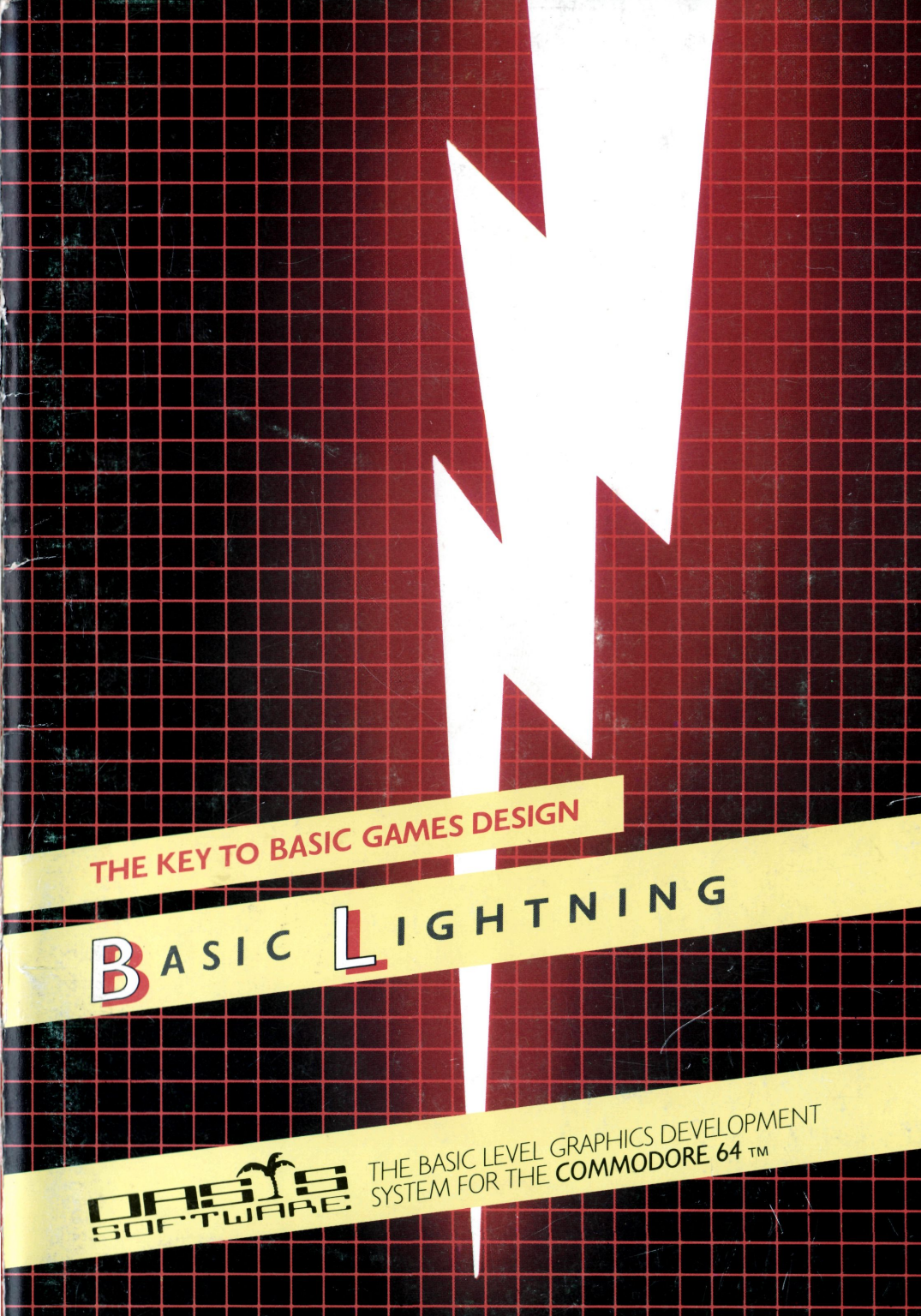

THE KEY TO BASIC GAMES DESIGN
BASIC LIGHTNING
OASIS SOFTWARE
THE BASIC LEVEL GRAPHICS DEVELOPMENT SYSTEM FOR THE COMMODORE 64 ™

BASIC LIGHTNING
by OASIS SOFTWARE

BASIC LIGHTNING
by OASIS SOFTWARE

Copyright Notice

Copyright © by Oasis Software. No part of this manual may be reproduced on any media without prior written permission from Oasis Software.

This Manual

Piracy has reached epidemic proportions and it is with regret that we are forced to reproduce this manual in a form which cannot be photocopied. Our apologies for the inconvenience this may cause to our genuine customers. A reward will be paid for information leading to the successful prosecution of parties infringing this Copyright Notice.

NOTE

This manual is essential for the use of Machine Lightning. For this reason we would warn customers to look after it very carefully, as separate manuals will not be issued under any circumstances whatsoever.

Copyright © by Oasis Software

™ Commodore is the Trade Mark of Commodore Business Machines Ltd.

CONTENTS

BASIC LIGHTNING

User Manual
by David Hunter

INTRODUCTION

BASIC LIGHTNING is an extension to the CBM64's resident BASIC interpreter which adds over 150 commands. The additional commands cover three main areas: structured programming, graphics and sound. One of BASIC LIGHTNING's most powerful features is the ability to multi-task; up to five parts of a BASIC program can be run at once. A compiler for BASIC LIGHTNING will be available in early 1985 which will produce "stand-alone" compiled programs which can be marketed without paying royalties.

The structured programming commands included in BASIC LIGHTNING include all the control commands found in PASCAL. This includes multiple-line IF-THEN-ELSE, REPEAT-UNTIL, WHILE-WEND as well as PROCEDURES with full parameter passing (including arrays), multiple-line functions and CASE-OF.

The sound and graphics commands included are the same as those in WHITE LIGHTNING, adapted to BASIC syntax, and include commands to plot points, draw lines, move software sprites to and from the screen, scroll blocks of the screen or sprites with character or pixel resolution, rotate, invert or mirror sprites or screen windows and exchange sprites with parts of the screen. You can also make use of the 64's inbuilt hardware sprites and it is possible to redefine the 64's character set.

To use this manual, it is essential that you already understand how to program using the 64's resident BASIC interpreter.

LOADING BASIC LIGHTNING

To load BASIC LIGHTNING from tape, type shift-run/stop and start the tape in the normal way. To load it from disk, type LOAD "BL",8,1.

1. STRUCTURED PROGRAMMING

The structured programming commands provided will allow you to program without using the "GOTO" statement and should result in programs which are easier to understand and modify.

Before proceeding, please note that commas are used in the LIST command instead of minus signs: for example, use LIST 100,200 instead of LIST 100-200. Also, keywords are no longer abbreviated using the shift key. Instead, a full stop is used. For example, "LIST" is abbreviated to "LI." not "L shift I". A list of all the keywords and abbreviations can be found in Appendix B.

Also, you can temporarily halt a listing by using the spacebar - press it again to re-start.

IF-THEN-ELSE

This is a simple extension to the existing IF-THEN statement. If the condition is true, the statements between THEN and ELSE are executed, otherwise the statements from the ELSE until the end of the line are executed. The ELSE can, of course, be omitted.

Putting more than one IF-THEN-ELSE on a line is allowed, for example:

```
IF a THEN PRINT 1 ELSE IF b THEN PRINT 2 ELSE PRINT 3
```

1 is printed if 'a' is true, irrespective of the value of 'b'.
2 is printed if 'a' is false and 'b' is true.
3 is printed if 'a' is false and 'b' is false.

However, the following is ambiguous:

```
IF a THEN IF b PRINT 1 ELSE PRINT 2
```

1 is printed if 'a' is true and 'b' is true.
2 is printed if 'a' is true and 'b' is false.

- the ELSE statement is assumed to belong to the most recent IF.

The THEN can be omitted if it is directly followed by a command (rather than an assignment), e.g.

```
IF I=6 PRINT "UNDEFINED"
```

CIF-CELSE-CEND

This is a more general version of IF-THEN-ELSE: it can be spread over any number of lines. The CIF is equivalent to IF, CELSE is equivalent to ELSE and CEND is used to mark the end of the statement. CIF, CELSE and CEND are separated from other statements by colons if more than one statement is put on the line.

Example:

```
100 LABEL stcheck          'subroutine to check ST
110 '
120 CIF st<>0
130   OPEN 15,8,15
140   INPUT#15,w$,x$,y$,z$
150   PRINT w$;",";x$;",";y$;",";z$
160   er=1
170   CLOSE 15
180 CELSE
190   er=0
200 CEND
210 RETURN
```

If you have typed in any of the examples so far, you will have noticed that all reserved words are printed in upper case when listing, and everything else is printed in lower case. This makes the program easier to read, particularly if many statements are put on one line without any spaces.

In the above listing, you will also see that indentation of two spaces has been used for statements inside the CIF-CEND. This makes it possible to see at a glance what the structure of a program is, although it uses up a lot of memory. Also, single quotes have been used for on-line commenting. A single quote is equivalent to ":REM". There is a LABEL statement in line 100 - this is dealt with next.

LABELS

The use of labels enables subroutines and data to be given symbolic names rather than being referenced by meaningless line numbers.

A label is defined by using the LABEL statement - for an example, look at the previous listing. The subroutine in that example would be called using "GOSUB stcheck".

The label itself must start with a letter, the rest of it consisting of letters, numbers, "$" and "%" signs. For example, the following are all legal labels:

```
z9    grid3%    i3vl    v$62
```

Unlike ordinary variables, all the characters are significant, not just the first two. Also, there is no restriction on the use of reserved words in labels; it would be perfectly legal to define a label called 'print' or 'teletype' for example. There is one exception to this rule; 'ELSE' cannot be included.

The RESTORE command has been extended so that it can be used with a line number or label. Also, ON-RESTORE can be used in the same way as ON-GOTO or ON-GOSUB.

Example:

```
10 INPUT "Skill level (1,2 or 3)";i
20 ON i RESTORE bl,wl,ml
30 READ a$
40 PRINT "loading ";a$;"..."
50 READ b$
60 LOAD b$,8,1
70 LABEL bl:DATA "basic lightning","bl"
80 LABEL wl:DATA "white lightning","wl"
90 LABEL ml:DATA "machine lightning","ml"
```

Labels are also used to define procedures and multiple-line functions; this is explained fully later.

REPEAT-UNTIL

REPEAT and UNTIL are used to set up a loop where it is required to repeat a set of statements at least once. The REPEAT command is put at the top of the loop, and the UNTIL is at the bottom of the loop. Every time the UNTIL is encountered, the expression after it is evaluated, and if it gives a FALSE value, the computer goes back to the corresponding REPEAT and starts the loop again.

To see how this works, type in the following and run it:

```
10 REPEAT
20   INPUT "4-character code";a$
30 UNTIL LEN(a$)=4
```

The computer will keep prompting for the string a$ until you type in a string of the correct length - four characters.

WHILE-WEND

WHILE and WEND are used to set up a loop where it is required to repeat a set of statements zero or more times. This is quite similar to the REPEAT-UNTIL loop; however, the test for exiting the loop is made at the top, not the bottom. Therefore, if the condition is not met when first entering the loop, it will not be executed at all.

Try running this program:

```
10 INPUT x$
20 WHILE x$<>""
30   PRINT x$
40   x$=RIGHT$(x$,LEN(x$)-1)
50 WEND
60 PRINT
70 PRINT" * * * end * * * "
```

If you type in "while-wend", the following will be printed:

```
while-wend
hile-wend
ile-wend
le-wend
e-wend
-wend
wend
end
nd
d

* * * end * * *
```

However, if you type in a null string (two quotes and RETURN), you will see that the loop itself is not executed at all since x$<>"" gives a FALSE value the first time around.

Since TRUE and FALSE return values of -1 and 0 respectively, it is possible to set up an infinite loop using:

```
      REPEAT .... UNTIL FALSE
or    WHILE TRUE .... WEND
```

EXIT

Sometimes it is convenient to exit from a loop prematurely, and the EXIT command is provided to do this. For example, let's look at the example that was used before for the REPEAT-UNTIL loop:

```
10 REPEAT
20   INPUT "4-character code";a$
30 UNTIL LEN(a$)=4
```

Suppose that we want to give an error message if the code is not 4 characters long:

```
10 REPEAT
20   INPUT "4-character code";a$
25   IF LEN(a$)<>4 PRINT "4 characters please - try again"
30 UNTIL LEN(a$)=4
```

This could be more efficiently coded using EXIT, since in the above, the test for the correct length of a$ has to be made twice:

```
10 REPEAT
20   INPUT "4-character code";a$
25   IF LEN(a$)=4 EXIT
27   PRINT "4 characters please - try again"
30 UNTIL FALSE
```

If the loop conditions are more complex, EXIT can save a lot of re-typing. EXIT can also be used to leave WHILE-WEND and FOR-NEXT loops.

NOTE: It is not advisable to jump out of a loop using a GOTO statement as this corrupts the stack and may cause error messages later in the program.

CASE-OF-CASEND

The CASE statement is used to select between a number of alternative courses of action, depending on the value of a variable which may be either numerical or string.

The CASE is used at the top of the statement and is followed by the variable to be tested. After each OF there is a list of values, separated by commas. If the variable being tested is equal to one of these expressions, the code up until the next OF or CASEND is executed, otherwise it is ignored. The CASEND is put at the end of the CASE statement. "OF OR" is interpreted as an instruction to execute the following code if none of the statements so far have been executed.

Example:

```
10 INPUT a
20 CASE a
30 OF 3:PRINT "Three French hens,"
40 OF 2,3:PRINT "Two turtle doves,"
50 OF 1,2,3:PRINT "And a partridge in a pear tree."
60 OF OR : STOP
70 CASEND
```

PROCEDURES

Procedures, like subroutines, are sections of code which are used several times in a program. However, they are much more powerful than subroutines because it is possible to pass parameters to them.

Clear the memory using NEW, then type in the following:

```
10 LABEL test(i)
20 LOCAL j
30 FOR j=1 to i
40   PRINTj;
50 NEXTj
60 PRINT
70 PROCEND
```

(the computer will automatically convert the keywords to upper case).

Now type " proctest(10) ", and 1 2 3 4 5 6 7 8 9 10 is printed.

The "10" in proctest(10) is called an actual parameter, and the variable i is called the formal parameter. When the procedure is called, the formal parameter is made equal to the actual parameter, and the body of the procedure is executed. All formal parameters are local to the procedure - this means that they are separate from any variables of the same name that are used in the rest of the program. It is also possible to create additional local variables for use within a procedure using the LOCAL command, as in line 20. If there is a variable called j in the main program, it will not be altered by the procedure because j is local. To prove this, type the following:

```
for j=1 to 12 : proctest(j) : next

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10 11
1 2 3 4 5 6 7 8 9 10 11 12
```

As you can see, the two variable j's are separate and do not interfere with one another.

Parameter passing is not restricted to reals - integers and strings can be passed as well. Any expression can be used as an actual parameter.

To illustrate passing strings as parameters, type in the following procedure:

```
100 LABEL centre (a$,i)
110 PRINT@0,i;"                                        "   '40 spaces
120 PRINT@20 - LEN(a$)/2,i;a$
130 PROCEND
```

Note that this procedure has two parameters instead of one as in the previous example - the number of parameters a procedure can have is only limited by the maximum line length of 80 characters.

In lines 110 and 120, PRINT@ is used: PRINT@X,Y; will position the cursor on screen column X and row Y before printing the data following it.

Type 'proccentre ("basic lightning",10)', and "basic lightning" will be printed on the middle of the tenth line of the screen.

The LOCAL command can be used to create more than one variable by separating the names by commas. Local integers and strings can be defined also, and if more variables are required than can fit on one line, several LOCAL commands can be used on separate consecutive lines. LOCAL, if used, is normally the first command in the procedure.

VAR Parameters

In the examples you have seen so far, none of the procedures have had to alter any of the actual parameters. Indeed, you will find that if you modify any of the previous examples so that a formal parameter is changed inside the procedure, the corresponding actual parameter will remain unaltered.

If a procedure is to modify any of its parameters, they must be declared as VAR parameters in the procedure heading, like this:

```
100 LABEL inp(A$,VAR B$)
110 PRINT@0,10;a$;
120 INPUTb$
130 PRINT@0,10;"                                        "    '40 spaces
140 PROCEND
```

Now clear the screen using SHIFT-CLR and type:

```
PROCinp ("Name",x$)
```

The computer will execute the procedure "inp" and input a string from the keyboard. After this, if you type ?x$ you will see that x$ is equal to what you typed in, i.e. the formal parameter b$ was altered inside the procedure and this has resulted in x$ being changed as well.

Note that if a parameter is declared in a procedure as a VAR parameter, the corresponding actual parameter must be a variable name, not an expression. Also, you are not allowed to pass single elements of arrays as VAR parameters.

Arrays as Parameters

You can also pass whole arrays as parameters to procedures, although they must be declared as VAR parameters. Also, an actual parameter array must have been DIMensioned before calling the procedure, even if its dimensions are less than the default value of 10.

When arrays are passed as parameters to procedures, we often do not know what the size of the array will be when writing the procedure. For this reason, the SIZE function is provided in BASIC LIGHTNING. For example, type in the following:

```
DIM x(10,20)
```

Now, SIZE(x,0) will give a value of 2, the number of dimensions. SIZE(x,1) gives 21, the second dimension + 1 and SIZE(x,2) gives 11, the first dimension + 1.

Here is a simple program which illustrates the use of arrays as parameters:

```
 10 INPUT n                          'get number of records
 20 DIM a$(n)
 30 PROC enter(a$())                 'get records
 40 PROC sort(a$())                  'sort
 50 PROC out(a$())                   'and print
 60 END
 70 '
 80 LABEL enter (VAR x$())
 90 LOCAL i
100 FOR i=1 TO SIZE (x$,1)-1
110   INPUT x$(i)
120 NEXT i
130 PROCEND
140 '
150 LABEL sort (VAR y$())
160 LOCAL i,j,k$
170 REPEAT
180   j=TRUE
190   FOR i=1 TO SIZE (y$,1)-2
200     CIF y$ (i) > y$(i+1)
210       k$=y$(i)                  'swap two elements
220       y$(i)=y$(i+1)             'if out of sequence
230       y$(i+1)=k$
240       j=FALSE
250     CEND
260   NEXT i
270 UNTIL j                         'until elements in order
280 PROCEND
290 '
300 LABEL out (VAR z$())
310 LOCAL i
320 FOR i=1 TO SIZE (z$,1)-1
330   PRINT z$(i),
340 NEXT i
350 PROCEND
```

MULTIPLE-LINE FUNCTIONS

Multiple-line functions are similar to procedures, the difference being that they return a value; the end of the function is an "=" sign followed by the value to be returned. To call the function, the keyword "CFN" is followed by the label name and the parameters inside brackets, as with procedures. As an example, here is a function to calculate the factorial of a number:

```
10 LABEL fact(n)
20 LOCAL i,j
30 i=1
40 CIF n>1
50   FOR j=2 TO n
60     i=i*j
70   NEXT j
80 CEND
90 = i
```

When this has been typed in, CFNfact(i) will return the factorial of a number. For example, ?CFNfact(4) will print '24'.

2. MISCELLANEOUS KEYWORDS

Hexadecimal Numbers

Hexadecimal numbers can be used in expressions by preceding them with a '$' sign. For example:

```
? $9800/2
```

will give a result of 19456.

It is also possible to convert a numerical result into hex using the string function HEX$, e.g. ? HEX$ (19456) will give $4C00. The string is always five characters long - the '$' plus four digits. The number being converted must be in the range 0 to 65535.

DEEK and DOKE

These are similar to PEEK and POKE, but they operate on two bytes at once instead of one only, using the usual 6502 low byte/high byte order.

DEEK returns a value between 0 and 65535; for example ? DEEK (65532) will give the same result as ? PEEK (65532) + 256 * PEEK (65533).

DOKE $C000,$55FF is equivalent to POKE $C000,$FF : POKE $C001,$55.

DISABLE can be used to disable the STOP key. This will only work if it is included in a program since it is automatically re-enabled every time you type in a line in immediate (or "command") mode. There is no way to break out of a program once this command has been used, unless an error occurs.

DLOAD and DSAVE

These are very similar to LOAD and SAVE, but they assume that you are using a disk drive (device No. 8) rather than a tape recorder (device No. 1).

Example: DLOAD "SPIGEN"

PULL

This command removes the return line number put on the stack by GOSUB from the stack. It is useful in menu-driven programs where it is required to move back to a higher level menu, in which case the subroutine to do this would be PULL:RETURN.

