7	flag—cartridge interrupt

CARTRIDGE EDGE CONNECTOR PIN DESCRIPTION

1 (TOP RIGHT)	− 12v
2 (BOTTOM RIGHT)	+ 12v
3	Halt input to CPU
4	Non maskable interrupt to CPU
5	Reset
6	E (Main CPU Clock (0.89 MHZ))
7	Q (Quadrative Clock Signal) (leads 6))
8	Cartridge interrupt input
9	+ 5v
10	Data bit 0
11	Data bit 1
12	Data bit 2
17	Data bit 7
18	R/W - CPU read/write signal
19	Address bit 0
20	Address bit 1
31	Address bit 12
32	Cartridge select
33	ground
34	ground
35	Sound input
36	Spare select signal
37	Address bit 13
38	Address bit 14
39 (TOP LEFT)	Address bit 15
40 (BOTTOM LEFT)	Devise selection disable input

SOUND OUTPUT SELECTION

Sound enable (FF23 bit 3)	multiplexer select lines (FF01 bit 3)	(FF03 bit 3)	Sound source
1	0	0	6 bit D/A
1	1	0	cassette
1	0	1	cartridge connector
1	1	1	unused

The 6821 PIA is a versatile chip which is able to perform many I/O functions. In order to understand it more fully I would recommend LEVENTHAL'S "6809 ASSEMBLY LANGUAGE PROGRAMMING".

NOTES

1. GRAPHICS MODES

Programs using the method of selecting graphics modes by poking to location 65314 may have problems if the bit patterns of table 3 are used. The reason is that the least significant 3 bits in this location control other functions and should not be changed. If you encounter difficulties then use something like:

```
POKE 65314, (PEEK(65314) AND 7)+128+C
```

This particular example is for the 64 by 64 four colour mode but is easily modified to cover all other modes. Having said all this I must say that I have not yet had any problems using straightforward pokes.

2. PROCESSOR SPEEDS

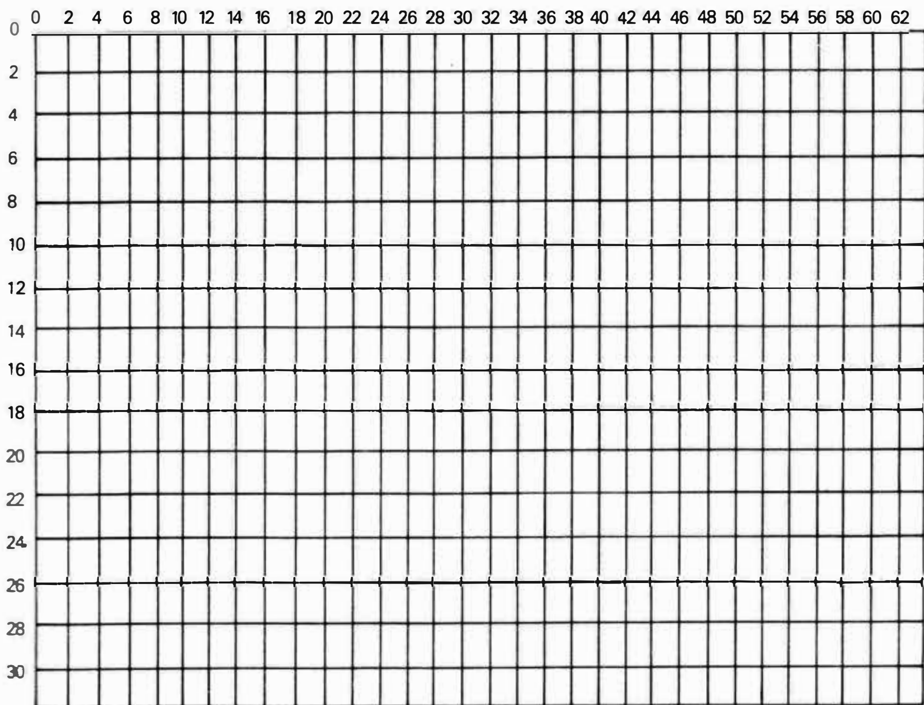
It has come to my attention that not all Dragons will respond to increasing the clock rate. If this is the case with your machine than I am afraid I know of no way of rectifying the situation.