PI

PI is a reserved word which will return the value 3.14159265.

DIR

If you have a disk drive, DIR can be used to print out the disk directory. As with LOAD or SAVE, the filename and device number can be specified; DIR"$0:O*",9 would print out all filenames on device number 9 which start with "O". For further details, refer to your disk drive manual.

OLD

This command will attempt to restore a program which has been NEWed. If you have typed in program lines or created new variables since typing NEW, it will not be successful.

3. GRAPHICS COMMANDS

The graphics commands included in BASIC LIGHTNING are designed to allow easy manipulation of images on and off the screen. This is achieved by carrying out operations on a table of sprites which can initially hold up to 8k of data. In this context, "sprite" means a graphics character of user-definable dimensions (up to 255 characters height or width) which may be displayed on the screen in one of the high-resolution modes (either bit-map mode or multi-colour bit-map mode). This is not to be confused with the 64's own hardware sprites - these can also be used from BASIC LIGHTNING and will be dealt with later.

Each sprite in the table is given a number from 1 to 255, and the screen is treated as sprite number zero - thus, the same commands can be used for both the screen and the sprites.

SPRITE VARIABLES

The sprite graphics commands in BASIC LIGHTNING use thirteen variables to pass parameters. The variables are:

SPN	sprite no. 1
COL	column in sprite no. 1
ROW	row in sprite no. 1
WID	width of window
HGT	height of window
SPN2	sprite no. 2
COL2	column in sprite no. 2
ROW2	row in sprite no. 2
NUM	number of sides on polygon or number of pixels to scroll
INC	inclination of polygon
ATR	current attribute
CCOL	column for collision detection
CROW	row for collision detection

(The way that these variables are used will become clear later).

They can be treated like normal variables; i.e. they can be assigned a value:

```
SPN = 3          (Don't use let)
```

or used in expressions:

```
PRINT SPN-1
```

Most of the graphics commands can be used either with or without the parameters after them. For example, the command WCLR uses SPN, COL, ROW, WID and HGT as parameters. Thus WCLR 0,8,8,4,4 is equivalent to :

```
SPN=0:COL=8:ROW=8:WID=4:HGT=4:WCLR
```

The commands which can be used withiut parameters in this way are marked with an asterisc in appendix A, part 2. Note that putting parameters after a command will still alter the variables. The first sprite commands that we will look at are RESET, SPRITE and WIPE, which are used to create and delete sprites in the table.

SPRITE UTILITIES

RESET has no parameters, and it simply removes all the sprites from the table.

SPRITE SPN,WID,HGT creates a new sprite in the table with number SPN, width WID and height HGT character blocks. All the data in the sprite is cleared when it is created. For example, SPRITE 1,16,16 would create space for a sprite number 1, 16 character blocks square. Each character block in the sprite takes up ten bytes - eight bytes for the pixel data, one byte for the primary attribute data and one byte for the secondary attribute data. (The attributes determine what colour the pixel data will be displayed in - only the primary attributes are used when in two-colour mode). Also, there is an overhead of seven bytes for each sprite.

WIPE will remove sprite no. SPN from the table, releasing the space for new sprites. Note that trying to delete a sprite which doesn't exist will give an error message, and you are not allowed to delete sprite no. zero, the screen. Also, you cannot create a new sprite using SPRITE if a sprite with that number already exists.

DFA(SPN), AFA(SPN) and AFA2(SPN) are functions which return the starting addresses of the pixel data, the primary attribute data and the secondary attribute data of sprite SPN respectively. They also set WID and HGT to the size of the sprite. If a sprite is undefined, they return values of -1. So a procedure to delete a sprite without giving an error if it doesn't exist is:

```
100 LABEL:delete(sprnum)
110 IF DFA(sprnum)>-1 WIPE sprnum
120 PROCEND
```

Since the hi-res screen is "hidden" under the KERNAL ROM, you cannot read data from them directly using PEEK.

SAVING and LOADING Sprites

The command STORE saves the sprites currently in memory to tape. To save sprites to tape, use 'STORE' or 'STORE "filename"'. To save sprites to disk, use 'DSTORE "filename"'. As with SAVE, a device number can follow the filename so that you can use more than one disk drive.

Sprites can be loaded using 'RECALL' or 'RECALL "filename"' for tape, and 'DRECALL "filename"' for disk.

RECALL will overwrite any sprites that are already in memory - if you want to keep the sprites that are already there, you must use the MERGE command which has the same syntax as RECALL. There is also an equivalent command for disk - DMERGE.

Note that there is nothing to stop you from loading in more sprite data than there is room for in memory - this will overwrite the character memory and cause a "CORRUPTED SPRITE" error, so be careful!

Although only 8192 bytes are available for sprites after you load BASIC LIGHTNING, you can have up to 26621 bytes to store sprites in - to reserve n bytes, use RESERVE n. n should be bigger than 8191 and less than 26621. Of course, reserving more space for sprites leaves you with less room for your BASIC program.

If you have just MERGED more sprites onto those in memory, it is possible that more than one sprite will have the same number. You can renumber all the sprites using the RESEQ command which renumbers them from 1 in steps of 1.

DISPLAY MODES

Data can be displayed on the hi-res screen in either two-colour mode or four-colour mode. In two-colour mode, each character block contains 64 (8x8) pixels. In four-colour mode, each pixel can take on one of four colours, but each character block contains only 32 pixels, since each pixel is twice as wide as in two colour-mode. MONO and MULTI put the hardware into two-colour and four-colour modes respectively.

S2COL and S4COL govern whether the sprite commands operate on two-colour data or four-colour (multi-colour mode) data. Thus it is possible to display a picture on the screen in two-colour mode while preparing data to be displayed in four-colour mode.

SETTING THE ATTRIBUTE VALUE

SETATR is a command which sets up the value of ATR, the current attribute value. When in S2COL mode, it takes the form: SETATR 0,foreground,background. "foreground" and "background" are one of the following:

BLACK	WHITE	RED	CYAN
PURPLE	GREEN	BLUE	YELLOW
ORANGE	BROWN	.RED	GRAY1
GRAY2	.GREEN	.BLUE	GRAY3

(These are in fact predefined constants which return values in the range 0 to 15).

A full stop before the colour should be read 'light' - e.g. ".GREEN" is light green.

In S4COL mode, use:

SETATR colour3,colour1,colour2 (note the order!)

(Colour zero is the same for the whole screen).

PAPER, BORDER AND INK COLOURS

TPAPER, TBORDER, HPAPER and HBORDER define the paper (background) and border colours used in LORES (TEXT) mode and HIRES mode. For example:

```
TPAPER BROWN:TBORDER YELLOW
```

will give a brown background with a yellow border when in text mode, and

```
HBORDER BLACK
```

will give a black border when in hi-res mode.

Also 'INK colour' will set the colour used when printing characters in text mode.

The command LORES puts the screen into text mode, while the command HIRES puts it into hi-res mode. To put a text window on the screen, use WINDOW n which will make the top n lines of the screen hi-res, and the rest text. For example, WINDOW 16 will give you 16 lines of hi-res at the top of the screen and 9 lines of text at the bottom. When a window is set up, the hi-res border colour is used. Note also that use of a disk drive while a window is set up will make it flicker.

Now we will look at some words which place data inside sprites:

PLOT, BOX, DRAW, POLY and POINT

First type "WINDOW 16" to set up a screen window. Unless you have already used the computer, the upper part of the screen will be filled with garbage which must be removed.

SCLR SPN,ATR clears all the pixel data in a sprite SPN and sets the attributes to ATR. Type "ATTON" (this is explained later) and then "SETATR 0,BLACK,WHITE" to set up the attribute used. Now, if you type "SCLR 0,ATR" the upper part of the screen will be cleared. Note that an "SCLR" is done automatically when you create a sprite using "SPRITE".

PLOT is used to set or clear individual points in a sprite; the format is:

```
PLOT  SPN,COL,ROW
```

If you type "PLOT 0,0,0" you will see that the point at the top left corner of the screen is set. Try using other values of COL and ROW. The maximum value of COL is 319, while the maximum value of ROW is 199, although you cannot see any points plotted with ROW greater than 127 because you have set up a text window.

PLOT can also be used to clear points in a sprite to the background colour or to toggle (invert) a point; this is achieved by using the word MODE first. MODE 0 or MODE 1 will cause points to be set to background colour, while MODE 2 or MODE 3 will cause them to be set to the foreground colour. If you use MODE 4, the points will be inverted.

In S4COL mode, 0, 1, 2 and 3 correspond to the background colour, colour 1, colour 2 and colour 3 respectively (see the section "SETTING THE ATTRIBUTE VALUE"), and mode 4 will cause colour 3 to change to background, colour 2 to change to colour 1 and vice versa. For example, if you now type MODE 0:PLOT 0,0,0 the point at the top left of the screen that you set earlier will be cleared.

```
MODE 4:FOR I=0 TO 100:PLOT 0,0,0:FOR J=0 TO 100:NEXT J,I
```

will make it flash on and off.

MODE also affects the BOX, DRAW and POLY commands.

DRAW SPN,COL,ROW,COL2,ROW2 draws a line from one point to another. For example:

```
DRAW 0,0,0,319,127
```

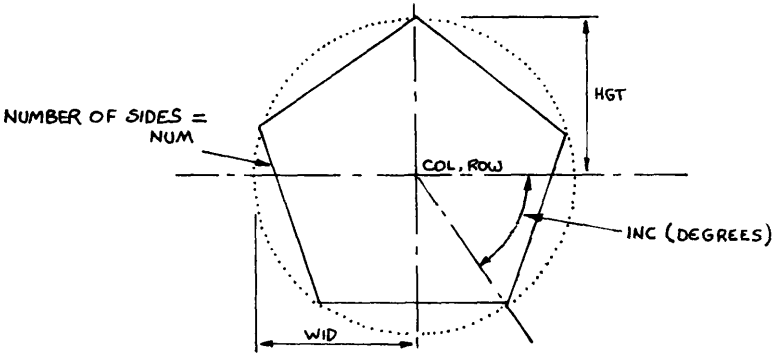
will draw a diagonal line across the screen.

BOX SPN,COL,ROW,WID,HGT plots a rectangular block inside a sprite. The top left corner is at COL,ROW. WID is the width and HGT is the height of the block.

```
MODE 3:BOX 0,150,54,20,20
```

will draw a 20x20 pixel square at the centre of the screen.

POLY SPN,COL,ROW,WID,HGT,NUM,INC draws a polygon on the screen:



COL and ROW are the centre of the polygon. WID is the horizontal radius and HGT is the vertical radius. INC is the inclination in degrees and NUM is the number of sides. If NUM is large, or less than three, a circle (or ellipse) is drawn instead of a polygon. For example, POLY 0,32,32,32,32,5,0 draws a pentagon in the top left of the screen. POLY 0,32,32,32,32,5,36 draws another one over it, while POLY 0,32,32,32,32,0,0 draws a circle.

Here is a program which illustrates the use of the DRAW and POLY commands:

```
10 SETATR 0,BLACK,.GREEN
20 S2COL
30 SCLR 0,ATR
40 MODE 3
50 MONO
60 HIRES
70 '
80 PROCquad(100,160,48)
90 '
```

(Continued over page)

(continued from previous page)

```
100 REPEAT
110 UNTIL the cows come home
120 '
130 LABELquad(qrw,qcl,qsz)
140 CIF qsz>2
150   POLY 0,qcl,qrw,qsz/1.5,qsz/1.5,0,0
160   PROCquad(qrw+qsz,qcl,qsz/2)
170   PROCquad(qrw,qcl+qsz,qsz/2)
180   PROCquad(qrw-qsz,qcl,qsz/2)
190   PROCquad(qrw,qcl-qsz,qsz/2)
200   DRAW 0,qcl,qrw-qsz,qcl,qrw+qsz
210   DRAW 0,qcl-qsz,qrw,qcl+qsz,qrw
220 CEND
230 PROCEND
```

Notice that the procedure 'quad' in line 130 is recursive; it calls itself.

POINT (SPN,COL,ROW) is a function which is used to examine a pixel on the hi-res screen. In S2COL mode, it will return a value of 0 or 1, corresponding to the point being cleared or set respectively. In S4COL mode, it returns 0,1,2 or 3 corresponding to background or the three colours.

Although the pixels are twice as wide in multi-colour mode, the scaling when using COL is still the same; in S4COL mode, COL=0 and COL=1 will refer to the same pixel.

SPRITE DATA MOVEMENT COMMANDS

BASIC LIGHTNING includes 39 commands for moving rectangular blocks of sprite data:

MOVBLK	PUTBLK	GETBLK	CPYBLK
MOVOR	PUTOR	GETOR	CPYOR
MOVXOR	PUTXOR	GETXOR	CPYXOR
MOVAND	PUTAND	GETAND	CPYAND
BLK%BLK	OR%BLK	XOR%BLK	AND%BLK
BLK%OR	OR%OR	XOR%OR	AND%OR
BLK%XOR	OR%XOR	XOR%XOR	AND%XOR
BLK%AND	OR%AND	XOR%AND	AND%AND
MOVATT	SWAPATT		
ATTOFF	ATTON	ATT2ON	
DICTON	DICTOFF		

Note that the screen is treated throughout as Sprite 0 with height 25 characters and width 40 characters.

ATTOFF,ATTON and ATT2ON control the movement of attribute data with the pixel data. After executing the ATTOFF command, attribute data movement is disabled. Only the primary attribute data is moved after ATTON, and both sets (primary and secondary data) are moved after ATT2ON. The screen's secondary attribute data uses the same memory as the text colour memory, so you must not scroll the text screen if you are displaying data in multi-colour mode.

Thus, to go into two-colour mode, use

```
MONO:S2COL:ATTON
```

and for four-colour mode, use

```
MULTI:S4COL:ATT2ON
```

ONE-WAY DATA MOVEMENT

All the data movement commands have been named to make them easy to remember:

SUFFIXES:

BLK the sprite data overwrites its destination.
OR the sprite data is ORed with its destination.
AND the sprite data is ANDed with its destination.
XOR the sprite data is exclusive-ORed with its destination.

PREFIXES:

MOV parameters: SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
A window of width WID and height HGT in sprite SPN whose top left-hand corner is at COL,ROW is MOVed into SPN2 at COL2 and ROW2.

PUT parameters: SPN,COL,ROW
Sprite no. SPN is PUT onto the screen with its top left corner at COL,ROW

GET parameters: SPN,COL,ROW
GETs the sprite no. SPN from the screen at position COL,ROW

CPY parameters: SPN,SPN2
COPYs sprite no. SPN into sprite no. SPN2

The combinations of the four prefixes and four suffixes given yield the first sixteen commands given above.

The logical operations OR, AND and XOR will be familiar to anyone with a knowledge of Boolean algebra; however, it is easy to understand what they do in terms of pixels:

OR The destination pixel is set if either the source OR the destination pixel is set.

AND The destination pixel is set only if the source AND the destination pixels are set.

XOR The destination pixel is set only if one, but not both, of the source and destination pixels are set.

XOR is particularly useful when it is necessary to move a sprite over some background data - the sprite can be put on the screen using a PUTXOR and the background can be restored again with another PUTXOR.

Clear the computer's memory using NEW and type in the following:

```
 10  MONO:S2COL:ATTON:RESET
 20  '
 30  SETATR 0,BLACK,RED
 40  SPRITE 1,16,16
 50  '
 60  SETATR 0,BLACK,.GREEN
 70  SCLR 0,ATR
 80  WINDOW 16
 90  '
100  POLY 1,64,64,64,64,5,0
110  POLY 0,64,64,64,64,5,36
```

Line 10 puts the graphics into two-colour mode. Lines 20 and 30 dimension sprite no. 1 to be 16 character blocks square with black foreground and red background. 60 to 80 set up the screen window. 100 draws a pentagon inside sprite no. 1, 110 draws one at an angle on the screen.

RUN the program, and type PUTBLK 1,0,0.

You will see that sprite no. 1 is put on the screen, and it completely overwrites what was there before. Also, the attributes from sprite 1 are moved (i.e. black lines with red background) because of the ATTON in line 10.

RUN the program again, and type PUTOR 1,0,0. Sprite 1 is "put on top of" any data that it is already on the screen.

After RUNning it again, type PUTAND 1,0,0. The only points that are left on the screen are where the lines from the two pentagons cross.

RUN it again, followed by PUTXOR 1,0,0. The resulting display is similar to that from PUTOR, but the points are cleared where the lines cross. Type PUTXOR 1,0,0 again, and the second pentagon vanishes, leaving the first one as it was before.

If you change the ATTON in line 10 to an ATTOFF, the screen colour will not be changed when you type the MOV command.

After changing line 10, run the program and type PUTOR 1,0,0, followed by GETBLK 1,0,0. Sprite 1 now contains the double pentagon that is on the screen, and this can be placed on the screen at any point using PUTBLK. If you try to put the sprite wholly or partially off the screen (eg. PUTBLK 1,-1,-1), BASIC LIGHTNING does not give an error message, but places as much of the sprite on the screen as is possible. This automatic adjustment applies to all the data movement commands.

If you type SPRITE 2,16,16:CPYBLK 1,2 and PUT sprite 2 onto the screen, you will see that sprite 1 has been copied into sprite 2.

The MOV commands are much more general than the PUTs, GETs and CPYs and can be used to copy a window from any position in one sprite to any position in another. For example, MOVBLK 0,0,0,4,4,0,36,12 will copy a 4x4 block from the top left-hand corner of the screen to halfway down the right-hand side.

When using AND,OR and XOR with multi-colour mode, the situation is more complex, and this is summarised below for the advanced user:

source colour	dest colour	dest. colour after BLK	OR	AND	XOR
0	**0**	**0**	0	0	0
0	**1**	**0**	1	0	1
0	**2**	**0**	2	0	2
0	**3**	**0**	3	0	3
1	**0**	**1**	1	0	1
1	**1**	**1**	1	1	0
1	**2**	**1**	3	0	3
1	**3**	**1**	3	1	2
2	**0**	**2**	2	0	2
2	**1**	**2**	3	0	3
2	**2**	**2**	2	2	0
2	**3**	**2**	3	2	1
3	**0**	**3**	3	0	3
3	**1**	**3**	3	1	2
3	**2**	**3**	3	2	1
3	**3**	**3**	3	3	0

(NB colour 0 means the background.)

TWO-WAY DATA MOVEMENT COMMANDS

The second set of sixteen data movement commands are the two-way move commands:

BLK%BLK	OR%BLK	XOR%BLK	AND%BLK
BLK%OR	OR%OR	XOR%OR	AND%OR
BLK%XOR	OR%XOR	XOR%XOR	AND%XOR
BLK%AND	OR%AND	XOR%AND	AND%AND

As you can see, they each consist of two of the logical operators BLK, OR, XOR or AND separated by a '%'.

The first operator specifies the logical operation for the data from SPN2 going into SPN, the second is the logical operation for the data going into SPN2 from SPN. All these commands have the same parameters:

SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2

SPN,COL and ROW specify the top left-hand corner of window no. 1, while SPN2, COL2 and ROW2 give the top left-hand corner of window no. 2.

Data is moved between the two windows simultaneously, the logical operations operating in exactly the same way as with the single-way move commands.

MOVING ATTRIBUTES

Two commands are provided to move blocks of attributes:

MOVATT SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
SWAPATT SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2

They operate in a similar way to MOVBLK and BLK%BLK, moving the attributes only. If in ATTON mode, the primary set of attrbutes only is moved, while both primary and secondary sets are moved in ATT2ON mode.

COLLISION DETECTION

Collision detection is enabled by the DTCTON command and disabled by DTCTOFF. (It is always best to switch it off when not required since it slows down data movement slightly.)

The pseudo-variables CCOL and CROW contain the column and row of the collision after sprite data has been moved. If a collision has not occurred, CCOL and CROW are both set to -1.

In the case of single-way move commands, CCOL and CROW will record the position in the destination sprite. With two-way moves CCOL and CROW give the position in SPN2. (Note that the collision is detected by examining the data before it is moved.)

The position in CCOL and CROW is the first collision found - the data is moved starting at the top left-hand corner going from left to right along each line of character blocks.

In multi-colour mode, colour1 and the background are both regarded as transparent to collisions - only colour2 and colour3 will result in a collision being detected.

CLEARING AND INVERTING WINDOWS

SCLR SPN,ATR was mentioned earlier and is used to clear a whole sprite, setting the attributes to ATR. The secondary attributes are only set if in ATT2ON mode.

WCLR SPN,COL,ROW,WID,HGT,ATR is similiar to SCLR, but it only clears a window inside SPN rather than the whole sprite.