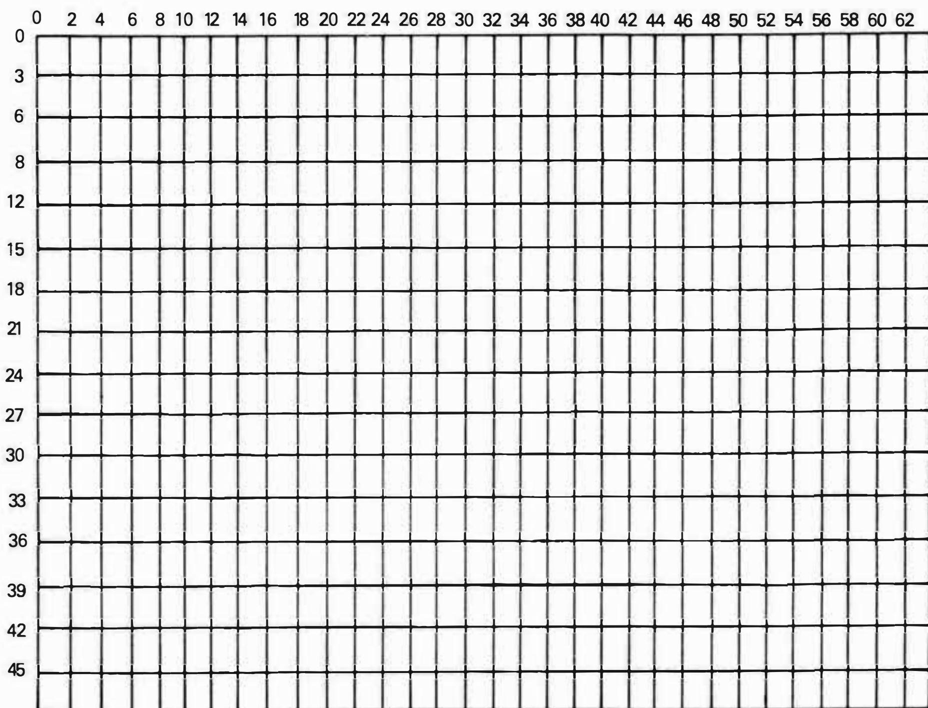
MEMORY MAP

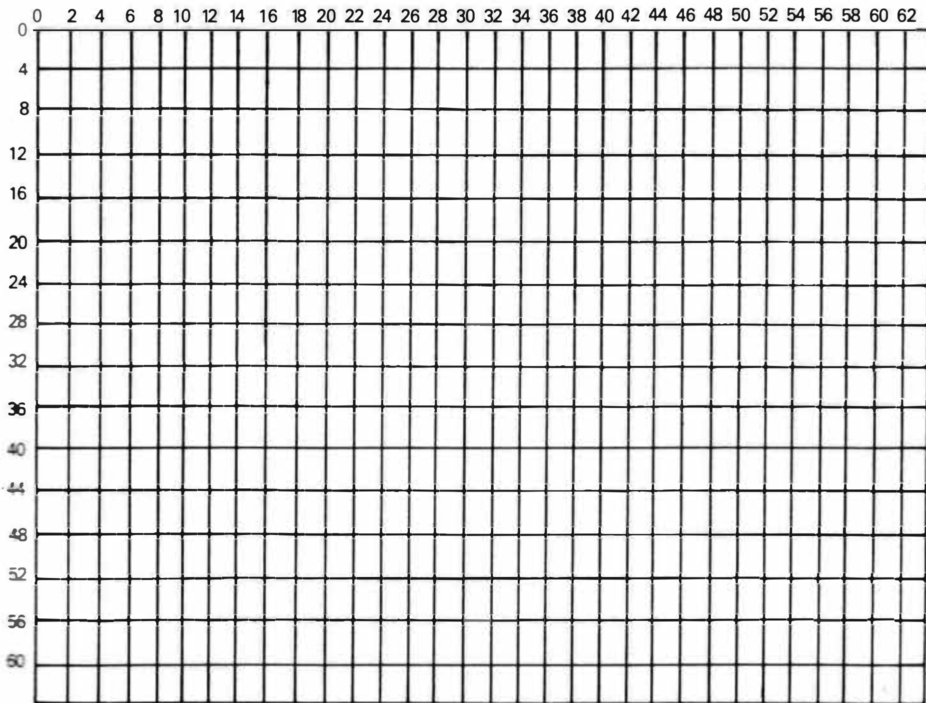
HEX	
19/1A	POINTER TO START OF BASIC
1B/1C	POINTER TO END OF BASIC/START OF VARIABLES
1D/1E	POINTER TO START OF ARRAY POINTER TABLE
1F/20	POINTER TO END OF RAM IN USE
21/22	POINTER TO TOP OF STACK
23/24	POINTER TO TOP OF STRING SPACE
27/28	POINTER TO TOP OF RAM
2D/2E	POINTER TO NEXT STATEMENT TO BE EXECUTED
2F/30	WARM START POINTER FOR RESET
31/32	LINE NUMBER OF CURRENT DATA
33/34	POINTER TO DATA
35/36	INPUT POINTER
68/69	CURRENT LINE NUMBER BEING EXECUTED
6F	OUTPUT DEVISE NUMBER (Ø=SCREEN, −1=CASSETTE, −2=PRINTER)
70	EOF FLAG
71	RESTART FLAG
72/73	RESTART POINTER
78	CASSETTE STATUS (Ø=CLOSED, 1=INPUT, 2=OUTPUT)
7C	BLOCK TYPE (CASSETTE)
7D	NUMBER OF DATA BYTES (CASSETTE)
7E/7F	BUFFER ADDRESS (CASSETTE)
80	CHECKSUM (CASSETTE)
81	CASSETTE ERROR CODE—IF Ø THEN NO ERRORS
82	FREQ COUNT FOR INPUT BIT (CASSETTE)
87	LAST KEY PRESSED
88/89	POINTER TO CURRENT CURSOR POSITION
90/91	LEADER BYTE COUNT (CASSETTE)
92	USUALLY 12 HEX—ON INPUT OF A SINGLE BIT FROM TAPE, LOCATION $82 CONTAINS A ROUGH MEASURE OF THE BIT'S FREQUENCY AS THE NUMBER OF TIMES THE INPUT PORT HAD TO BE POLLED BEFORE DETECTION. IF THIS VALUE IS GREATER THAN OR EQUAL TO THE VALUE IN $92 THEN THE BIT IS A ZERO OTHERWISE IT IS A ONE.
95/96	CASSETTE MOTOR DELAY
9D/9E	POINTER TO EXEC ADDRESS
9F-AA	SELF-MODIFYING ROUTINE
B0/B1	POINTER TO USR ADDRESS TABLE
B2	FOREGROUND COLOUR
B3	BACKGROUND COLOUR
B4	ACTIVE COLOUR