For example, if you put some data on the screen;

```
POLY  0,64,64,64,64,5,0
POLY  0,64,64,64,64,5,36
```

then the bottom right-hand corner of the figure is removed by:

```
WCLR  0,8,8,8,8,ATR
```

SETA SPN,COL,ROW,WID,HGT,ATR is used to initialise only the attributes in a window. As with SCLR and WCLR, the secondary attributes are not altered unless in ATT2ON mode.

For example try:

```
SETATR  0,BLACK,PURPLE:SETA 0,0,0,16,16,ATR
```

INV SPN,COL,ROW,WID,HGT will invert the pixel data in the specified window. If in MONO mode, background is changed to foreground and vice-versa. In MULTI mode, background is changed to colour 3, colour 1 is changed to colour 2 and vice-versa.

SCAN(SPN,COL,ROW,WID,HGT) is a function which will return -1 (true) if there is data in the specified window. For example, PRINT SCAN(0,0,0,16,16) will give -1 if the screen is still set up with the display from the WCLR example.

ATTGET SPN,COL,ROW will look at the attribute value at the specified character block and place the attribute in the pseudo-variable ATR.

SCROLLING COMMANDS

BASIC LIGHTNING has the following commands for scrolling data and attributes in sprite windows:

```
SCR1    WRR1    SCL1    WRL1
SCR2    WRR2    SCL2    WRL2
SCR8    WRR8    SCL8    WRL8

SCROLL  WRAP

ATTUP   ATTDN   ATTL    ATTR
```

Clear the memory using NEW, and type in the following program:

```
10  MONO
20  S2COL
30  ATTON
40  SCLR  0,3
50  WINDOW 16
60  POLY  0,32,32,32,32,4,0
70  POLY  0,32,32,32,32,0,0
```

RUN it; this puts some information on the screen to be scrolled.

The first twelve commands scroll or "wrap" data left or right by 1,2 or 8 pixels. With a scroll command, any data shifted off the edge is lost and blanks are shifted into the other side. In the case of the wrap commands, data which is shifted off one edge re-appears at the other side.

"SC" at the start of a command means "scroll" and "WR" means "wrap". The third letter is either "R" for right or "L" for left. The digit at the end is the number of pixels being scrolled in 2-colour mode. All 12 commands have the same parameters, SPN,COL,ROW,WID,HGT which define the window in which the scrolling is to take place. COL,ROW is the top left hand corner. None of these commands alter the attribute data.

If you type SCR1 0,0,0,8,8 you will see that the figure on the screen is shifted right by one pixel. To repeat a command several times, you can use "RPT n," before the command where n is the number of times that it is to be repeated. If you now type RPT 63,SCR1 the figure will be scrolled out of the window (remember that commands can be used without parameters as an abbreviation). RUN the program again and type "RPT 64,WRR1 0,0,0,8,8". The pixel data re-appears at the left of the window as it is shifted out of the right side.

The other ten commands in this group function in a similar way, only differing in the direction of scrolling or number of pixels scrolled. One WRR2 is not much faster than two WRR1s - the two-pixel scrolls are intended for use in multi-colour mode where one-pixel scrolls cannot be used.

WRAP and SCROLL are used to scroll vertically. The parameters are:

```
SPN,COL,ROW,WID,HGT,NUM
```

NUM is the number of pixels to be scrolled - a positive value indicates scrolling up and a negative value is used for scrolling down. The maximum value allowed is 127 (or -127).

ATTUP,ATTDN,ATTL and ATTR are used to scroll attributes by one character block up, down, left and right. The parameters for these commands are:

```
SPN,COL,ROW,WID,HGT
```

Type in the following lines over the program that you already have in the memory:

```
 60  SETATR 0,BLACK,RED
 70  SETA 0,0,0,4,4,ATR
 80  SETATR 0,BLACK,PURPLE
 90  SETA 0,4,0,4,4,ATR
100  SETATR 0,BLACK,GREEN
110  SETA 0,0,4,4,4,ATR
```

This sets up some attributes to be scrolled.

You can see that the attributes are scrolled right one character block by ATTR 0,0,0,8,8. Note that the attributes are always wrapped round. ATTL (left), ATTUP (up) and ATTDN (down) are similar.

SPRITE TRANSFORMATIONS

This set of commands are used to carry out transformations on sprites and are intended to be used prior to displaying sprites on the screen since they are not as fast as the move or scroll commands.

FLIP and FLIPA are provided to reflect a window around a horizontal line through its centre, and both have the following parameters:

```
SPN,COL,ROW,WID,HGT
```

FLIPA is used the reflect the attributes only; the secondary attributes are moved only if in ATT2ON mode. FLIP will move the pixel data, and attributes as well if the computer is in ATTON or ATT2ON mode.

Put some shapes on the screen:

```
POLY 0,16,16,16,16,0,0 : POLY 0,48,16,16,16,3,0
POLY 0,16,48,16,16,4,0 : POLY 0,48,48,16,16,5,0
```

If you now type:

```
FLIP 0,0,0,8,8
```

you will see that the window is turned upside-down.

MIR and MAR are equivalent to FLIP and FLIPA; they have the same parameters but they reflect the window around a vertical line rather than a horizontal one.

MIR 0,0,0,8,8 will reflect the figures at the top left of the screen.

SPIN is used to rotate one window through 90 degrees into another window - unlike FLIP and MIR, it cannot be used with multi-colour mode data. Its parameters are:

```
SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
```

WID and HGT are the width and height of the source window; obviously, these are interchanged to give the dimensions of the destination window in SPN2.

If you now type:

```
SPIN 0,0,0,8,8,0,8,8
```

the data you put on the screen earlier will be rotated and placed on a different part of the screen.

XPANDX and XPANDY expand one sprite window into another in the X (horizontal) and Y (vertical) directions respectively. They both use the following parameters:

SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2

WID and HGT give the size of the source window; in the case of XPANDX, the destination window will be twice as wide as WID, and with XPANDY, the destination will be twice as high as HGT.

Since expansion of the window begins at the right in the case of XPANDX and at the bottom of the window for XPANDY, it is possible to expand a window into itself. Try:

XPANDX 0,0,0,8,8,0,0,0

followed by

XPANDY 0,0,0,16,8,0,0,0

CHARACTER MANIPULATION

LCASE and UCASE put the text into lower case and upper case respectively. Since BASIC LIGHTNING holds the character set in RAM, the new character set has to be copied down from the character ROM.

The CHAR command moves a character into a sprite at a specified row and column. The parameters are:

SPN,COL,ROW,NUM

NUM is the number of the character. As you may already know, the C64 uses display codes when displaying data on the screen which differ from the ASCII codes. The display codes refer directly to the position in the character set. NUM is the number of the character being used:

0 to 255	ordinary ASCII characters
256 to 511	reverse ASCII characters
512 to 767	display codes
1024 to 1279	double width ASCII
1280 to 1535	reverse double width ASCII
1536 to 1791	double width display codes

If NUM is less than 256, it is assumed to be an ASCII character, and is converted into display codes 0 to 127 which are normally the non-reversed characters. If NUM is in the range 256 to 511, the display codes from 128 to 255 are used instead, which are normally reverse characters. When NUM is between 512 and 767, 512 is subtracted to give the display code.

It is also possible to put double-width characters into a sprite, using NUM>1024.

For example, an "a" can be placed on the screen using either CHAR 0,0,0,65 (65 is the ASCII code for "a") or CHAR 0,0,0,513 (513 - 512 = 1 is the display code for "a").

The STRPLOT command can be used to place a whole string on the screen; the parameters are:

SPN,COL,ROW,<string>,<offset>

The offset is simply added to the ASCII value of each character in the string before placing it in the sprite. Thus an offset of zero gives ordinary characters, 256 gives reverse characters, 1024 gives double width and 1280 gives reverse double-width. Also, an offset of 2048 will put single-width characters in the sprite, but will leave a character block between each one; 2304 will give double-spaced reverse characters. Try the following:

```
STRPLOT 0,10,10,"Basic Lightning",0
STRPLOT 0,10,10,"Basic Lightning",256
STRPLOT 0,10,10,"Basic Lightning",1024
STRPLOT 0,10,10,"Basic Lightning",1280
```

PUTCHR is the opposite of the CHAR command - it moves a character block from a sprite back into the character memory. It has the same parameters as CHAR:

```
SPN,COL,ROW,NUM
```

NUM can take on a value between 0 and 767, 0 to 255 converting NUM from ASCII, 256 to 511 giving reverse ASCII, and 512 to 767 being converted straight into the display code, as with CHAR. Note that if you want to use PUTCHR to redefine the letter "a" for example, you must also redefine the reverse "a" (display code 129) if it is to be used with the flashing cursor.

Finally, DBLANK will blank the entire display to border colour while DSHOW will turn on the display again.

READING THE KEYBOARD, JOYSTICK AND LIGHTPEN

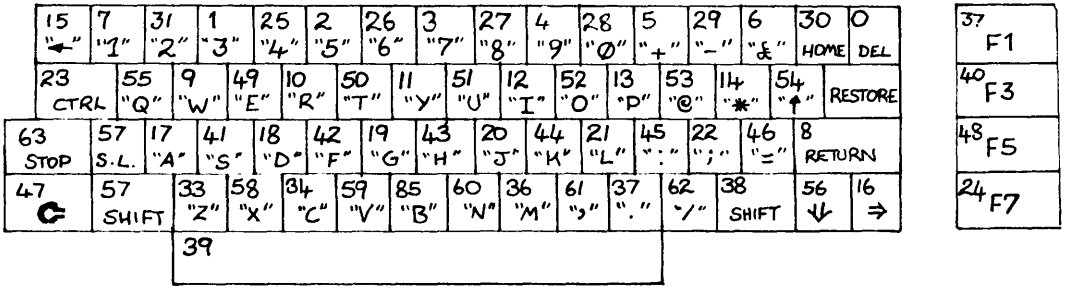
KEYBOARD

It is of course possible to read the keyboard using the existing GET command, but this cannot detect multiple key depressions and cannot detect the shift or Commodore keys.

KB(n) is a function which returns a true value (-1) if key no. n is pressed. The keys are numbered from 0 to 63:

0	INST/DEL	32	f1
1	"3"	33	"Z"
2	"5"	34	"C"
3	"7"	35	"B"
4	"9"	36	"M"
5	"+"	37	"."
6	" "	38	R.H.SHIFT
7	"1"	39	SPACE BAR
8	RETURN	40	F3
9	"W"	41	"S"
10	"R"	42	"F"
11	"Y"	43	"H"
12	"I"	44	"K"
13	"P"	45	":"
14	"*"	46	"="
15	"←" top l.h. of K.B.	47	'COMMODORE' KEY
16	⇐⇒	48	f5
17	"A"	49	"E"
18	"D"	50	"T"
19	"G"	51	"U"
20	"J"	52	"O"
21	"L"	53	"@"
22	";"	54	"↑"

23	CTRL	55	"Q"
24	f7	56	↑ ↓
25	"4"	57	SHIFT LOCK and L.H. shift
26	"6"	58	"X"
27	"8"	59	"V"
28	"0"	60	"N"
29	"-"	61	","
30	CLR/HOME	62	"/"
31	"2"	63	RUN/STOP



JOYSTICKS

FIRE1 and FIRE2 are functions which will give a true value if the fire button on the joystick in control ports 1 or 2 is pressed. For example:

```
.
.
.
1230 IF FIRE1 THEN PROC zap alien
.
.
.
```

JS1 and JS2 are functions which return the directions of joysticks 1 and 2 respectively; the direction is represented as a number from 0 to 8:

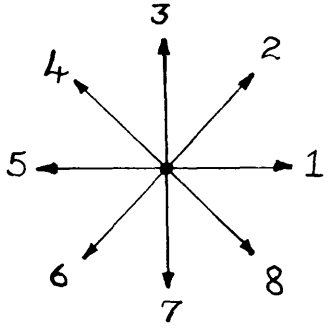


Fig 3

When a value of zero is returned by JS1 or JS2, this indicates that the joystick is in its unused position.

LIGHTPEN

LPX and LPY will return the X and Y positions of the lightpen respectively.

Besides having its own software sprites, BASIC LIGHTNING has commands which allow the use of the Commodore's hardware sprites.

Unlike software sprites, hardware sprites are not controlled using PUT and GET commands; each sprite can appear at only one location on the screen and is separate from the pixel data. In addition, each hardware sprite carries its own colour (independant of the attributes) and display mode (2 colour or 4 colour), and can be displayed with either hi-res or text data.

Defining a Hardware Sprite

Each hardware sprite is 24 pixels long and 21 pixels high, occupying 63 bytes in memory. Due to memory constraints, the sprite definitions must share memory with the character set, each sprite using the same amount of space as eight characters (one byte is left unused at the end of each definition to make 64 bytes). Since the character set takes up 2k bytes (from $C000 to $C7FF), there is room for 2048/64 = 32 definitions, which are numbered from 0 to 31, although only 8 can be put on the screen at once. Sprite definition no. n will share memory with the characters whose display codes are n*8 to n*8+7.

The data for a hardware sprite is designed with the Sprite Generator Program and saved as a software sprite in the normal way. At run-time, the pixel data can be copied from any part of a software sprite into a hardware sprite using the SPRCONV command.

The SPRCONV command has the following parameters:

```
SPN,COL,ROW,SPN2
```

SPN, COL and ROW define the top left-hand corner of the source software sprite in the normal way, COL and ROW being in pixels. SPN2 is the hardware sprite definition number. As an example type in the following:

```
STRPLOT 0,0,0,"*OA",256
STRPLOT 0,0,1,"SIS",256
STRPLOT 0,0,2,"   ",256
SPRCONV 0,0,0,16
```

Since the information was put into hardware sprite definition no. 16, the eight characters with display codes from 8*16 = 128 onwards will have been overwritten. To verify this, type CTRL-9 to give reverse characters, followed by "@abcdefg".

Only eight hardware sprites can exist on the screen at one time, each of which can be associated with one of the 32 definitions using the .SET command:

```
.SET <sprite no.>,<definition no.>
```

The sprites are numbered from 0 to 7. If you now type ".SET 1,16", hardware sprite no. 1 will be associated with the definition which you have just created.

It is of course possible to copy a large software sprite into several hardware sprites.

Switching on a Hardware Sprite

Before a sprite can be displayed, it must be turned on using the .ON command:

```
.ON <sprite#>
```

.ON 1 will enable sprite no. 1. The equivalent command to turn a sprite off again is:

```
.OFF <sprite#>
```

To define a sprite's colour, use:

```
.COL <sprite#>,<colour>
```

.COL 1,BLACK will make sprite no. 1 black.

Placing a Sprite on the Screen

Once the sprite has been enabled and given a colour, it will still not be visible because it is positioned off the screen. Positioning of a sprite on the screen is carried out by the .XPOS <sprite#>,<position> and .YPOS <sprite#>,<position> commands, the positions in both cases being at pixel resolution. Sprite no. 1 can be placed at the top left of the screen using:

```
.XPOS 1,24
.YPOS 1,50
```

Values of the x co-ordinate between 1 and 23 allow the sprite to be positioned partially off the screen; similarly for y co-ordinates between 30 and 49.

Moving a hardware sprite around the screen is very easy:

```
.YPOS1,100:FORI=100TO250STEP2:.XPOS1,I:NEXT
```

Double-Sized Sprites

It is possible to expand a hardware sprite to double size in either direction using .XPANDX <sprite#> to expand in the X-direction and .XPANDY <sprite#> to expand in the Y-direction. If you type .XPANDX 1 followed by .XPANDY 1, the sprite on the screen will be expanded to double size.

.SHRINKX <sprite#> and .SHRINKY <sprite#> have the opposite effect, returning a sprite to normal size.

A double-sized sprite is partially displayed on the screen with y-co-ordinates between 9 and 14, or x-co-ordinates from 481 to 503, then 0 to 24.

Multi-Coloured Sprites

A hardware sprite is put into multi-colour mode using .4COL <sprite#>. As with the hi-res screen, horizontal resolution is cut in half. Two more colours are required; these are the same for all sprites and are set by the .COL0 and .COL1 commands:

```
.COL0 <colour>
.COL1 <colour>
```

The four possible colours are displayed differently by a hardware sprite than by the hi-res screen:

Hi-Res Screen (Software Sprite)	Hardware Sprite
Background colour	transparent (screen colour)
Colour 1	colour zero (set by .COL0 command)
Colour 2	sprite colour (set by .COLn command)
Colour 3	colour 1 (set by .COL1 command)

A sprite can be put back into 2 colour mode using .2COL <sprite#>.

Display Priorities

Lower numbered sprites have priority over high numbered sprites, e.g. sprite 0 will always appear to pass in front of sprite 1 if they coincide. It is also possible to control the priorities between sprites and background data (i.e. the hi-res pixel data created by the software sprites). To give the background priority over a sprite, use .OVER <sprite#>. .UNDER <sprite#> puts the background underneath a sprite again, as normal.

Hardware Sprites Collision Detection

Collisions between two sprites or between a sprite and background data is detected using the function .HIT(n). If n is less than 8, it will return a true value (-1) if sprite n has hit another sprite. If n is greater than 8, a true value is returned if sprite n-8 has hit background data. When you use .HIT(n) with a value of less than 16, the records of any other sprite-to-sprite or background-to-sprite collisions are cleared. However, you can still detect these by adding 16 to n, in which case the value of the sprite collision register the last time that n was less than 16 is used.

In multi-colour mode, colour zero (set by .COL0) and background colour 1 are considered to be transparent for collisions.

SMOOTH SCROLLING

Smooth scrolling allows you to shift the entire screen over by 1 to 7 pixels horizontally or vertically. Once the screen has been shifted over by seven pixels, a wrap or scroll command must be used to move it by one character.

Before using smooth scrolling, the screen must be shrunk to 38 columns by 24 rows to give space for new data to be shifted in; this is achieved using the H38COL command (to go back into normal display mode, use H40COL). This gives two columns on either side of the screen which are hidden under the border, and one at the bottom of the screen.

Using the SCRLX n and SCRLY n commands, where n is between 0 and 7, it is possible to shift the entire screen by n pixels.

4. SOUND COMMANDS

BASIC LIGHTNING provides a set of sound commands which allow you to control the 64's SID chip.

To generate a sound from one of the three voices in the SID chip, you need to set up the following:

1. Master volume
2. Frequency
3. Envelope (ADSR)
4. Waveform

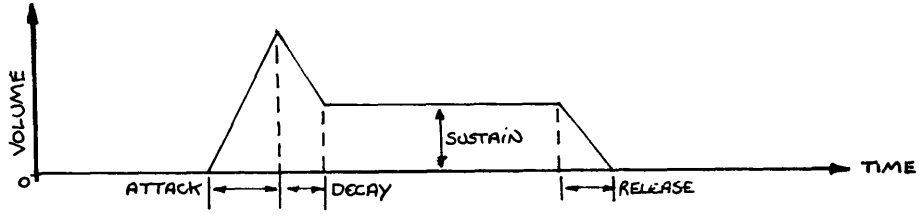
The master volume of the SID chip is set using the VOLUME command - it is followed by an expression in the range 0 to 15. For example, "VOLUME 15" sets the volume to its maximum value.

The frequency of a voice is set by the FRQ command which takes the following form:

```
FRQ voice,frequency
```

The voice is either 1, 2 or 3. The frequency is not in Hz (cycles per second); you must multiply the frequency in Hz by 16.4015 first. Thus, to set the frequency of voice 1 to A (440 Hz) you would use FRQ 1,440*16.4015. The maximum value of the frequency parameter is 65535.

The volume of a musical note changes from when it is first struck. This can be split into four phases: attack, decay, sustain and release:



After being struck, the volume rises to its peak value at a rate determined by the 'attack'. It then falls to the 'sustain' level at a rate determined by the 'decay'. At the end of the note, the volume falls away to zero at the 'release' rate. This 'envelope' shape can be set up using the ADSR command: ADSR voice, attack rate, decay rate, sustain level, release rate. The voice is 1, 2 or 3, and the rest of the parameters are in the range 0 to 15.

The time between the start of the attack and the start of the release is determined by the MUSIC command which has two parameters; the voice number and the length of the note. The length can be from 1 to 255 and is measured in 60ths of a second. A value of 0 indicates that the note lasts indefinitely.

As an example, type in the following and run it:

```
10 SIDCLR
20 VOLUME 15
30 FRQ 1,7217
40 ADSR 1,5,8,5,9
50 TRI 1
60 MUSIC 1,20
```

The SIDCLR command in line 10 simply clears any values that were set up in the sound chip. TRI in line 50 sets up the waveform type and is explained in the next section.