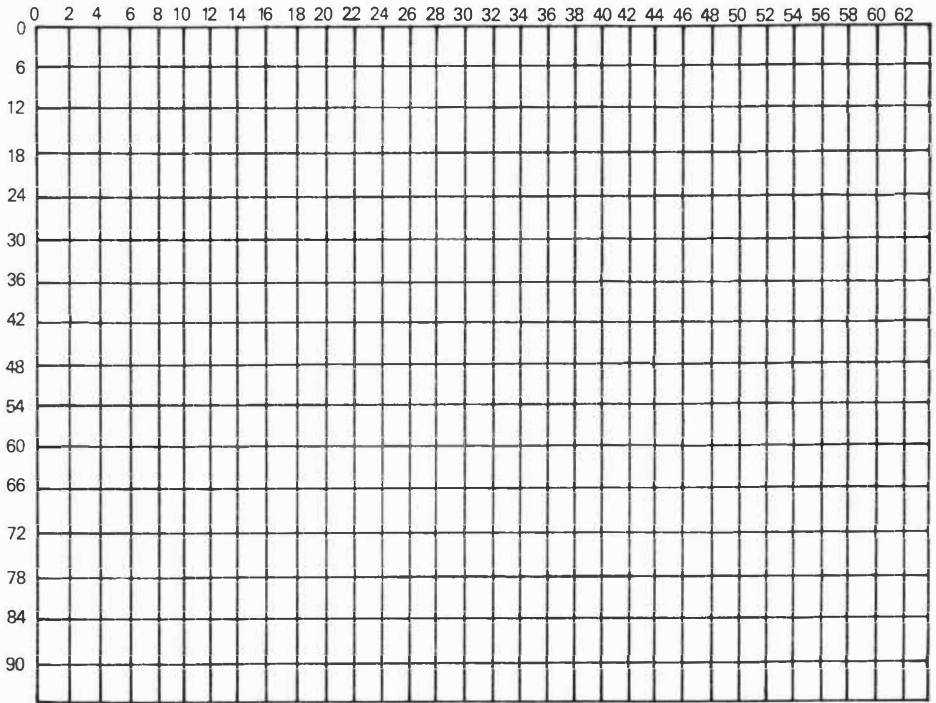
B6	GRAPHICS MODE
B7/B8	TOP OF CURRENT GRAPHICS SCREEN
B9	NUMBER OF BYTES PER ROW OF CURRENT GRAPHICS SCREEN
BA/BB	BASE OF CURRENT GRAPHICS SCREEN
BD/BE	CURRENT X POSITION ON SCREEN
BF/CO	CURRENT Y POSITION ON SCREEN
112-114	TIMER
120	NUMBER OF NORMAL RESERVED WORDS
121/122	POINTER TO START OF NORMAL RES. WORDS LIST
123/124	POINTER TO START OF ENTRY ADDRESSES
125	NUMBER OF FUNCTION RESERVED WORDS
126/127	POINTER TO START OF FUNCTION RESERVED WORD LIST
128/129	POINTER TO START OF FUNCTION ENTRY POINTS
134-147	USR ADDRESS TABLE
149	ALPHA LOCK FLAG
14A-14F	END OF LINE SEQUENCE FOR PRINTER
150-159	KEYBOARD ROLLOVER TABLE
15A-15D	JOYSTICK BUFFERS
15E-1AF	VECTOR TABLE FOR USER EXTENSIONS TO BASIC—DEFAULT RTS IN ALL BYTES. SOME ROUTINES USING THIS TABLE ARE:—
167-169	INPUT A CHARACTER
16A-16C	OUTPUT A CHARACTER
182-184	READ INPUT LINE
18B-18D	EVALUATE EXPRESSION
18E-190	ERROR HANDLER
194-196 19A-	RUN
19C 1A3-	READ NEXT STATEMENT
1A5 A6-1A8	CONVERT RES. WORDS TO TOKENS
1[illegible]	CONVERT TOKENS TO RES. WORDS
400-5FF	DEFAULT TEXT SCREEN
600-7FFF	BASIC WORK AREA SCREENS/PROGRAM/VARIABLES
8000-BFFF	BASIC ROM
C000-DFFF	CARTRIDGE ROM
FF00-FF03	PIA 1
FF20-FF23	PIA 2
FFCO-FFDF	SAM CONTROL BITS
FFFO-FFFF	CPU VECTORS (SET FROM BFFO-BFFF)
FFFO-FFF1	RESERVED
FFF2-FFF3	SW13
FFF4-FFF5	SW12
FFF6-FFF7	FIRQ
FFF8-FFF9	IRQ

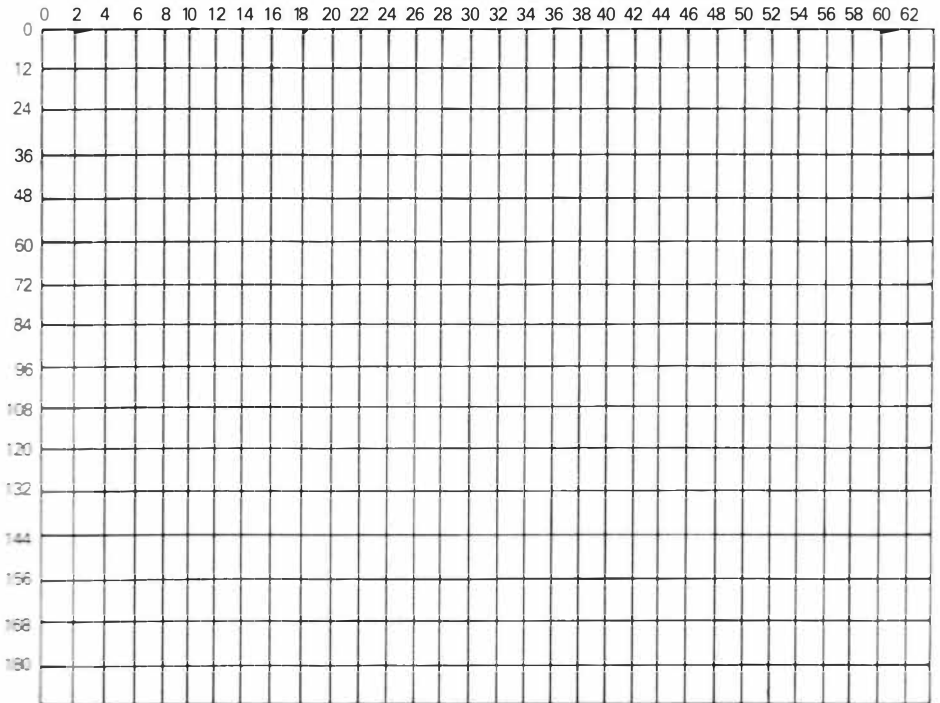
FFFA-FFFB	SWI
FFFC-FFFD	NMI
FFFE-FFFF	RESET