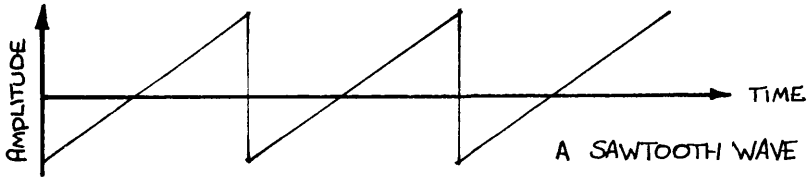
Try experimenting with different values of frequency and other parameters for ADSR.

Changing the Waveform

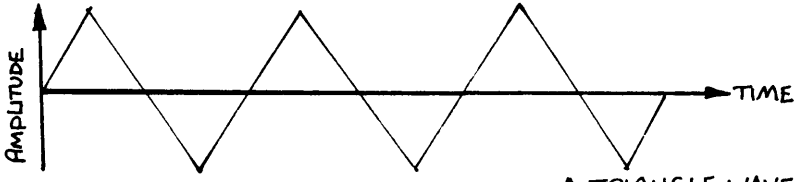
The 64's sound generator is capable of generating four types of waveform:

1. SAWTOOTH waves
2. TRIANGLE waves
3. PULSE waves
4. NOISE

'SAWTOOTH' and 'TRIANGLE' refer to the shape of the waveform when plotted on a graph:

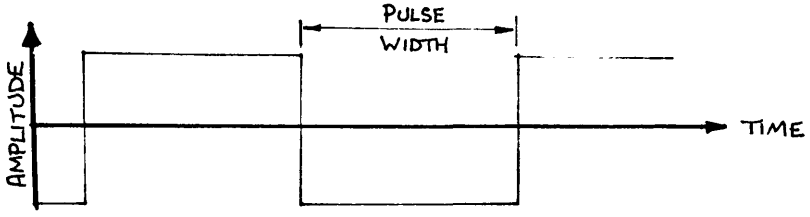


A SAWTOOTH WAVE



A TRIANGLE WAVE

A 'PULSE' wave looks like this:



The pulse width can be varied, so that a variety of sounds can be created.

NOISE can be used to generate realistic explosions.

The SAW command selects the sawtooth waveform:

SAW voice

TRI selects triangle waves:

```
TRI voice
```

NOISE selects noise:

```
NOISE voice
```

PULSE selects a pulse wave:

```
PULSE voice, pulse width
```

The pulse width is in the range 0 to 4095; 2048 gives a square wave.

Try changing line 50 of the last example program to use different waveforms, and experiment with different values of pulse width when using the PULSE waveform.

Here are some examples of the sounds that can be created:

GUNFIRE:

```
VOLUME15:FRQ1,7217:ADSR1,1,9,3,9:NOISE1:MUSIC1,20
```

EXPLOSION:

```
VOLUME15:FRQ1,3000:ADSR1,0,13,5,12:NOISE1:MUSIC1,20
```

DEPARTING UFO:

```
10 SIDCLR
20 VOLUME 15
30 ADSR 1,9,2,11,12
40 TRI 1
50 MUSIC 1,30
60 FOR j=1 TO 30
70   FOR i=12000 TO 28000 step 800
80     FRQ 1,i
90   NEXT i
100 NEXT j
```

Any of these examples could, of course, use voice 2 or 3.

FILTERING

The timbre of sound produced can be altered using filtering. Using the filter command it is possible to control whether the output from each oscillator is passed through the filter or not. The format for this command is:

```
FILTER voice,flag
```

The flag is either TRUE or FALSE, TRUE indicating that the voice is to be filtered and FALSE indicating that it is not.

PASS n selects the filter's mode of operation:

```
PASS 0 low pass
PASS 1 high pass
PASS 2 band pass
PASS 3 notch reject
```

In low pass mode, frequencies above the cut-off frequency are attenuated. In high pass mode, frequencies below the cut-off frequency are attenuated. In band pass mode, only a narrow band around the cut-off is passed while notch reject has the opposite effect.

CUTOFF n is used to select the cut-off frequency. The frequency is in the range 0 to 2047; i.e. the frequency used by FRQ must first be divided by 32 before being used with the CUTOFF command.

It is also possible to make the filter resonant around the cut-off frequency using RESONANCE n; the parameter is in the range 0 to 15.

Using a low pass filter in conjunction with resonance, the 'EXPLOSION' example given earlier can be made more realistic:

```
10 VOLUME 15
20 FRQ 1,3000
30 ADSR 1,0,13,5,12
40 NOISE 1
50 FILTER 1,TRUE
60 PASS 0
70 CUTOFF 120
80 RESONANCE 12
90 MUSIC 1,20
```

RING MODULATION AND SYNCHRONISATION

Using ring modulation of two voices, very complex waveforms can be produced. Ring modulation is enabled using the RING command:

```
RING voice,flag
```

The flag is TRUE or FALSE, enabling or disabling ring modulation respectively.

If voice=1, voice 1's output is replaced by voice 1 ring modulated with voice 3. When voice=2, voice 2 is ring modulated with voice 1. Voice 3 is ring modulated with voice 2 when voice=3.

Realistic bell effects can be generated by using ring modulation coupled with low pass filtering:

```
 10 SIDCLR
 20 VOLUME 15
 30 ADSR 1,1,9,8,12
 40 TRI 3
 50 TRI 1
 60 RING 1,TRUE
 70 FILTER 1,TRUE
 80 PASS 0
 90 CUTOFF 80
100 REPEAT
110   READ a
120   IF a=-1 THEN EXIT
130   PROCbell(a)
140   READ a
150   FOR b=1 to a
160   NEXT b
170 UNTIL FALSE
180 END
190 '
```

(Continued over page)

(Continued from previous page)

```
200 LABELbell(fr)
210 FRQ 1,fr
220 FRQ 3,fr*1.9766
230 MUSIC 1,40
240 PROCEND
250 '
260 DATA 1514,700,1201,700,1340,700,900,1400
270 DATA 900,700,1348,700,1514,700,1201,1400
280 DATA 1201,1500,1201,1500,1201,1500,1201,1500
290 DATA 1201,1500,1201,1500,1201,1500,1201,1500
300 DATA 1201,1500,1201,1500,1201,1500,1201,1500
310 DATA -1
```

Synchronisation of two voices is enabled using the SYNC command in a similar way to RING: SYNC voice,flag.

Synchronisation of two voices can be used to mimic the sound of engines. In the following example we hear what happens when a hedgehog is run over ...

```
 10 SIDCLR
 20 '
 30 'The Bikers ........
 40 VOLUME 15
 50 ADSR 1,15,5,15,15
 60 TRI 1
 70 TRI 3
 80 SYNC 1,TRUE
 90 REPEAT
100   READ j
110   IF j=-1 THEN EXIT
120   READ k,m
130   MUSIC 1,100
140   FOR i=j TO k STEP m
150     FRQ 3,i
160     FRQ 1,i*2.5
170   NEXT i
180 UNTIL FALSE
190 DATA 270,750,1.4,650,1000,1.4
200 DATA 750,1150,1.4,1000,1400,1
210 DATA -1
220 '
230 'meet Spiny Norman ......
240 SYNC 1,FALSE
250 RING 1,TRUE
260 ADSR 1,3,5,8,3
270 MUSIC 1,60
280 FOR i=0 to 5000 STEP 100
290   FRQ 1,20000+i
300   FRQ 3,20250+i
310 NEXT i
320 '
330 '
340 RING 1,FALSE
350 FRQ 2,3000
360 ADSR 2,0,13,5,12
370 NOISE 2
380 FILTER 2,TRUE
390 PASS 0
```

(Continued over page)

(Continued from previous page)

```
400 CUTOFF 120
410 RESONANCE 15
420 MUSIC 2,20
430 FOR i=0 to 300
440 NEXT i
450 '
460 '
470 x=10000
480 y=8409
490 PULSE 1,1000
500 ADSR 1,15,15,15,15
510 MUSIC 1,255
520 FOR j=0 TO 32
530   FRQ 1,x
540   FOR i=0 TO 200
550   NEXT i
560   FRQ 1,y
570   FOR i=0 TO 200
580   NEXT i
590   IF j=7 THEN x=9803:y=8244
600 NEXT j
```

MUTE, OSC and ENV

OSC and ENV are functions which return the output amplitude of voice 3's oscillator and envelope generator. It is possible to obtain a vibrato effect by using OSC to modify either voice 1 or voice 2's frequency.

If oscillator 3 is being used in this way its output must be disabled using the MUTE command. MUTE TRUE disables oscillator 3's output and MUTE FALSE enables it again.

5. MULTI-TASKING

In BASIC LIGHTNING it is possible to execute up to five parts of a program concurrently. This is done using a technique known as "time-slicing" - one part of a program is executed for 1/20th of a second before moving on to the next. Each "task" has its own set of variables which are independant of the others.

Before executing concurrent tasks, space must be allocated for the variables using the ALLOCATE command; this is followed by four numbers, separated by commas, which are the number of bytes reserved for variables in each of the four background tasks. This command should always go at the top of a program since it also performs a CLR to remove all variables.

Once the memory used by a task has been reserved, it can be invoked using the TASK command. The syntax for TASK is:

TASK <task number>,<line number or label>

The task number can be 1 to 4. This command can only be used from the foreground task.

Finally, the HALT command can be used to stop execution of a task. HALT takes the form HALT n where n is the task number (1..4) that is to be terminated.

Here is a simple example to illustrate the use of multi-tasking:

```
 10 ALLOCATE 100,0,0,0
 20 WINDOW 1
 30 SCLR 0,1
 40 STRPLOT 0,0,0,"BASIC LIGHTNING.....................",0 '25 dots
 50 TASK 1,blscr
 60 REPEAT
 70   INPUT a$
 80   IF a$="quit" THEN STOP
 90   PRINT a$
100 UNTIL FALSE
110 '
120 LABEL blscr
130 REPEAT
140   RPT 64, WRR1 0,0,0,40,1
150 UNTIL FALSE
```

If you RUN it, you will see that the computer is inputting and printing strings and simultaneously scrolling the top line of the display.

Error Messages and I/O

If an error, STOP or END occurs in one of the background tasks, the computer will wait until all the other tasks and the foreground task have finished executing the current command before leaving the program. In the case of an error, the task number is printed in angle brackets above the error message.

I/O commands such as INPUT, PRINT etc. can only be executed by one task at a time - a task will wait for the others to finish using I/O before proceeding with using an I/O command.

Passing Values between Tasks

Since each task has its own set of variables, special commands are provided to pass values between the tasks. The simplest way to do this is by using the pseudo-array COMMON% which is an array of 64 integers, the values of which are shared between the various tasks.

If you have to pass real numbers or strings, you can use IMPORT which is a function: IMPORT (<task number>,<expression>). The task number can be 0 to 4 - zero means the foreground task. The expression is evaluated by that task. Note that IMPORT cannot be used with an I/O command, i.e. you cannot use PRINT IMPORT (0,A$), and it cannot be used from direct mode. Also, IMPORT will wait for the task to finish executing its current command before evaluating the expression.

Allocation of Scrolling Buffers

When using the WRAP, ATTUP or ATTDN commands, a 256-byte buffer is used. In the foreground task or task 1, this is hidden from the user. However, in tasks 2, 3 or 4, parts of the text screen are used to hold temporary scrolling data, so the text screen must not be scrolled if WRAP, ATTUP or ATTDN are operating in tasks 2, 3 or 4.

6. THE SPRITE GENERATOR PROGRAM

INTRODUCTION

The Sprite Generator Program was developed to compliment the Lightning series of languages. The languages are comprised of commands for manipulating sprites and screen data but do not have the facility to directly design sprites. This means there are two phases to games creation. The first involves designing and editing your sprites with the sprite generator program, and the second involves the writing of the game itself using the Lightning languages. In practice the two areas of work will probably be carried out simultaneously. For those of us who are not artistically inclined, there are two sets of previously defined sprites ready to use.

The Arcade Sprite Set

The first set of sprites (stored directly after the Sprite Generator Program on the tape version) are the arcade sprites. A table is given at the end of this section. These can be used in your games without any copyright problems whatsoever. The filename is "DEMO1".

The Demonstration Sprites

These are stored directly after the arcade sprite set and are the sprites used in the demo program. Again these can be used, edited etc., with no copyright restrictions, and they are stored under the filename "DEMO2".

COLD START

If you enter the sprite generator program via a COLD start, then all sprites previously stored will be cleared and all system variables reset. If, for instance, you wish to use the demonstration sprites, you would enter via a COLD start.

WARM START

If you enter the program via a WARM start then all sprites will be conserved and all system variables left unchanged. If you accidentally break out of the program, re-enter via a WARM start.

THE CHR$ SQR

CHR$ SQR is the abbreviation used throughout this text for character square, and refers to the 8 by 8 (or 8 by 4) grid to the left of the sprite screen. This is the area used to create and edit sprites one character at a time.

THE HI-RES SCREEN

This is the area of screen 15 characters by 30 characters on which sprites are created, developed, transformed and generally worked on.

THE CHR$ SQR CURSOR

This is the non-destructive flashing cursor which is used to design and edit the character currently held in the CHR$ SQR.

HI-RES WINDOW

The area of the screen currently being worked on is referred to as the screen, or hi-res, window. Its position is defined by COL and ROW, which correspond to the positions of the top left of the window, and its dimensions are defined by HGT and LEN. Top left of the hi-res screen has co-ords COL:10, ROW:0. To see the window you are currently working on just press the SPACEBAR. The window will flash.

SPRITE LIBRARY

This refers to the set of sprites you are currently working with and can contain up to 255 sprites or use 8192 bytes.

SOFTWARE SPRITES

A sprite is a programmable graphics character. The Sprite Generator Program can develop up to 255 sprites of user selectable dimensions, up to a total of 8k. All sprite commands also apply to the screen which can be treated as a fixed dimension sprite and is given the number 0.

HARDWARE SPRITES

The Commodore 64 has its own extremely powerful sprites which for clarity we will refer to as the hardware sprites. These are 24 pixels wide (or 12 wide in 4 colour mode) and 21 pixels high. Your software sprites can be converted to hardware sprites in FORTH or BASIC but they will need to have the above dimensions or less. This means a sprite should be 3 characters wide and 3 characters high but with at least three free pixel rows above or below the character.

INKS and PAPER

When working in two colour mode the Commodore 64 allows two colours per character. The background colour (displayed by pixels "off") is the colour indicated by PAPER. The foreground colour (displayed by pixels "on") is the colour indicated by INK 3. When working in four colour mode the Commodore 64 allows four colours per character. The background colour is again the colour indicated by PAPER. The three foregound colours are the colours indicated by INK 1, INK 2 and INK 3. It is advisable to be careful not to set any of these three INKs or the PAPER to hold the same value. It is also advisable not to change the PAPER colour when in four colour mode.

The Commodore Key

This is the key marked with the Commodore decal and is located on the far left of the bottom key row.

THE FUNCTION KEYS

The Sprite Generator Program operates in five distinct modes, although some functions are available in more than one of the five modes. To select a particular mode press the Commodore key together with one of the keys "1", "2", "3", "4" or "5". Release the Commodore key when the mode is displayed. To exit any of the five modes merely press the Commodore key and 0 until MODE:0 is displayed.

MODE: 1

This mode deals with the 2 colour mode character development. Once entered, MODE:1 will be displayed and the CHR$ square cursor will flash.

The Cursor Keys

The normal Commodore cursor keys are used to move the non-destructive cursor around the 64 individual cells. Use RIGHT SHIFT and not LEFT SHIFT to move left and up. To switch a cell "ON" press "3". The appropriate cell will switch on (turn black). To switch a cell "OFF" press "4". The appropriate cell will switch off (turn white). So the cursor keys and the keys "3" and "4" can be used to build up sprites, by designing a character (64 pixels) at a time.

The "D" and "U" Keys

These keys will <D>ownload or <U>pload between the CHR$ square and the hi-res screen. Once you have designed your character using the cursor keys or Numerical Data Input ("N" key), typing "D" will write the contents of the CHR$ square to the current cursor position on the hi-res screen. The current cursor position is displayed in the panel as COL and ROW, but pressing the SPACEBAR will flash the current hi-res window. When "D" is pressed, data will be downloaded into the top left of this window. If ATTR (displayed in the panel and toggled by pressing "A") is zero then the current INK 3 and PAPER colours will not be downloaded with the pixel data. If ATTR is 1 then these attributes will be set in the downloaded character. Similarly, pressing "U" will upload from the hi-res screen to the CHR$ square and if ATTR is 1 then INK 3 and PAPER will take on the values of the uploaded character from the hi-res screen. If ATTR is zero then attributes INK 3 and PAPER remain unchanged. Note that pressing "U" will destroy whatever is currently held in the CHR$ square and if the character square being picked up (top left of screen window) is empty then this will have the effect of clearing the CHR$ square.

The "A" Key

Pressing the "A" key will toggle the attribute flag on and off. In other words if ATTR is 0 then it becomes 1, and if 1, then it becomes 0. In fact, if you hold "A" down you will see the value of ATTR switching between 0 and 1. The value of ATTR affects the operation of many of these functions but it is generally true to say that if ATTR is 1 then attributes will be moved with pixel data and when ATTR is 0 pixel data alone will be moved.

The "E" Key

This key is used to change the current values of the INKs and PAPER. Once "E" has been pressed the CHR$ square will cease flashing. On releasing the "E" key the prompt "INK TO BE EDITED" will be displayed. You should then type a number in the range 1 to 4 followed by ENTER. "1", "2" and "3" correspond to INK 1, INK 2 and INK 3 and "4" corresponds to PAPER. In mode 1 there is little point in editing INK 1 or INK 2 as these are only utilised when four-colour mode is in operation, i.e. MODE:2. If the number entered is not in the range 1 to 4 the prompt will simply repeat. Once the INK (or PAPER) to be changed has been selected, a second prompt "NEW VALUE" is displayed. This should be a value in the range 0 to 15. The number input can be in decimal or hex preceded by "$", i.e. typing $E is

equivalent to typing 14. Again, an out of range value will cause the prompt to repeat. The colours and their corresponding values are as follows:

0	BLACK	8	ORANGE
1	WHITE	9	BROWN
2	RED	10	LIGHT RED
3	CYAN	11	GRAY1
4	PURPLE	12	GRAY2
5	GREEN	13	LIGHT GREEN
6	BLUE	14	LIGHT BLUE
7	YELLOW	15	GRAY3

The "N" Key

The "N" key is used to enter a character as a series of 8 decimal or hexadecimal numbers. As before, hex numbers should be preceded by a "$". Pressing "N" will cause the CHR$ cursor to stop flashing and releasing "N" will display the prompt "BYTE 0 : N" where BYTE 0 means the first of the 8 bytes making up the character in the CHR$ square and N is the value it currently holds. If the character is blank then N will be zero, if not then the value will be the binary equivalent of the pattern in that byte. If you do not wish to change this value then just press ENTER, otherwise enter the new value. In either case, pressing ENTER will advance to BYTE : 1 and so on through the 8 bytes. You will, of course, need to be familiar with binary arithmetic to make use of this facility.

The "C" Key

Pressing "C" simply clears the CHR$ square.

The SPACEBAR Key

Pressing the spacebar will cause the hi-res screen window to flash. Pressing the SPACEBAR in conjunction with the Commodore cursor keys will move the hi-res screen window around the hi-res screen.

The COMMODORE Key

Pressing the Commodore key will exit MODE:1 and return to MODE:0 ready to enter any of the four modes.

The LEFT SHIFT Key

Using the LEFT SHIFT key in conjunction with the Commodore cursor keys, the size of the screen window can be altered.

LEFT SHIFT AND RIGHT ARROW	EXTEND ONE CHARACTER IN WIDTH
LEFT SHIFT AND LEFT ARROW	CONTRACT ONE CHARACTER IN WIDTH
LEFT SHIFT AND DOWN ARROW	EXTEND ONE CHARACTER IN HEIGHT
LEFT SHIFT AND UP ARROW	CONTRACT ONE CHARACTER IN HEIGHT

The panel is updated at the completion of the change.

The "-" Key

Pressing "-" will set the current INKs and PAPER colours throughout the hi-res screen window.

The "+" Key

Pressing "+" will set the current INKs and PAPER colours in the panel display to those of the top left character position of the hi-res window.

MODE: 2

This mode deals with multicolour (4 colours per character) development and has essentially the same commands as MODE:1 except that numerical data entry is not available.

The Cursor Keys

Again, the Commodore cursor keys are used to move a non-destructive cursor around the CHR$ square which this time has only 32 cells since the resolution in multicolour mode is only half that of the hi-res mode. To set the current cursor position to INK 1, INK 2, INK 3 or PAPER colours press 1, 2, 3 or 4 respectively.

The "D" and "U" Keys

These keys will <D>ownload or <U>pload CHR$ data to or from the hi-res screen but this time 4 colours are involved so the attribute flag ATTR is somewhat more significant. If the attribute flag (ATTR) is on (=1) then pressing "D" will cause the hi-res character to take on the current INK and PAPER colours, if ATTR is off (0) then attributes of the hi-res screen window will be used. Initially these will need to be set, so characters are normally downloaded with ATTR "ON" to begin with. In fact, it is only rarely that ATTR will be set to 0 (off) during mode 2 operations. Pressing "U" with ATTR on (=1) will lift the character from the hi-res screen together with its INK and PAPER colours and reset the current values in the panel if these were different from those of the character being uploaded. Pressing "U" with ATTR off (=0) will lift the character but display it in the character cell with the current INK and PAPER colours.

The "A" Key

This is used to toggle the attribute switch, ATTR, in exactly the same way as described in the previous section covering MODE:1 operation.

The "E" Key

This operates in the same way as the INK and PAPER editor in MODE:1. This time, however, changes made to the INKS or PAPER are reflected in the CHR$ square. Suppose INK 3 were RED and the editor was used to change INK 3 from RED to BLUE. As soon as the new value were entered all the RED cells would change to BLUE and the panel would be updated.

The "C" Key

This has the same function as the "C" key under MODE:1 and merely clears the CHR$ square.

The SPACEBAR Key

This flashes the screen window and in conjunction with the Commodore cursor keys allows its position to be changed in the hi-res screen.

The COMMODORE Key

Again, its operation is the same as that for MODE:1 and causes an exit to MODE:0.

The LEFT SHIFT Key

The dimensions of the hi-res window can be updated using the LEFT SHIFT key in conjunction with the cursor keys in the manner described in MODE:1.

The "-" Key

Pressing "-" will set the current INKs and PAPER colours throughout the hi-res screen window.

The "+" Key

Pressing "+" will set the current INKs and PAPER colours in the panel display to those of the top left character position of the hi-res window.

MODE: 3

This mode is essentially concerned with operations on the screen window and works in hi-res or 4 colour mode in exactly the same way.

The "W" and "I" Keys

Pressing "W" and then "I" will invert (1's complement) the contents of the screen window. This means that pixels set "on" will be set "off" and vice versa. No further operation will take place until the "W" key is released.

The "W" and "F" Keys

Pressing "W" and then "F" will flip (vertically mirror) the contents of the screen window. If ATTR is "on" then attributes will also be flipped, if ATTR is "off" then pixel data only will be flipped. No further operation will take place until "W" is released.

The "W" and "M" Keys

Pressing "W" and then "M" will mirror (horizontally) the contents of the screen window. If ATTR is "on" then attributes will also be mirrored, if ATTR is "off" then pixel data only will be mirrored. No further operation will take place until the "W" key is released.

The "W" and "C" Keys

Pressing "W" and then "C" will clear the pixel data in the screen window. If ATTR is "on" then the attributes throughout the window will be set to the current INKs and PAPER, otherwise INK 3 will be set to BLACK and INKs 1 and 2 set to WHITE throughout the window.

The "S" Key

Pressing "S" in conjunction with the Commodore cursor keys will cause the screen window to scroll without wrap in the appropriate direction. Note that scrolling without wrap causes data to be lost at the edges of the window.

The "R" Key

Pressing "R" in conjunction with the Commodore cursor keys will cause the screen window to scroll with wrap in the appropriate direction. Note that data scrolled with wrap will not be lost at the edges of the window.

The HOME Key

Pressing the HOME key will cause the screen window to move to the top left of the hi-res screen (COL = 10, ROW = 0). Its dimensions (height and length) remain unaffected.

The RIGHT SHIFT and HOME Keys

Pressing RIGHT SHIFT and HOME together will cause the whole hi-res screen to be cleared and the screen window to move to the top left (COL = 10, ROW = 0). The window dimensions remain unchanged. If ATTR is 1 then the attributes throughout the hi-res screen will be set to those of the current panel dislay.

The "V" Key

Pressing the "V" key in conjunction with the Commodore up or down cursor keys will scroll with wrap the whole of the hi-res screen from ROW 0 to ROW 24. Note that this means that characters can be stored underneath the panel in the lower half of the screen. If ATTR is "on" then the attributes in the hi-res screen from ROW 0 to ROW 14 (NOT ROW 24 as in the pixel data) will also be scrolled with wrap. If ATTR is "off" attribute data is unaffected.

The "E" Key

The current INKS and PAPER can be edited in exactly the same way as they were in MODE:1 and MODE:2.

The "A" Key

The attribute flag ATTR can be toggled in exactly the same way as it was in MODE:1 and MODE:2.

The SPACEBAR Key

The screen window can be flashed and moved around the hi-res screen by pressing the SPACEBAR key and the cursor keys in the same manner as that described in the sections covering MODE:1 and MODE:2.

The COMMODORE Key

Again, its operation is the same as that for MODE:1 and is used to exit to MODE:0.

The LEFT SHIFT Key

The dimensions of the hi-res window can be updated using the LEFT SHIFT key in conjunction with the cursor keys in the manner described in MODE:1.

The "-" Key

Pressing "-" will set the current INKs and PAPER colours throughout the hi-res screen window.

The "+" Key

Pressing "+" will set the current INKs and PAPER colours in the panel display to those of the top left character position of the hi-res window.

MODE: 4

This mode can be thought of as dealing with data movement. Most of the operations, though not all, deal with movement between the screen and a sprite.

The "G" Key

This function is used to convert the screen window into a sprite. When "G" is pressed the screen window will cease flashing. Releasing "G" will produce the prompt "ENTER SPRITE NUMBER". If the number entered is not in the range 1 to 255 then "SPRITE PARAMETER OUT OF RANGE <CR>" will be printed. Note that <CR> means "PRESS RETURN TO CONTINUE". If the sprite number entered is the number of a sprite which has already been defined then "SPRITE ALREADY DEFINED PRESS <CR>" will be printed. If you wish to redefine the sprite then you will need to destroy it using the "W" key or use the more sophisticated "M" commands (see this section). The sprite created using the "G" key will have the dimensions and contents of the screen window. If ATTR is "on" it will have the atttributes of the screen window, if ATTR is "off" it will have the current INK and PAPER attributes displayed in the panel. After the sprite has been created the values of SPND (the end of sprite space), SPRITE (the start address of the sprite created) and FREE MEMORY, are updated on the panel display.

The "P" Key

This function is used to <P>ut a sprite stored in memory onto the hi-res screen at the current screen window position. The dimensions of the screen window are unaffected. When "P" is pressed the screen window will cease flashing and when it is released the prompt "ENTER SPRITE NUMBER" will appear. Again, if the sprite number entered is not in the range 1 to 255 the sprite parameter out of range

error is generated. If the sprite number entered is the number of a sprite which has not been created then the prompt "SPRITE NOT DEFINED PRESS <CR>" will be displayed. If ATTR is "on" then the sprite attributes will also be <P>ut to the screen, if ATTR is "off" then the sprite will have the attributes of the hi-res screen area that it occupies. Note that data on the hi-res screen will be overwritten by the sprite being <P>ut.

The "C" Key

This function is provided to enable the user to set up a sprite of user defined dimensions without actually filling it with any data. The user will be prompted to enter sprite number, sprite height and sprite length. Errors will be displayed if:

The sprite number is not in the range 1 to 255
The sprite number has already been used
The height is not in the range 1 to 255
The width is not in the range 1 to 255
There is insufficient memory available

The sprite generated will have all zeros in the pixel data and have the current INKs and PAPER attributes irrespective of ATTR. The panel display is updated.

The "W" Key

This function is provided to enable the user to delete a sprite from memory, reclaiming the bytes used and leaving its number free for reallocation. The only parameter entered is the sprite number and errors will be generated if the sprite number is out of range or the sprite does not already exist. The values of SPND and MEMORY FREE are updated in the panel display. Use this function with great care!

The "I" Key

This function updates the value of SPRITE in the panel display to hold the start address of the sprite whose number is entered.

The "R" Key

This function is used to <R>un an animated sequence of consecutive sprites with a programmable delay between frames. The user is first prompted to "ENTER SPRITE NUMBER". This should be the number of the first sprite in the series. The second prompt is "NUMBER IN SERIES", and finally "DELAY FACTOR" which is a positive number in the range 1 to 32767. In practice delays of more than 100 would hardly ever be used. Errors will be generated if:

The sprite series lies outside the range 1 to 255
Any of the sprites in the series is found not to exist.

The "A" Key

Toggles ATTR as in previous modes

The "M" Key

There are seven operations prefixed by "M". The parameters input are the same in each case. There are seven parameters and these are used to specify two windows. In each case data moves from the "source" window to the "target" window. The first prompt is "ENTER TARGET SPRITE NUMBER". This should be the number of the sprite to which the data is to be moved. The next two prompts are "ENTER TARGET COLUMN" and "ENTER TARGET ROW". The next three enter the source sprite number, source column and source row. These specify the sprite window which holds the data to be moved. Finally the dimensions of the window; width followed by height are entered. The final prompt is "ENTER OPERATION". The seven operations available are:

The "B" Operation

After the above parameters are entered the source window is block moved into the target window, replacing the contents of the target window. If ATTR is "on" the attributes will move with the pixel data.

The "A" Operation

Entering "A" will cause the data from the source window to be "ANDed" with the data in the target window and the result left in the target window. ATTR is used to control the flow of attributes.

The "O" Operation

Entering "O" will cause the data from the source window to be "ORed" with the data in the target window and the result left in the target window. ATTR is used to control the flow of attributes.

The "E" Operation

Entering "E" will cause the data from the source window to be "XORed" with the data in the target window and the result left in the target window. ATTR is used to control the flow of attributes.

The "S" Operation

This function will rotate the source window by 90 degrees clockwise and place it in the target sprite at the target column and row. The "SPRITE PARAMETER OUT OF RANGE" error message will be generated if the source or target windows are not fully contained within the source and target sprites respectively. ATTR controls the flow of attributes. This function may produce "garbage" in four colour mode.

The "X" Operation

This function will expand the source window by a factor of 2 in the horizontal direction and place it in the target sprite at its target column and row. The "SPRITE PARAMETER OUT OF RANGE" error will be generated if the source and target windows are not fully contained and attribute flow is controlled by ATTR.

The "Y" Operation

This function is identical to the "X" operation except that the expansion is in the vertical direction.

In all the above operations, the source window is unaffected by the operation, but care must be exercised to ensure that the two windows do not overlap or unpredictable results may occur.

THE SPACEBAR Key

The screen window can be flashed and moved around the hi-res screen by pressing the SPACEBAR key and the cursor keys in the same manner as that described in the sections covering MODE:1 and MODE:2.

The COMMODORE Key

Again, its operation is the same as that for MODE:1 and is used to exit to MODE:0.

The LEFT SHIFT Key

The dimensions of the hi-res window can be updated using the LEFT SHIFT key in conjunction with the cursor keys in the manner described in MODE:1.

The "-" Key

Pressing "-" will set the current INKs and PAPER colours throughout the hi-res screen window.

The "+" Key

Pressing "+" will set the current INKs and PAPER colours in the panel display to those of the top left character position of the hi-res window.

MODE: 5

This mode is concerned with <L>oading and <S>aving sprites to tape or disk.

The "S" Key

Pressing "S" will cause the CHR$ square cursor to stop flashing. Releasing "S" will produce the prompt "ENTER FILENAME". This is the filename under which the sprites will be <S>aved and re<L>oaded.

The "L" Key

This operates in the same way as the "S" key except that the current sprites will be lost and replaced by those sprites <L>oaded from tape or disk.

DISK COMMANDS

If you are using the disk version of the Sprite Generator then six additional commands are available in MODE:5.

The "E" Key

This causes the sprite file whose filename is entered to be <E>rased from the disk.

The "R" Key

Pressing "R" until the CHR$ cursor stops flashing and then releasing the keys produces the prompt "OLD NAME". You should respond by entering the name of the file to be renamed. A second prompt then follows; "NEW NAME". The name that the file it is to become is now entered.

The "C" Key

This operates in the same manner as the previous function, and provides a facility for copying files. The prompts "FROM" and "TO" should be responded to with the name of the file to be copied and the name of the new file to be copied to.

The "I" Key

Pressing "I" will initialise the disk after an error.

The "V" Key

Pressing "V" will validate the disk.

The "D" Key

This function will cause the directory to be listed to the INPUT prompt line. RETURN is used to advance to the next filename. On completion the CHR$ cursor will recommence flashing.

A SAMPLE SESSION WITH THE SPRITE GENERATOR

If you have not already loaded BASIC LIGHTNING then load this first. The Sprite Generator Program can then be loaded and RUN.

For the purposes of this sample session, the arcade sprites should be used, so respond to the first prompt "COLD OR WARM C/W" by pressing "C" to execute a COLD start. The demo sprites should now be loaded using the following proceedure:-

1. If you are using tape, the sprites are straight after the Sprite Generator, so this tape should be placed in the cassette player.

2. Type the Commodore key and "5" to selecte MODE:5. Hold these two keys down until MODE:0 changes to MODE:5 on the panel display. You are now in MODE:5.

3. Hold down "L" until the cursor in the CHR$ SQR stops flashing. Release "L" and the prompt "ENTER FILENAME" will appear. Enter DEMO1 and the arcade sprites will load.

4. Type the Commodore key and "0" to select MODE:0 and you are now ready to begin the session.

Let's start with the MODE:1 commands, so press the Commodore key and "1" until MODE:0 changes to MODE:1. The principal use of this mode is to design sprites a character at a time. The CHR$ SQR is used to do this.

The Cursor Keys

The usual Commodore cursor keys are used in conjunction with the RIGHT SHIFT key. The LEFT SHIFT key does not operate with the cursor keys. The cursor is displayed as a flashing square. Use the keys to move it around until you get a feel for it. Notice that the cursor "wraps around".

In order to set a particular pixel, move the cursor to the required position, release the cursor keys and simply press "3". Move the cursor to the next position you want to set and press "3" again. To unset a pixel, position the cursor and press "4". If neither of these keys ("3" or "4") is pressed, the cursor moves non-destructively. That is to say it moves around the screen without affecting the cells it moves across. If, however, you wish to draw lines of 3 or 4 pixels it is very annoying to keep releasing the cursor keys and pressing the "3" alternately. To draw a line, press "3", then the appropriate cursor keys. Once the cursor is moving "3" can be released and as long as the cursor keys are held down, a line will be drawn. Likewise, lines can be wiped out by holding down the cursor keys in conjuction with "4". Now spend a few minutes getting used to the cursor keys by creating the invader shown here as Fig. 1.

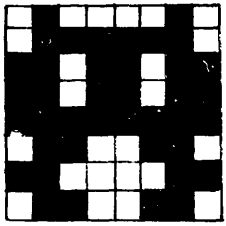


Fig 1

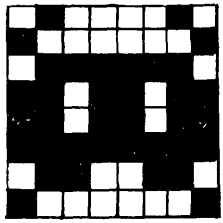


Fig 2

The "D" and "U" Keys

OK, now that you've designed a character it's time to see what it will look like, reduced to real size, on the hi-res screen. First check that ATTR is set to 0. If it isn't press "A" until ATTR is 0 on the panel display. Now press "D" to <D>ownload your invader. The invader should now appear at the top left of the hi-res screen. Now type "C" to clear your CHR$ SQR. Then press "U" to <U>pload the invader, back into the CHR$ SQR.

Other MODE: 1 Functions

What we're going to do now is edit your invader, Fig. 1, to become the slightly different invader in Fig. 2. This time, though, we're not going to use the cursor keys, but instead we'll use direct number entry.

When characters are stored in memory the pixel data is stored as 8 bytes. A byte is an 8-bit number. This makes 64 bits which can each take the value of 1 or 0 corresponding to a pixel which is "on" or "off". The first byte is the top row of the character. This is referred to as byte 0. The second row is byte 1 and so on up to byte 7. The 8 bits in a byte each have their own values and from left to

right these are:

```
2 to the power seven  =  128  =  bit 7
2 to the power six    =   64  =  bit 6
2 to the power five   =   32  =  bit 5
2 to the power four   =   16  =  bit 4
2 to the power three  =    8  =  bit 3
2 to the power two    =    4  =  bit 2
2 to the power one    =    2  =  bit 1
2 to the power zero   =    1  =  bit 0
```

Let's look at Fig. 1.

```
The far left bit, bit 7, is "off"  = 0 * 128 =  0
The next bit, bit 6, is "on"       = 1 *  64 = 64
The next bit, bit 5, is "off"      = 0 *  32 =  0
The next bit, bit 4, is "off"      = 0 *  16 =  0
The next bit, bit 3, is "off"      = 0 *   8 =  0
The next bit, bit 2, is "off"      = 0 *   4 =  0
The next bit, bit 1, is "on"       = 1 *   2 =  2
The far right bit, bit 0, is "off" = 0 *   1 =  0
```

If we add these, 0+64+0+0+0+0+2+0=66, we get the value of byte 0, the top row, which is 66.

Now try the calculations on the second from top row and you should get 126. You don't need to understand all this binary stuff at all but it can't do any harm to look at it. In fact the 8 numbers making up Figs. 1 and 2 are:

Fig. 1	Fig. 2
66	66
126	129
219	126
219	219
255	219
102	255
195	102
102	129

To enter Fig. 2 directly as 8 numbers, press "N" until the CHR$ cursor stops flashing, then release the "N" key and line:

```
BYTE 0 IS 66 ?
```

will appear. We don't want to change this top row, so just press ENTER and you will get:

```
BYTE 1 IS 126 ?
```

Enter 129 and watch the character change. Now proceed through all eight bytes until you have changed all the Fig. 1 numbers to the Fig. 2 numbers, as in the above list.

Now that we've completed the maths lesson we can get on with something a bit more interesting and look at some more functions.

The hi-res screen window can also be moved from within MODE:1. Let's move it one character to the right. First press SPACEBAR and the current screen window will flash. Whilst the SPACEBAR is held down, the cursor keys can be used to move the hi-res window around. When the cursor keys are released, the panel display will indicate the current window position. If you have moved the window correctly one

characrer right, you should see COL:11 ROW:0. Before we download the new character let's look at the use of colours. Let's start by switching the attribute flag on. Hold down "A" until ATTR displays a 1 on the panel. Let's make this new sprite RED and BLACK.

1. Press "E" until the CHR$ cursor stops flashing.

2. Release "E" and the prompt "ENTER INK TO BE EDITED" will appear.

3. Enter 3.

4. The prompt "NEW VALUE" will now appear. Enter 2 to make the INK RED.

5. Repeat the above but this time enter the INK TO BE EDITED as 4 and the new value as 0.

6. Press "D" to download the new character, together with it's attributes.

Now reset INK3 to 0 and INK4 to 15 using the "E" key.

Let's now look at some MODE:2 operations. To exit to MODE:0 press the Commodore key and "0", then press the Commodore key and "2" to enter MODE:2.

The first thing you'll notice is that the 8x8 grid is replaced by an 8x4 grid. You are now in 4 colour mode. This mode operates in a similar manner to MODE:1 but sprites cannot be entered as binary numbers. In fact 4 colour sprites can be entered as binary numbers in MODE:1 and used in 4 colour mode. The arithmetic is much more complicated and beyond the scope of this manual.

The same keys (cursor keys) are used to move the CHR$ cursor around the screen and when the appropriate cell is reached one of the 4 colours is selected by pressing:

"1" for INK 1	Initially set to GREEN
"2" for INK 2	Initially set to RED
"3" for INK 3	Initially set to BLACK
"4" for PAPER	Initially set to LIGHT GREY

Now design the character in Figure 3.

1	1	1	1
1	1	1	1
1	2	2	2
1	2	2	2
1	2	3	3
1	2	3	3
1	2	3	4
1	2	3	4

Fig 3

The numbers 1 to 4 indicate INKs 1 to 3 and PAPER.