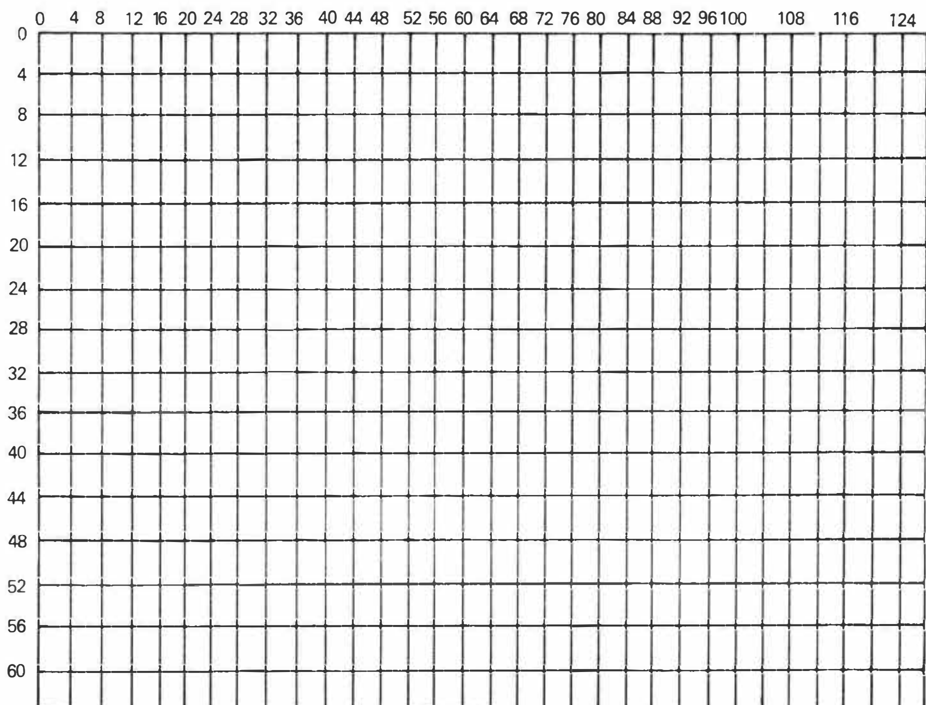

0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30

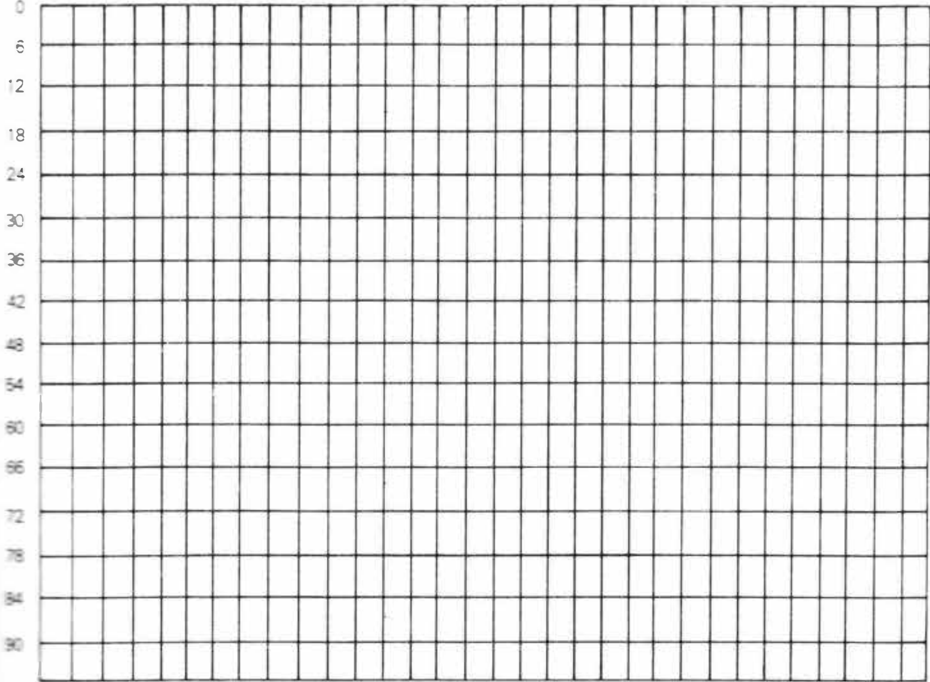

0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
52
54
56
58
60
62
0
3
6
8
12
15
18
21
24
27
30
33
36
39
42
45