Suppose you now decide to change INK 3 from BLACK to BLUE. Press "E" and release when the CHR$ cursor stops flashing. Enter INK TO BE EDITED as 3 and the new value as 6 (BLUE). Now watch as all the BLACK cells change to BLUE. Be careful not to set two INKs or the PAPER to have the same value or they will become indistinguishable. Press the SPACEBAR and the cursor keys to move the hi-res window to COL:12, ROW:0. Note that the characters downloaded in MODE:1 are still on the screen and appear as "garbage".

Now set ATTR to 1 and press "D" to download the character to the hi-res screen. Now let's see what happens if we make the mistake of setting INK 3 to the value of INK 1 (GREEN). Press "E", enter 3 as the INK to be edited and 5 as the new value. All the BLUE cells change to GREEN. Now press "E" again, and this time edit INK 3, currently GREEN, to be BLUE again. Notice that "all" the GREEN cells turn BLUE. Now edit INK 1 to be YELLOW (7) and nothing happens because there are no GREEN cells to turn YELLOW.

Don't worry, all is not lost. Since ATTR is 1 and the hi-res window is over the character you last downloaded, press "U" and hey-presto, the character re-appears and all the INKs and PAPER are reset to the values they were, when the sprite was downloaded.

Now use "E" to edit INK 3 (currently BLUE) to become PURPLE (4). Now press "A" to switch ATTR to 0. Press "C" to clear the CHR$ SQR and then press "U" to upload. This time the character re-appears with the INKs and PAPER colours that are currently displayed in the panel. It should now be clear that if ATTR is zero colours are not moved with the character but if ATTR is 1 the destination takes the attributes of the source. This is true in MODE:1 and MODE:2.

Now let's move on to something more interesting. Let's look at MODE:3.

In order to fully demonstrate the MODE:3 commands we really need some data on the screen so lets just jump ahead for a moment. Before doing that we want to return to MODE:1 so that we are placed in 2 colour mode (The commands also work in colour mode). So type commodore key "0", to enter MODE:0, commodore key "1" to enter MODE:1, commodore key "0" to enter to MODE:0 again and finally commodore key "3" to enter MODE:3. Now set ATTR to 0 and press RIGHT SHIFT and HOME. This will clear the screen and put the hi-res window to the top right i.e. COL:10 ROW:0. This is where we jump ahead for a moment. Type commodore key 0 to exit MODE:3 then type commodore key "4" to enter MODE:4 .

Put the arcade sprite in full colour onto the hires screen, type "A" until ATTR is 1, then press "P" until the CHR$ curser stops flashing, then release and enter "38" in response to "ENTER SPRITE NUMBER". Once the sprite has been Put to the screen, exit using commodore key "0" and enter MODE:3 using commodore key "3". We'll return to MODE:4 later.

The hires window should be positioned so that its top left and the top left of sprite 38 coincide. What we have to do now is to change the size of the hi-res window so that it has the same dimensions as sprite 38. To do this press LEFT SHIFT and use the curser keys to extend or contract the window in the horizontal and vertical direction. Use there keys to make the hires window have the same dimensions as sprite 38 ie. HGT = 3 and WID = 6. We're now ready to look at some of the window operations. To begin with set ATTR to 1.

Press "W" and then "I" together. The window will invert (1's compliment). To invert the window back release both keys and repeat the operation. Press "W" and then "F" together. The window will "FLIP", ie mirror about a horizontal line accross the centre. Since ATTR was 1 the attributes flipped along with the pixel data. Release both keys and repeat the operation to return the original sprite.

Press "W" and them "M" together. The window will "Mirror" about its vertical centre. Again, since ATTR was 1 the attributes were also mirrored. To mirror the sprite back, release both keys and repeat the operation.

Lets now look at the fine scroll commands. In order to do this we'll first need to extend the hi-res window horizontally by one character. To do this press LEFT SHIFT and RIGHT ARROW. Let's now scroll this with wrap, right by 1 pixel. To do this press "R" and then RIGHT ARROW. Now release both keys and repeat a further 7 times until the sprite has scrolled by 1 full character to the right. Notice that

attributes are not scrolled with the character. To return the pixel data, to its attributes press "R" and then RIGHT SHIFT and the curser keys until the separation is completed.

Finally, lets look at another very useful function. Although the hires screen itself only occupies the top 15 ROWS, it is possible to store sprites under the text panel. Set ATTR to 0 using the "A" key. Now press "V" and use the vertical curser keys to scroll the whole hi-res screen upward. The sprite will disappear off the top of the screen but if you keep pressing, will eventually emerge from the bottom of the hi-res screen. This covers the use of MODE:3 commands, lets move on to MODE:4. Before exiting set ATTR to 0 then press RIGHT SHIFT and HOME to clear the hires screen. Now press the commodore key and 0 followed by the commodore key and 4 to enter MODE:4.

MODE:4 operations are chiefly concerned with sprite operations as opposed to hires window operations.

Press "P" and in response to the prompt "ENTER SPRITE NUMBER" input 2. Now use the SPACEBAR and curser keys to move the hi-res window to the right of sprite 2. Press "P" again and this time enter sprite number as 3. Now move the window back to the top left of sprite 2. Use LEFT SHIFT and the curser keys to enlarge the hi-res window to embrace both sprites 2 and 3. Press "G". In response to the prompt "ENTER SPRITE NUMBER", input 2. "SPRITE ALREADY DEFINED <CR>" will appear. You have tried to redefine sprite 2. Ok let's wipe sprite 2. Press "W" and in response to "ENTER SPRITE NUMBER" type 2. Now type "G" again, enter a sprite number of 2 and this time no error will occur. Move the hi-res window down below the sprite above and press "P", entering 2, to Put the new sprite (comprised of the old sprites 2 and 3) to the hi-res screen.

We now go on to consider some of the advanced commands, all of which should be used with great care.

Quite often it is required to produce large sprites, wider than a screen. These are useful for scrolling landscapes etc. There is insufficient sprite space at the moment because you have the arcade sprites in memory but the technique can be demonstrated using a smaller sprite.

Use the "W" facility to wipe sprites 1 to 3. Set ATTR to 1. Press "C" and enter a sprite number of 1, height of 2 and width of 8. This has created a sprite in memory which we can now fill with data.

Hold down the "M" key until the CHR$ cursor ceases flashing and then release. Enter target sprite number of 1, target column 0 and target row 0. Now enter source sprite as 4, source column as 0 and source row as 0. Now enter a window width of 4 and a widow height of 2. Finally enter the operation as "B". This will block shift a window 4 characters wide and two characters high from the top left of sprite 4 to the top left of sprite 1 since ATTR was 1 the attributes will also have been moved. To examine your handiwork press "P" and enter a sprite number of 1. The sprite should be empty, except for sprite 4 at the top left.

Hold down the "M" key again, until the CHR$ cursor ceases flashing and then release. This time enter target sprite number, column and row as 1,4 and 0 respectively. Enter the source sprite number column and row as 8,0 and 0 respectively. Enter the window width as 2 and height as 2. Finally enter the operation as "S". Again use "P" to put sprite 1 to the screen. Sprite 8 has been rotated clockwise 90 degrees with attributes and placed in sprite 1 at column 4.

Using the "M" key again enter the target sprite number column and row as 1,5, and 0. Enter the source sprite number, column and row as 8,0,0. Enter a window of height and width 2 and enter the operation as "Y". "A SPRITE PARAMETER OUT OF RANGE" error is generated because the expanded character couldn't fit into sprite 1.

Now spend more time experimenting with the other "M" operations.

One last function is provided to enable the user to develop animated graphics. For the purpose of this exercise we'll use the sequence of sprites from 100 to 109.

Press "R" until the CHR$ SQR cursor ceases flashing and release. Enter 100 as sprite number, 09 as number in series and 30 as delay factor. Repeat this procedure with different delays.

This completes the sample session.

FUNCTION KEY SUMMARY

MODE: 1

KEYS	FUNCTIONS
cursor keys in conjunction with right shift.	Movement of the CHR$ cursor
D	Download current CHR$ SQR to hi-res screen window.
U	Upload character from hi-res window to CHR$ SQR.
A	Toggle attribute flag.
E	Enter INK/PAPER editing mode.
N	Enter numerical entry mode.
C	Clear CHR$ SQR.
SPACEBAR	Flash hi-res window.
SPACEBAR & CURSOR KEYS	More hi-res windows.
COMMODORE KEY	Change mode.
-	Fill hi-res window with current attributes.
+	Set INKs and PAPER to those of top left hi-res window.

MODE: 2

KEYS	FUNCTIONS
cursor keys in conjuction with RIGHT SHIFT	Movement of CHR$ cursor
D	Download current CHR$ SQR to hi-res screen window.
U	Upload character frorm hi-res window to CHR$ Square.
A	Toggle attribute flag.

E	Enter INK/PAPER editing mode.
C	Clear CHR$ SQR.
SPACEBAR	Flash hi-res window.
SPACEBAR & CURSOR KEYS	More hi-res windows.
COMMODORE KEY	Change mode.
-	Fill hi-res window with current attributes.
+	Set INKs and PAPER to those of top left hi-res window.

MODE: 3

KEYS	FUNCTIONS
W & I	Invert screen window.
W & F	Flip screen window.
W & M	Mirror screen window.
W & C	Clear screen window.
S & Cursor keys	Fine scroll screen window without wrap.
R & Cursor keys	Find scroll screen window with wrap.
HOME	Move hi-res window to top left of hi-res screen.
RIGHT SHIFT HOME	Clear hi-res screen and move hi-res window to top left of hi-res screen.
V & Cursor keys	Scroll hi-res screen vertically with wrap.
E	Enter INK/PAPER editing mode.
A	Toggle attribute flag.
SPACEBAR	Flash screen window.
SPACEBAR & Cursor keys	Move hires window.
COMMODORE KEY	Change mode.
LEFT SHIFT & Cursor keys	Modify hires window dimensions.
-	Fill hi-res window with current attributes.
+	Set INKs and PAPER to those of top left hi-res window.

MODE: 4

KEYS	FUNCTIONS
G	Get hires window and define as a sprite.
P	Put sprite to the current hires window position.
C	Create sprite of user defined dimensions.
W	Wipe sprite and reclaim memory.
I	Interrogate sprite.
R	Run animated sequence of sprites.
A	Toggle attribute flag.
M then B	Block shift sprite.
M then A	AND sprites.
M then O	OR sprites.
M then E	XOR sprites.
M then S	Spin sprites.
M then X	XPAND sprite in X direction.
M then Y	XPAND sprite in Y direction.
SPACEBAR	Flash screen window.
SPACEBAR & Cursor Keys	Move hi-res window.
COMMODORE KEY	Change mode.
LEFT SHIFT & Cursor Keys	Modify hi-res window dimensions.
-	Fill hi-res window with current attributes.
+	Set INKs and PAPER to those of top left hi-res window.

MODE: 5

KEYS	FUNCTIONS
S	SAVE sprites.
L	LOAD sprites.
E	ERASE sprite file (disk only).
R	RENAME sprite file (disk only).
C	COPY sprite file (disk only).
I	INITIALISE DISK (disk only).
V	VALIDATE DISK (disk only).
D	List directory (disk only).
COMMODORE KEY	Change mode.

Notation

Angle brackets are used to enclose symbols. Most of these are self-explanatory: <integer>, <variable> etc. A <label> is either an <integer> or an <identifier>, being a letter followed by zero or more alphanumeric characters. "|" is used to denote "or". Items inside square brackets are optional and items in curly brackets can be repeated zero or more times.

1. Structured Programming and Miscellaneous Commands

$

This is used before a hexadecimal number.

e.g. J=DEEK($C000)

' (apostrophe)

This is equivalent to :REM and is used before comments.

=

=<expression>

COMMAND: When used as a command, this is used to signify the end of a multi-line user defined function, equating the value of the function to the value of the expression.

ALLOCATE

ALLOCATE <expression>,<expression>,<expression>,<expression>

COMMAND: This allocates the amount of memory to be used for the variables for concurrent tasks 1, 2, 3 and 4. Thus ALLOCATE 0,0,0,0 will reserve no memory, and ALLOCATE 100,0,0,0 would reserve 100 bytes for task number 1.

This command stops execution of any background tasks and clears all variables from the foreground program and it may not be used by one of the background tasks.

Associated keywords: HALT, IMPORT, TASK.

CASE

CASE <expression>

COMMAND: This is used in conjunction with OF and CASEND to select between several courses of action.

Associated keywords: OF, CASEND.

CASEND

COMMAND: This denotes the end of a CASE statement.

Associated words: CASE, OF.

CELSE

COMMAND: This is used as part of the multiple-line IF-THEN-ELSE in conjunction with CIF and CEND.

CEND

COMMAND: This is used to mark the end of a multi-line IF-THEN-ELSE.

CFN

CFN<label>|CFN<label> (<parameter>{,<parameter>})

FUNCTION: Returns the value given by a multi-line function.

Associated keywords: LABEL, PROC, =, LOCAL.

CIF

CIF<expression>

COMMAND: This is the multiple-line equivalent of IF. The end of the statement is denoted by CEND.

COMMON%

COMMON%(<expression>)

This is a pseudo-array which is shared by all five concurrent tasks. It contains 64 elements and can be used to pass values between the tasks.

DSAVE

DSAVE <string>[,<expression>[,<expression>]]

COMMAND: This is equivalent to SAVE, the difference being that the default device number is 8 (the disk drive).

Associated keywords: DLOAD

DEEK

DEEK (<expression>)

FUNCTION: This is the double-byte version of PEEK - it returns the value of two memory locations in the usual low-byte/high-byte order.

e.g. I=DEEK(49152)
is equivalent to
I=PEEK(49152)+256*PEEK(49153)

Associated keywords: DOKE

DIR [<filename>[,<device>]]

COMMAND: print disk directory.

DISABLE

COMMAND: This disables the run-stop key. It should be used as a command inside a program rather than being typed from direct mode.

DLOAD

DLOAD <string>[,<expression>[,<expression>]]

COMMAND: This is equivalent to LOAD, the difference being that the default device number is 8 (the disk drive).

Associated keywords: DSAVE

DOKE

DOKE <expression>,<expression>

COMMAND: This is the double-byte version of POKE.

e.g. DOKE A,B
is equivalent to
POKE A,B AND 255 : POKE A+1, INT(B/256)

ELSE

This is an addition to the IF...THEN command already existing in BASIC. If the condition is false, the part after the ELSE is executed. If it is true, the part after THEN is executed.

Associated keywords: IF, THEN.

EXIT

COMMAND: This is used to leave a loop prematurely. It can be used with FOR-NEXT, REPEAT-UNTIL or WHILE-WEND loops.

Associated keywords: FOR, NEXT, WHILE, WEND, REPEAT, UNTIL.

N.B. When using EXIT in conjunction with FOR-NEXT loops, each NEXT must correspond to one FOR only.

FALSE

FUNCTION: This returns a value of zero.

Associated keywords: TRUE.

HALT

HALT <expression>

COMMAND: This stops execution of a background task.

Associated keywords: TASK, ALLOCATE.

HEX$

HEX$(<expression>)

FUNCTION: This returns a string with the hexadecimal representation of the numerical value of <expression>. The string is always five characters long, and in the form $XXXX.

e.g. HEX$(60000) returns "$EA60"

IMPORT

IMPORT(<expression>,<expression>)

FUNCTION: The first expression is a task number; this function returns the value of the second expression as evaluated by that task. A task number of zero means the foreground task. Values cannot be passed when a command is actually being executed - for example in the middle of an INPUT statement.

LABEL

LABEL<identifier>|LABEL<identifier>(<parameter>{,<parameter>})

LABEL is used to define a label for use by GOTOs, GOSUBs and RESTOREs or to define procedures and multi-line functions.

To define labels for GOTOs, GOSUBs or RESTOREs, the LABEL is just followed by the identifier.

To define procedures and multiple-line functions, the label is followed by a list of parameters inside brackets. If you want the actual parameter to be altered by the procedure, precede the variable name with VAR. ARRAYS MUST BE DECLARED AS VAR PARAMETERS.

LOCAL

LOCAL <variable>{,<variable>}

This is used inside a procedure to create variables (arrays are not allowed) which will be destroyed upon leaving the procedure or function. Variables created in this way are independent of those in the main program.

OF

OF <expression>

COMMAND: This is used in conjunction with the CASE and CASEND statements. If the <expression> after the OF is the same value as the <expression> after the CASE, the set of commands until the next OF or CASEND are obeyed, otherwise they are not.

Also, the use of OF OR is allowed - this results in the execution of the following statements only if none of the OFs in the current CASE have been executed.

Associated keywords: CASE, CASEND.

OLD

COMMAND: Restore NEWed program.

PROC

PROC<label>| PROC<label>(<parameter>{,<parameter>})

(Simple variables may be used as <parameters>; to pass an array as a parameter, follow the array name with an empty set of brackets: A$(), A()).

COMMAND: Calls a procedure using the specified parameters (if any).

Associated keywords: LABEL, PROCEND, SIZE, LOCAL.

PROCEND

COMMAND: This is used at the end of a procedure definition.

Associated keywords: PROC, LABEL, LOCAL.

PULL

COMMAND: This removes one level of subroutine nesting from the stack.

Associated keywords: GOSUB, RETURN.

REPEAT

COMMAND: This is used in conjunction with the UNTIL command to set up a loop for situations in which it is required to execute the body of the loop at least once.

Associated keywords: UNTIL.

RPT

COMMAND: This executes a graphics command several times.

RPT<expression>,<graphics command>

For example:

```
RPT 8, WRR1
```

is equivalent to

```
FOR I=0 TO 7 : WRR1 : NEXT
```

but the first version is faster.

SIZE

SIZE(<variable>,<expression>)

FUNCTION: This gives the size of an array +1. If <expression> = 0, the number of dimensions is returned.

TASK

TASK <expression>,<label>

This initialises execution of a background task at <label>. The <expression> is the task number and may be 1, 2, 3 or 4.

TRUE

FUNCTION: This returns a value of -1.

Associated keywords: FALSE

UNTIL

UNTIL <expression>

This is used in conjunction with REPEAT, the UNTIL going at the end of the loop. Execution of the loop continues until the <expression> is non-zero.

Associated keywords: REPEAT

WEND

COMMAND: This is used in conjunction with the WHILE command: WEND marks the end of the loop.

Associated keywords: WHILE

WHILE

WHILE <expression>

COMMAND: Sets up a loop, in conjunction with the WEND where it is required to execute the body of the loop zero or more times.

Associated keywords: WEND

PI

FUNCTION: Returns the value of 3.14159265

NOTE: In the following commands, if the parameters are missing, the interpreter uses existing values. This only applies to the commands marked with an asterisk below.

In describing operations on windows,

W1 XOR W2 -> W1

would mean:

"Window 1 exclusive-ored with window 2 goes into window 1"

'S1' and 'S2' are used in the same way, meaning 'sprite'.

IDEAL pseudo-variables:

SPN COL ROW WID HGT
SPN2 COL2 ROW2
NUM INC ATR
CCOL CROW

.2COL <Sprite no.>

COMMAND: puts a hardware sprite (number is zero to 7) into two-colour mode.

.4COL <Sprite #>

COMMAND: puts a hardware sprite into 4-colour mode.

.COL <sprite #>,<colour>

COMMAND: sets the colour of a hardware sprite. (<colour> is 0..15).

.COL0 <colour>

COMMAND: sets multicolour sprite colour #0

.COL1 <colour>

COMMAND: sets multicolour sprite colour #1

.HIT <sprite #>

FUNCTION: returns a true value if the hardware sprite has hit something.

.OFF <sprite #>

COMMAND: removes a hardware sprite form the screen.

.ON <sprite #>

COMMAND: turns on a hardware sprite.

.OVER <sprite #>

COMMAND: Background is given priority over the hardware sprite.

.SET <sprite#>,<definition#>

COMMAND: Associate a hardware sprite with its definition.

.SHRINKX <sprite #>

COMMAND: The sprite is given normal size in the X-direction.

.SHRINKY <sprite #>

COMMAND: The sprite is given normal size in the Y-direction.

.UNDER <sprite #>

COMMAND: The background goes under the hardware sprite.

.XPANDX <sprite #>

COMMAND: The sprite is expanded to double size in the X-direction.

.XPANDY <sprite #>

COMMAND: The sprite is expanded to double size in the Y-direction.

.XPOS <sprite #>,<position>

COMMAND: Sets up the X position of hardware sprite.

ADSR voice,attack,decay,sustain,release

COMMAND: specify envelope

.YPOS <sprite #>,<position>

COMMAND: Sets up Y position of hardware sprite.

AFA(SPN)

FUNCTION: Returns the attribute field address of sprite. -1 if non-existant.

AFA2(SPN)

FUNCTION: Returns the 2ndary attribute field address of a sprite. -1 if non-existant.

* AND%AND SPN,ROW,COL,HGT,WID,SPN2,ROW2,COL2

COMMAND: W1 AND W2 ->W1
W1 AND W2 ->W2

* AND%BLK SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2

COMMAND: W1->W2
W1 AND W2->W1

* AND%OR SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2

COMMAND: W1 OR W2 ->W2
W1 AND W2 ->W1

* AND%XOR SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2

COMMAND: W1 XOR W2 ->W2
W1 AND W2 ->W1

ATT2ON

COMMAND: Enables movement of both sets of attributes with the data movement commands.

* ATTDN SPN,COL,ROW,WID,HGT

COMMAND: Scrolls down all attributes in a window by 1 character block with wrap.

* ATTGET SPN,COL,ROW

COMMAND: Puts the attribute at the specified position into ATR.

* ATTL SPN,COL,ROW,WID,HGT

COMMAND: Scrolls attributes in a window left by one character with wrap.

ATTOFF

COMMAND: Disables movement of attributes when pixel data is moved.

ATTON

COMMAND: Enables movement of primary set of attributes only.

* ATTR SPN,COL,ROW,WID,HGT

COMMAND: Scrolls attributes in window right with wrap.

* ATTUP SPN,COL,ROW,WID,HGT

COMMAND: Scrolls attributes in window up with wrap.

* BLK%AND SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2

COMMAND: W1 AND W2 ->W2
W2 -> W1

* BLK%BLK SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2

COMMAND: W1->W2
W2->W1

* BLK%OR SPN,ROW,COL,HGT,WID,SPN2,ROW2,COL2

COMMAND: W1 OR W2 ->W2
W2 ->W1

* BLK%XOR SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2

COMMAND: W1 XOR W2 ->W2
W2 -> W1

* BOX SPN,COL,ROW,WID,HGT

COMMAND: Fills a block of pixels inside a sprite - this command is dependant on the current value of MODE.

* CHAR SPN,COL,ROW,NUM

COMMAND: Puts a character in a sprite at character block position specified.

Value of NUM

0 to 255	ASCII characters
256 to 511	reverse ASCII characters
512 to 767	display codes
1024 to 1279	double width ASCII characters
1280 to 1535	reverse double width ASCII characters
1536 to 1791	double width display codes

* CPYAND SPN,SPN2

COMMAND: S1 AND S2 -> S2

* CPYBLK SPN,SPN2 S1 -> S2
* CPYOR SPN,SPN2 S1 OR S2 -> S2
* CPYXOR SPN,SPN2 S1 XOR S2 -> S2

CUTOFF frequency

COMMAND: Set cutoff frequency (0..2047) for filter.

DBLANK

COMMAND: Blanks the screen to border colour.

DFA(SPN)

FUNCTION: Returns pixel data address of a sprite. -1 if non-existant.

DMERGE "filename"[,device]

COMMAND: Merge sprites from disk to those in memory.

* DRAW SPN,COL,ROW,COL2,ROW2

COMMAND: Draws a line from (COL,ROW) to (COL2,ROW2).

DRECALL "filename"[,device]

COMMAND: Load new sprites from disk.

DSHOW

COMMAND: Enable screen display (opposite of DBLANK)

DSTORE "filename"[,device]

COMMAND: Save sprites to disk.

DTCTOFF

COMMAND: Turn off collision detection

DTCTON

COMMAND: Turn on collision detection

ENV

FUNCTION: Return output from oscillator 3 envelope generator.

FILTER voice,flag

COMMAND: enable/disable filtering of a voice.

FIRE1

FUNCTION: Return true flag if joystick in port 1 has fire button pressed.

FIRE2

FUNCTION: Return true flag if joystick in port 2 has fire button pressed.

* FLIP SPN,COL,ROW,WID,HGT

COMMAND: Flip over window top to bottom.

* FLIPA SPN,COL,ROW,WID,HGT

COMMAND: Flip over attributes top to bottom.

FRQ voice, frequency

COMMAND: Set frequency

* GETAND: SPN,COL,ROW

Copy screen at (COL,ROW) into SPN with AND.

Similarly, GETBLK, GETOR and GETXOR

H38COL

COMMAND: Shrink display.

H40COL

COMMAND: Expand display to normal size

HBORDER <colour>

COMMAND: Sets border colour for hi-resolution screen.

HIRES

COMMAND: Go into hires mode.

HPAPER <colour>

COMMAND: Set hi-res background colour (applies to 4-colour mode only).

INK <colour>

COMMAND: Set INK colour for printing.

* INV SPN,COL,ROW,WID,HGT

COMMAND: Invert window.

JS1

FUNCTION: Returns direction of joystick in port 1.

JS2

FUNCTION: Returns direction of joystick in port 2.

KB(n)

FUNCTION: Return true value if key no. n is pressed.

LCASE

COMMAND: go into lower case.

LORES

COMMAND: go into LORES mode.

LPX

FUNCTION: Returns light-pen X-position.

LPY

FUNCTION: Returns light-pen Y-position.

* MAR SPN,COL,ROW,WID,HGT

COMMAND: Mirror attributes left-to-right in window.

MERGE ["filename"[,device]]

COMMAND: Load in sprites, keeping existing ones.

* MIR SPN,COL,ROW,WID,HGT

COMMAND: Mirror data left-to-right in window.

MODE <expression>

COMMAND: Sets mode number for PLOT, POLY, BOX etc.:

0 to 3 - colour to be plotted in multi-colour code.
0 or 1 - clear point in 2 colour mode.
2 or 3 - set point in 2 colour mode.
4 - invert point.

MONO

COMMAND: Puts hardware into 2-colour mode.

* MOVAND SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2

COMMAND: W1 AND W2 -> W2

* MOVATT SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2 (move attributes only)
* MOVBLK SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2 W1 -> W2
* MOVOR SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2 W1 OR W2 -> W2
* MOVXOR SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2 W1 XOR W2 -> W2

MUTE flag

COMMAND: enables/disable muting of voice 3.

MUSIC voice,length

COMMAND: Sound note; length in 60ths of a second.

NOISE voice

COMMAND: Set up voice to generate noise.

MULTI

COMMAND: Puts hardware into 4-colour mode.

```
*  OR%AND          SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
*  OR%BLK          SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
*  OR%OR           SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
*  OR%XOR          SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
```

OSC

FUNCTION: Return output from voice 3 oscillator.

PASS N

COMMAND: Set filter to low pass (n=0), high pass (1), band pass (2), or notch reject (3).

* PLOT SPN,COL,ROW

COMMAND: Plot a point.

POINT (SPN,COL,ROW)

FUNCTION: Return value of point referenced.

0 or 1 if in S2COL mode.
0, 1, 2, or 3 if in S4COL mode.

* POLY SPN,COL,ROW,WID,HGT,NUM,INC

COMMAND: Draw polygon

PULSE voice,width

COMMAND: Set voice to generate pulse wave form; 0<width<4096.

* PUTAND SPN,COL,ROW

Move sprite to screen at (COL,ROW), ANDing with screen.

```
*  PUTBLK    SPN,COL2,ROW2    move directly to screen.
*  PUTOR     SPN,COL2,ROW2    OR with screen.
*  PUTXOR    SPN,COL2,ROW2    XOR with screen.

*  PUTCHR  SPN,COL,ROW,NUM
```

Copy character block from sprite into character memory. NUM same as for CHAR.

```
RECALL    ["filename"[,device]]
```

COMMAND: Load in new sprites.

```
RESERVE    <expression>
```

COMMAND: Reserve space for sprites.

```
RESET
```

COMMAND: Erase all sprites and reset sprite storage.

```
RESEQ
```

COMMAND: Renumber sprites.

```
RESONANCE n
```

COMMAND: Set resonance (0..15) of filter.

```
RING voice,flag
```

COMMAND: enable/disable ring modulation.

```
RPT <expr>, command
```

COMMAND: Execute a graphics command more than once.

```
S2COL
```

COMMAND: Puts software into 2-colour mode.

```
S4COL
```

COMMAND: Puts software into 4-colour mode.

```
SAW  voice
```

COMMAND: Set voice to generate sawtooth waves.

```
SCAN(SPN,COL,ROW,WID,HGT)
```

FUNCTION: Returns true value if window contains data.

* SCL1 SPN,COL,ROW,WID,HGT

COMMAND: Scroll left by one pixel.

similarly: SCL2,SCL8,SCR1,SCR2,SCR8

* SCLR SPN,ATR

COMMAND: Clear sprite.

SCRLX <expr>

COMMAND: Shift screen left/right by 0 to 7 pixels.

* SCROLL SPN,COL,ROW,WID,HGT,NUM

COMMAND: Scroll window vertically by NUM pixels. NUM>0 = up, NUM<0 = down.

SCRLY <expr>

COMMAND: Shift screen up/down by 0 to 7 pixels.

* SETA SPN,COL,ROW,WID,HGT,ATR

COMMAND: Set attributes in window to ATR

SIDCLR

COMMAND: Reset sound chip.

* SPIN SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2

COMMAND: Rotate W1 ->W2 by 90 degrees clockwise.

* SPRCONV SPN,COL,ROW,SPN2

COMMAND: Convert software to hardware sprite.

* SPRITE SPN,WID,HGT

COMMAND: Create new sprite.

STORE ["filename"[,device]]

COMMAND: Save sprites.

STRPLOT SPN,COL,ROW,A$,<offset>

COMMAND: Put text in sprite (cannot be used with RPT).

```
Offset
  1          normal characters
 256         inverse characters
1024         double-width characters
1280         double-width inverse characters
2048         double-spaced characters
2304         double-spaced inverse characters
```

* SWAPATT SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2

COMMAND: Swap attributes in windows.

SYNC voice, flag

COMMAND: enable/disable synchronisation of a voice

TBORDER <colour>

COMMAND: Set text border colour

TPAPER <colour>

COMMAND: Set text paper colour.

TRI voice

COMMAND: Set voice to generate triangular waves.

UCASE

COMMAND: Go into upper case.

VOLUME number

COMMAND: Set master volume (0..15).

WCLR SPN,COL,ROW,WID,HGT,ATR

COMMAND: Clear window.

WINDOW <expr>

COMMAND: Set up hires/text window.

* WIPE SPN

COMMAND: Remove sprite from sprite table.

* WRAP SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2,NUM

COMMAND: Scroll window by NUM pixels with wrap.

* WRL1 SPN,COL,ROW,WID,HGT

COMMAND: Wrap window one pixel left.

Similarly - WRL2,WRL8,WRR1,WRR8

* XOR%AND SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
W1 XOR W2 -> W1, W1 AND W2 -> W1

* XOR%BLK SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
W1 XOR W2 -> W1, W1 -> W2

* XOR%OR SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
W1 XOR W2 -> W1, W1 OR W2 -> W2

* XOR%XOR SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
W1 XOR W2 -> W1 W1 XOR W2 -> W2

* XPANDX SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2

COMMAND: Expand W1 -> W2 in X direction.

* XPANDY SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2

COMMAND: Expand W1 -> W2 in Y direction.

APPENDIX B: BASIC LIGHTNING TOKENS

Token		Keyword	abbr				
128	$80	END	E.	176	$B0	OR	-
129	$81	FOR	F.	177	$B1	>	-
130	$82	NEXT	N.	178	$B2	=	-
131	$83	DATA	D.	179	$B3	<	-
132	$84	INPUT#	I.	180	$B4	SGN	SG.
133	$85	INPUT	-	181	$B5	INT	-
134	$86	DIM	DI.	182	$B6	ABS	AB.
135	$87	READ	R.	183	$B7	USR	US.
136	$88	LET	L.	184	$B8	FRE	FR.
137	$89	GOTO	G.	185	$B9	POS	-
138	$8A	RUN	RU.	186	$BA	SQR	SQ.
139	$8B	IF	-	187	$BB	RND	RN.
140	$8C	RESTORE	RES.	188	$BC	LOG	-
141	$8D	GOSUB	GOS.	189	$BD	EXP	EX.
142	$8E	RETURN	RE.	190	$BE	COS	-
143	$8F	REM	-	191	$BF	SIN	SI.
144	$90	STOP	ST.	192	$C0	TAN	-
145	$91	ON	O.	193	$C1	ATN	AT.
146	$92	WAIT	W.	194	$C2	PEEK	PE.
147	$93	LOAD	LO.	195	$C3	LEN	-
148	$94	SAVE	SA.	196	$C4	STR$	STR.
149	$95	VERIFY	V.	197	$C5	VAL	VA.
150	$96	DEF	DE.	198	$C6	ASC	AS.
151	$97	POKE	P.	199	$C7	CHR$	CH.
152	$98	PRINT#	PR.	200	$C8	LEFT$	LEF.
153	$99	PRINT	?	201	$C9	RIGHT$	RI.
154	$9A	CONT	C.	202	$CA	MID$	M.
155	$9B	LIST	LI.	203	$CB	GO	-
156	$9C	CLR	CL.	204	$CC	ELSE	EL.
157	$9D	CMD	CM.	205	$CD	HEX$	H.
158	$9E	SYS	SY.	206	$CE	DEEK	DEEK.
159	$9F	OPEN	OP.	207	$CF	TRUE	TR.
160	$A0	CLOSE	CLO.	208	$D0	IMPORT	IM.
161	$A1	GET	GE.	209	$D1	CFN	CF.
162	$A2	NEW	NE.	210	$D2	SIZE	SIZ.
163	$A3	TAB(	T.	211	$D3	FALSE	FA.
164	$A4	TO	-	212	$D4	(RESERVED FOR FUTURE EXPANSION	
165	$A5	FN	-	213	$D5	LPX	LP.
166	$A6	SPC(	SP.	214	$D6	LPY	-
167	$A7	THEN	TH.	215	$D7	COMMON%	COM.
168	$A8	NOT	NO.	216	$D8	CROW	CR.
169	$A9	STEP	S.	217	$D9	CCOL	CC.
170	$AA	+	-	218	$DA	ATR	-
171	$AB	-	-	219	$DB	INC	-
172	$AC	*	-	220	$DC	NUM	NU.
173	$AD	/	-	221	$DD	ROW2	RO.
174	$AE		-	222	$DE	COL2	COL.
175	$AF	AND	A.	223	$DF	SPN2	SPN.

224	$E0	HGT	HG.	1,16	$10	SCRLX	SCR.
225	$E1	WID	WI.	1,17	$11	WRR1	WR.
226	$E2	ROW	-	1,18	$12	WRL1	WRL.
227	$E3	COL	-	1,19	$13	SCR1	-
228	$E4	SPN	-	1,20	$14	SCL1	-
229	$E5	TASK	TAS.	1,21	$15	WRR2	-
230	$E6	HALT	HA.	1,22	$16	WRL2	-
231	$E7	REPEAT	REP.	1,23	$17	SCR2	-
232	$E8	UNTIL	UN.	1,24	$18	SCL2	-
233	$E9	WHILE	WH.	1,25	$19	WRR8	-
234	$EA	WEND	WE.	1,26	$1A	WRL8	-
235	$EB	CIF	CI.	1,27	$1B	SCR8	-
236	$EC	CELSE	CE.	1,28	$1C	SCL8	-
237	$ED	CEND	CEN.	1,29	$1D	ATTR	ATT.
238	$EE	LABEL	LA.	1,30	$1E	ATTL	-
239	$EF	DOKE	DO.	1,31	$1F	ATTUP	ATTU.
240	$F0	EXIT	EX.	1,32	$20	ATTDN	ATTD.
241	$F1	ALLOCATE	AL.	1,33	$21	CHAR	CHA.
242	$F2	DISABLE	DIS.	1,34	$22	WINDOW	WIN.
243	$F3	PULL	PU.	1,35	$23	MULTI	MU.
244	$F4	DLOAD	DL.	1,36	$24	MONO	MON.
245	$F5	DSAVE	DS.	1,37	$25	TBORDER	TB.
246	$F6	VAR	-	1,38	$26	HBORDER	HB.
247	$F7	LOCAL	LOC.	1,39	$27	TPAPER	TP.
248	$F8	PROCEND	PRO.	1,40	$28	HPAPER	HP.
249	$F9	PROC	-	1,41	$29	WRAP	WRA.
250	$FA	CASEND	CA.	1,42	$2A	SCROLL	SCRO.
251	$FB	OF	-	1,43	$2B	INK	-
252	$FC	CASE	-	1,44	$2C	SETA	-
253	$FD	RPT	RP.	1,45	$2D	ATTGET	ATTG.
254	$FE	SETATR	SE.	1,46	$2E	ATT2ON	ATT2.
255	$FF	PI	-	1,47	$2F	ATTON	ATTO.

TWO-BYTE TOKENS

1,0	$00	unused		1,48	$30	ATTOFF	ATTOF.
1,1	$01	SCLR	SC.	1,49	$31	MIR	-
1,2	$02	SPRITE	SPR.	1,50	$32	MAR	MA.
1,3	$03	WIPE	WIP.	1,51	$33	WCLR	WC.
1,4	$04	RESET	RESE.	1,52	$34	INV	-
1,5	$05	H38COL	H3.	1,53	$35	SPIN	SPI.
1,6	$06	LORES	LOR.	1,54	$36	MOVBLK	MOV.
1,7	$07	HIRES	HI.	1,55	$37	MOVXOR	MOVX.
1,8	$08	PLOT	PL.	1,56	$38	MOVAND	MOVA.
1,9	$09	BOX	BO.	1,57	$39	MOVOR	MOVO.
1,10	$0A	POLY	POL.	1,58	$3A	MOVATT	MOVAT.
1,11	$0B	DRAW	DR.	1,59	$3B	XPANDX	X.
1,12	$0C	MODE	MO.	1,60	$3C	XPANDY	-
1,13	$0D	S2COL	S2.	1,61	$3D	GETBLK	GET.
1,14	$0E	S4COL	S4.	1,62	$3E	PUTBLK	PUT.
1,15	$0F	H40COL	H4.	1,63	$3F	CPYBLK	CP.

1,64	$40	GETXOR	GETX.	1,112	$70	OR%XOR	OR%X.
1,65	$41	PUTXOR	PUTX.	1,113	$71	AND%XOR	AND%X.
1,66	$42	CPYXOR	CPYX.	1,114	$72	XOR%XOR	XOR%X.
1,67	$43	GETOR	GETO.	1,115	$73	STRPLOT	STRP.
1,68	$44	PUTOR	PUTO.	1,116	$74	FLIP	-
1,69	$45	CPYOR	CPYO.	1,117	$75	.HIT	.H.
1,70	$46	GETAND	GETA.	1,118	$76	SCAN	SCA.
1,71	$47	PUTAND	PUTA.	1,119	$77	POINT	POI.
1,72	$48	CPYAND	CPYA.	1,120	$78	DFA	DF.
1,73	$49	DBLANK	DB.	1,121	$79	AFA2	AF.
1,74	$4A	DSHOW	DSH.	1,122	$7A	AFA	-
1,75	$4B	PUTCHR	PUTC.	1,123	$7B	KB	-
1,76	$4C	LCASE	LC.	1,124	$7C	FIRE1	FI.
1,77	$4D	UCASE	UC.	1,125	$7D	FIRE2	-
1,78	$4E	SPRCONV	SPRC.	1,126	$7E	JS1	J.
1,79	$4F	.ON	..	1,127	$7F	JS2	-
1,80	$50	.OFF	.OF.	**2,0**	$00	unused	
1,81	$51	.SET	.S.	**2,1**	$01	BLACK	BLA.
1,82	$52	FLIPA	FL.	**2,2**	$02	WHITE	WHIT.
1,83	$53	.4COL	.4.	**2,3**	$03	RED	-
1,84	$54	.2COL	.2.	**2,4**	$04	CYAN	CY.
1,85	$55	.COL0	.C.	**2,5**	$05	PURPLE	PUR.
1,86	$56	.COL1	-	**2,6**	$06	GREEN	GR.
1,87	$57	.XPANDX	.X.	**2,7**	$07	BLUE	BLU.
1,88	$58	.SHRINKX	.SH.	**2,8**	$08	YELLOW	Y.
1,89	$59	.XPANDY	-	**2,9**	$09	ORANGE	ORA.
1,90	$5A	.SHRINKY	-	2,10	$0A	BROWN	BR.
1,91	$5B	.XPOS	.XPO.	2,11	$0B	.RED	.R.
1,92	$5C	.YPOS	.Y.	2,12	$0C	GRAY1	GRA.
1,93	$5D	.COL	-	2,13	$0D	GRAY2	-
1,94	$5E	.OVER	.OV.	2,14	$0E	.GREEN	.G.
1,95	$5F	.UNDER	.U.	2,15	$0F	.BLUE	.B.
1,96	$60	SWAPATT	SW.	2,16	$10	GRAY3	-
1,97	$61	DTCTON	DT.	2,17	$11	OSC	OS.
1,98	$62	DTCTOFF	DTCTOF.	2,18	$12	ENV	-
1,99	$63	BLK%BLK	BLK.	2,19	$13	FRQ	-
1,100	$64	OR%BLK	OR.	2,20	$14	NOISE	NOI.
1,101	$65	AND%BLK	AND.	2,21	$15	PULSE	PULS.
1,102	$66	XOR%BLK	XO.	2,22	$16	SAW	-
1,103	$67	BLK%OR	BLK%O.	2,23	$17	TRI	-
1,104	$68	OR%OR	OR%O.	2,24	$18	RING	RIN.
1,105	$69	AND%OR	AND%O.	2,25	$19	SYNC	SYN.
1,106	$6A	XOR%OR	XOR%O.	2,26	$1A	MUSIC	MUS.
1,107	$6B	BLK%AND	BLK%A.	2,27	$1B	ADSR	AD.
1,108	$6C	OR%AND	OR%A.	2,28	$1C	FILTER	FIL.
1,109	$6D	AND%AND	AND%A.	2,29	$1D	MUTE	MUT.
1,110	$6E	XOR%AND	XOR%A.	2,30	$1E	VOLUME	VO.
1,111	$6F	BLK%XOR	BLK%X.	2,31	$1F	CUTOFF	CU.