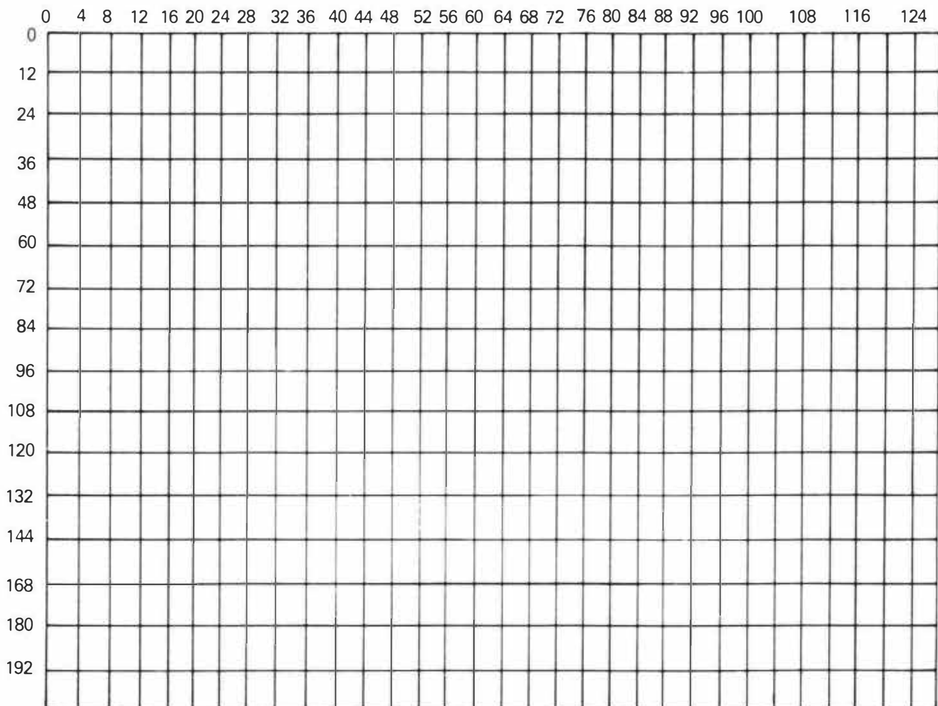

0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62
0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60

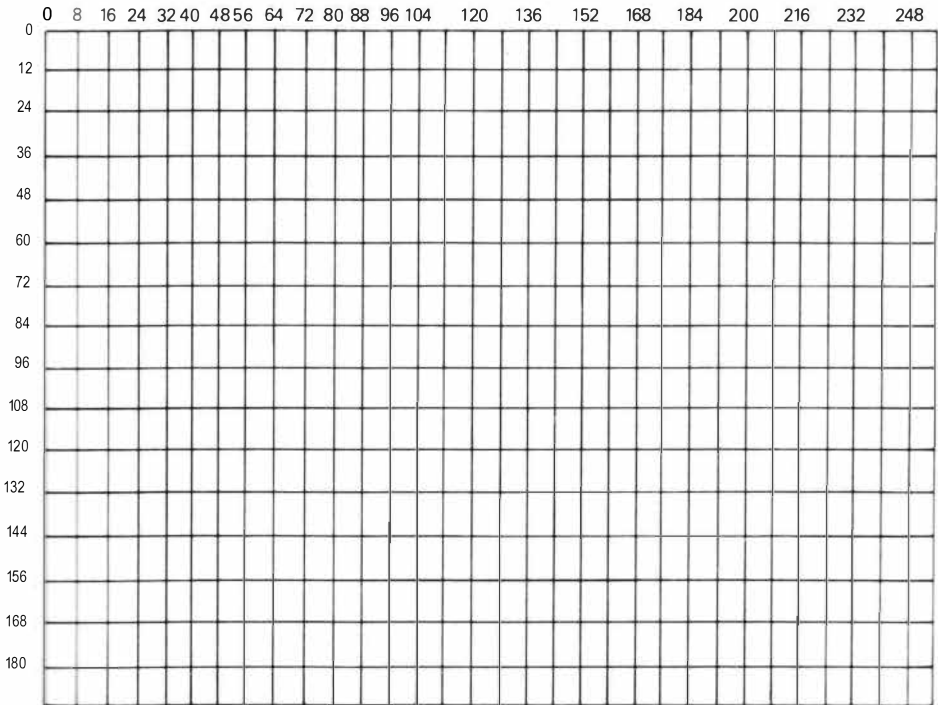

0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62
0
6
12
18
24
30
36
42
48
54
60
66
72
78
84
90


0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62
0
12
24
36
48
60
72
84
96
108
120
132
144
156
168
180


0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96 100 108 116 124
0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60


0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96 100 108 116 124
0 6 12 18 24 30 36 42 48 54 60 66 72 78 84 90


0
4
8
12
16
20
24
28
32
36
40
44
48
52
56
60
64
68
72
76
80
84
88
92
96
100
108
116
124
0
12
24
36
48
60
72
84
96
108
120
132
144
168
180
192


0
8
16
24
32
40
48
56
64
72
80
88
96
104
120
136
152
168
184
200
216
232
248
0
12
24
36
48
60
72
84
96
108
120
132
144
156
168
180