2,32	$20	RESONANCE	RESO.
2,33	$21	PASS	PAS.
2,34	$22	SCRLY	-
2,35	$23	RECALL	REC.
2,36	$24	STORE	STOR.
2,37	$25	SIDCLR	SID.
2,38	$26	MERGE	ME.
2,39	$27	RESEQ	-
2,40	$28	MEM	-
2,41	$29	OLD	-
2,42	$2A	DIR	-
2,43	$2B	DSTORE	DST.
2,44	$2C	DRECALL	DRE.
2,45	$2D	DMERGE	DM.

APPENDIX C – SPRITE STORAGE FORMAT

The location of the start of sprite storage is held in $2FFC and can be obtained using DEEK($2FFC). Similarly, the end of sprites in memory (+1) is obtained using DEEK($2FFE).

Each sprite is stored in memory as follows:-

byte 0 = sprite number.
bytes 1/2 = offset from start of sprite to primary attribute data.
bytes 3/4 = offset from start of sprite to secondary attribute data.
byte 5 = width of sprite in character blocks.
byte 6 = height of sprite in character blocks.
byte 7 onwards = width*height*8 bytes pixel data, then width*height bytes primary attribute data followed by width*height bytes secondary attributes data.

After the last sprite in memory, a dummy zero is stored where the number of the next sprite would be expected.

Each sprite uses up 10*width*height+7 bytes in memory.

APPENDIX D – GLOSSARY OF TERMS

actual parameters	Variable or expression used when calling a procedure or function using PROC or CFN.
ADSR	Attack-decay-sustain-release envelope.
AND	In terms of pixel data, a logical operation indicating that the destination pixel is set only if both source and destination pixels are already set.
attack	The rate at which a musical note reaches its peak volume.
attribute	Data associated with each character block indicating the colours to be used for displaying pixel data. In two-colour mode, one byte of attribute data is associated with each character block. In four-colour mode, two bytes of attribute data are associated with each character block.
bit-map mode	A screen display mode, enabled by the HIRES command, in which each pixel can be individually set or cleared.
BLK	A prefix/suffix used for data movement commands indicating that pixel data at the destination of a move command is to be overwritten.
compiler	A program which takes a program written in BASIC or some other high-level language and translates it into machine code.
decay	The rate at which a musical note falls from its peak volume to the sustain volume.
envelope	The shape of the volume of a musical note over time.
filter	An electronic circuit which removes certain frequencies from a signal.
formal parameters	The variables used as parameters in a procedure or function definition.
four-colour mode	One of the hi-res screen display modes in which each pixel can take on one of four colours. The pixels in this mode are twice as wide as in two-colour mode.
frequency	Cycles per second.
hi-res mode	See "bit map mode"
label	A symbolic name associated with a program statement.
local variable	In a procedure or function, a variable which is created inside the procedure and destroyed upon exit.
lores mode	See "text mode".

MACHINE LIGHTNING	A games-writing utility in which the graphics commands available in BASIC LIGHTNING are controlled by a assembly language program.
multi-colour mode	See "four-colour mode".
multi-tasking	A mode in which the computer runs several programs at once.
OR	In terms of pixel data, a logical operation indicating that the destination pixel is set if either the source or destination pixels are already set.
pixel	An individual 'point' on the screen which can only be accessed individually in bit-map mode.
procedure	A piece of code which can be called with parameters.
recursion	See recursion.
release	The rate at which a musical note falls from sustain volume to no volume.
S2COL mode	A mode, enabled by the S2COL command, in which BASIC LIGHTNING'S graphics commands operate on pixel data to be displayed in two-colour mode.
S4COL mode	A mode, enabled by the S4COL command, in which BASIC LIGHTNING'S graphics commands operate on pixel data to be displayed in four-colour mode.
SID	Sound Interface Device.
sprite variables	A set of 13 pseudo-varibles which can be used to pass parameters to some of the graphics commands.
structured programmming	The coding of algorithms without the use of unconditional jumps.
sustain	The volume level for the sustain of a musical note.
syntax	Programming language sentance structure.
text mode	A mode in which only characters can be displayed on the screen and individual pixels cannot be accessed.
two-colour mode.	One of the hi-res screen display modes in which each pixel can take on one of two colours.
variable parameter	A parameter of a procedure which is altered by the procedure.
WHITE LIGHTNING	A games-writing utility in which the graphics commands available in BASIC LIGHTNING are available from FORTH.
XOR	In terms of pixel data, a logical operation indicating that the destination pixel is set if one but not both of the source and destination pixels are already set.

APPENDIX E–SCREEN DISPLAY CODES

UPPER	LOWER	CODE
@	@	0
A	a	1
B	b	2
C	c	3
D	d	4
E	e	5
F	f	6
G	g	7
H	h	8
I	i	9
J	j	10
K	k	11
L	l	12
M	m	13
N	n	14
O	o	15
P	p	16
Q	q	17
R	r	18
S	s	19
T	t	20
U	u	21
V	v	22
W	w	23
X	x	24
Y	y	25
Z	z	26
		27
		28
		29
		30
		31
SPACE	SPACE	32
!	!	33
"	"	34
#	#	35
$	$	36
%	%	37
&	&	38
'	'	39
(	)	40
)	(	41
*	*	42
+	+	43

UPPER	LOWER	CODE
,	,	44
-	-	45
.	.	46
/	/	47
0	0	48
1	1	49
2	2	50
3	3	51
4	4	52
5	5	53
6	6	54
7	7	55
8	8	56
9	9	57
:	:	58
;	;	59
<	<	60
=	=	61
>	>	62
?	?	63
		64
	A	65
	B	66
	C	67
	D	68
	E	69
	F	70
	G	71
	H	72
	I	73
	J	74
	K	75
	L	76
	M	77
	N	78
	O	79
	P	80
	Q	81
	R	82
	S	83
	T	84
	U	85

UPPER	LOWER	CODE
	V	86
	W	87
	X	88
	Y	89
	Z	90
		91
		92
		93
		94
		95
SPACE	SPACE	96
		97
		98
		99
		100
		101
		102
		103
		104
		105
		106
		107
		108
		109
		110
		111
		112
		113
		114
		115
		116
		117
		118
		119
		120
		121
		122
		123
		124
		125
		126
		127

Codes from 128-255 are reversed images of codes 0-127.

Note: graphics characters are not shown in the above table.

APPENDIX F—ASCII AND CHRS CODES

PRINTS	CHR$	PRINT$	CHR$	PRINTS	CHR$	PRINTS	CHR$
	0	1	49		98	UP	**145**
	1	2	50		99	RVS OFF	146
	2	3	51		100	CLS	147
	3	4	52		101	INST	148
	4	5	53		102	BROWN	149
WHITE	5	6	54		103	L.RED	150
	6	7	55		104	GREY1	151
	7	8	56		105	GREY2	152
	8	9	57		106	L.GRN	153
	9	:	58		107	L.BLUE	154
	10	;	59		108	GREY3	155
	11	<	60		109	PURPLE	156
	12	=	61		110	LEFT	157
RETURN	13	>	62		111	YELLOW	158
	14	?	63		112	CYAN	159
	15	@	64		113	SPACE	160
	16	A	65		114		161
DOWN	17	B	66		115		162
RVS ON	18	C	67		116		**163**
HOME CSR	19	D	68		117		**164**
DELETE	20	E	69		118		**165**
	21	F	70		119		**166**
	22	G	71		120		**167**
	23	H	72		121		**168**
	24	I	73		122		**169**
	25	J	74		123		**170**
	26	K	75		124		**171**
	27	L	76		125		**172**
RED	**28**	M	77		126		**173**
RIGHT	29	N	78		127		**174**
GREEN	30	O	79		128		**175**
BLUE	31	P	80	ORANGE	129		**176**
SPACE	32	Q	81		130		**177**
!	33	R	82		131		**178**
"	34	S	83		132		**179**
#	35	T	84	f1	133		**180**
$	36	U	85	f3	134		**181**
%	37	V	86	f5	135		**182**
&	38	W	87	f7	136		**183**
.	**39**	X	88	f2	137		**184**
(	**40**	Y	89	f4	138		**185**
)	**41**	Z	90	f6	139		**186**
*	**42**		91	f8	140		**187**
+	**43**		92	SHFT-RTN	141		**188**
,	**44**		93		142		**189**
-	45		94		143		**190**
.	46		95	BLACK	144		**191**
/	47		96				
0	48		97	CODES	192-223	SAME AS	96-127
				CODES	224-254	SAME AS	160-190
				CODE	255	SAME AS	126

Note: graphics characters are not shown in the table above.

APPENDIX G – BASIC LIGHTNING MEMORY MAP

$0000	to	$00FF	zero page
$0100	to	$01FF	stack
$0200	to	$03FF	system variables
$0400	to	$04FF	storage for pseudo-variables
$0500	to	$05FF	wrap buffer no. 1
$0600	to	$06FF	wrap buffer no. 2
$0700	to	$07FF	sprite pointers - low bytes
$0800	to	$57FF	BASIC LIGHTNING extension
$5800	to	$9FFF	*BASIC LIGHTNING program text
$A000	to	$BFFF	*sprite storage
$A000	to	$BFFF	BASIC ROM
$C000	to	$C7FF	character set and/or hardware sprites
$C800	to	$CBFF	text screen
$CC00	to	$CFFF	hi-res primary attributes
$D000	to	$D7FF	I/O devices
$D800	to	$DBFF	text colour memory/hi-res secondary attributes
$DC00	to	$DFFF	I/O devices
$E000	to	$FFFF	hi-res pixel data
$E000	to	$FFFF	KERNAL ROM

* These can be altered using the RESERVE command.

APPENDIX H – ERROR MESSAGES

BAD UNTIL	:	An UNTIL command was found without a corresponding REPEAT.
BAD WEND	:	A WEND command was found without a corresponding WHILE.
BAD EXIT	:	An EXIT command was found which is not inside a loop.
BAD LOCAL	:	A LOCAL command was found which is not at the top of a procedure or function definition.
BAD PROCEND	:	A PROCEND was found when a procedure has not been called.
NO ROOM	:	There is insufficient sprite space left.
CORRUPTED SPRITE	:	The sprite data in the table has been corrupted.
REDEF'D SPRITE	:	An attempt has been made to define a sprite which already exists.
NO SUCH SPRITE	:	An attempt was made to reference a sprite which does not exist.
DELETE SPRITE ZERO	:	Sprite zero is the hi-res screen and therefore it cannot be deleted.
OUT OF RANGE	:	An attempt was made to reference pixel data off the edge of a sprite.

APPENDIX I—THE ARCADE SPRITE LIBRARY

THE ARCADE SPRITES SAVED AS DEMO1

SPRITE NO.	DESCRIPTION	INK 3	INK 4	HGT	WID
1	PACMAN GHOST	7	0	2	2
2	MAN	1	0	2	2
3	LIGHT TANK	5	0	2	4
4	BI-PLANE	2	15	2	4
5	HELICOPTER #1	5	0	2	4
6	SPACESHIP #1	2	15	2	4
7	SPACE TANK	6	15	2	4
8	ASTROID SPACESHIP #1	1	0	2	2
9	ASTROID SPACESHIP #2	3	0	2	2
10	INVADER #1	1	0	2	2
11	INVADER #2	7	0	2	2
12	SPACESHIP #2	0	15	2	4
13	SPACESHIP #3	0	15	2	4
14	BUG EYED MONSTER #1	3	0	2	2
15	WOMAN	7	0	2	2
16	SPACESHIP #4	5	1	3	5
17	HEAVY TANK	15	0	3	6
18	INVADER #3	5	0	2	2
19	SPITFIRE	0	3	2	5
20	FLYING SAUCER #1	1	0	3	5
21	FACE	1	4	2	2
22	SPACESHIP #5	0	3	2	5
23	SPACESHIP #6	3	0	2	5
24	SPACESHIP #7	1	0	2	6
25	METEOR	5	0	2	5
26	FLYING SAUCER #2	0	5	2	4
27	BUG EYED MONSTER	6	3	2	4
28	BUG EYED MONSTER	11	7	4	3
29	SPACESHIP #8	0	1	2	5
30	SPACESHIP #9	0	1	2	6
31	INVADER #4	3	0	2	2
32	OCEAN LINER	6	3	2	7
33	TRI-PLANE	2	3	2	4
34	BUILDING			3	3
35	BULL DOZER	0	3	2	5
36	SPACESHIP #10	3	2	0	5
37	DUCK	7	0	3	4
38	MEDIUM TANK	0	3	3	6
39	HELICOPTER #2	6	3	2	7
40	HOVERCRAFT	2	1	3	5
41	SUBMARINE	11	3	2	8
42	TANK DESTROYER	11	7	3	6
43	SPACESHIP #11	11	3	3	5
44	RABBIT	1	4	3	2
45	FROG	5	0	3	3
46	CROCODILE	11	3	3	6
47	CRAB	5	1	3	5
48	SPACESHIP #12	6	1	2	6
49	SPACESHIP #13	3	0	2	6
50	SPACESHIP #14	5	0	2	6

SPRITES 100 TO 109 USED AS A DEMONSTRATION OF ANIMATION.

APPENDIX J – CASSETTE STORAGE

TAPE 1

SIDE A

1. BASIC LIGHTNING	"BL"	
2. SPRITE GENERATOR	"SPTGEN"	

SIDE B

1. BASIC LIGHTNING	"BL"	(TURBO LOADING)
2. SPRITE GENERATOR	"SPTGEN"	(TURBO LOADING)

TAPE 2

SIDE A

1. ARCADE SPRITES	"DEMO1"
2. DEMO SPRITES	"DEMO2"

SIDE B

1. BASIC DEMO	"DEMO"
2. DEMO SPRITES	"DEMO2"

APPENDIX K – RUNNING THE DEMO

1. Load BASIC LIGHTNING from Tape 1 using SHIFT-RUN/STOP.
2. Load BASIC DEMO Side B using SHIFT-RUN/STOP.
3. To re-run just type RUN.

INTERRUPT-DRIVEN COMMANDS: PLAY, RPLAY, TRACK AND MOVE

These four commands allow you to play tunes or move hardware sprites under interrupt, the necessary data being taken from a software sprite.

PLAY AND RPLAY

These are used to play tunes under interrupt and both have three parameters- SPN,COL and ROW. Unlike normal usage, COL and ROW are both sprite numbers. Sprite SPN is the software sprite with the data for voice 1, sprite COL contains voice 2's data, and sprite ROW contains voice 3's data.

For example, if you wanted the data in sprite 3 to be played by voice 1, sprite 9 by voice 2 and sprite 15 by voice 3, you would use:

```
PLAY  3,9,15
```

It is also possible to keep a voice silent (so that it could be used to generate sound effects with the MUSIC command) by specifying sprite zero:

PLAY 3,0,15 would keep voice 2 silent.

Using this method it is possible to silence all voices:

```
PLAY  0,0,0
```

If you use the PLAY command, the voices will remain silent after the last byte of data has been read from the sprite. If however you use RPLAY instead of PLAY, then the tune is repeated indefinately, or until a PLAY 0,0,0 is executed.

FORMAT FOR STORING TUNES IN SPRITES:

Each note inside the sprite takes up four bytes - two bytes for frequency in the usual low byte-high byte order followed by the length of the note in 60ths of a second, and the time (again in 60ths of a second) taken between releasing the present note and striking the next one.

The data is contained in the pixel data part of the sprite only - the attribute part is not used. Since each character block uses up 8 bytes, 2 notes will fit into one character block. Thus, if the tune contained 30 notes, a 15x1 sprite could be used to store the data.

There are two ways of getting the data into the sprite - it could either be put there using the 'N' option in the sprite generator program or it can be POKEd in directly from data statements:

```
250  SPRITE 1,15,1
260  FOR i=DFA(1) TO AFA(1)-1 STEP 4
270    READ j
280    DOKE i,j
290    READ j
300    POKE i+2,j
310    READ j
320    POKE i+3,j
330  NEXT i
```

In this case, the data should be arranged in groups of three - frequency, length and gap between notes:

```
400  DATA 6407,20,10
410  DATA 12814,40,20
430  DATA 6407,20,10
440  ....
450
```

For a table of frequency values, see appendix M.

Before using PLAY or RPLAY, the wave form, volume and envelope must be set up as for the MUSIC command.

TRACK

TRACK is similar to PLAY in that it reads data from a software sprite under interrupt - however, in this case the data is used to move a hardware sprite around the screen. The data is held in groups of two bytes inside the sprite; one byte x-offset followed by the byte y-offset. These offsets are signed (-128 to 127) and are added to the sprites x and y positions on the screen every 50th of a second.

Note: If you want to enter negative numbers into the sprite using the 'N' command if its sprite generator then add 256 to your negative number ie:

```
Instead of entering -1   enter 256 + - 1   = 255
Instead of entering -8   enter 256 + - 8   = 248
Instead of entering -120 enter 256 + - 120 = 136
Instead of entering -128 enter 256 + - 128 = 128
```

TRACK SPN,SPN2

Track has two parameters, SPN is the software sprite containing the data and SPN2 is the hardware sprite to be moved (0 to 7). As with PLAY, the data can be put in the software sprite using the sprite generator, or it can be POKEd in directly:

```
250  SPRITE 1,15,1
260  FOR i = DFA(1) TO AFA(1)-1
270    READ j
280    POKE i,j -(j<0)*256
290  NEXT i
'
'
'
'
400  DATA 1,0
410  DATA 1,-1
```

Note that line 280 allows negative numbers to be POKEd into memory. Since each character block contains data for 4 interrupts, and interrupts occur 50 times a second, the above 15x1 sprite would contain enough data for 1.2 second of animation.

TRACK deals with offsets which are added to the current position on the screen, so it can be used to carry out an animation from any starting position. Before using TRACK, the sprite must be turned on and positioned in the normal way.

MOVE SPN,COL,ROW is similar to TRACK - however, it only allows movement by a constant amount, and data is not read from a sprite. SPN is the hardware sprite, COL is the x-offset and ROW is the y-offset.

If a hardware sprite moves off the screen, it is automatically halted and turned off. If you remove a sprite from the screen using .OFF any animation that was taking place will stop.

It is also possible to use .XPOS(n) and .YPOS(n) in expressions to read a sprites position:

```
IF .HIT(3) THEN PROCexplode( .XPOS(3),.YPOS(3) )
```

APPENDIX M

MUSIC NOTE VALUES

This appendix contains a complete list of Note#, actual note, and the values to be POKEd into the HI FREQ and LOW FREQ registers of the sound chip to produce the indicated note.

MUSIC NOTE VALUES OSCILLATOR FREQ

Note	Octave	Decimal	Hi	Low
0	C-0	268	1	12
1	C#-0	284	1	28
2	D-0	301	1	45
3	D#-0	318	1	62
4	E-0	337	1	81
5	F-0	358	1	102
6	F#-0	379	1	123
7	G-0	401	1	145
8	G#-0	425	1	169
9	A-0	451	1	195
10	A#-0	477	1	221
11	B-0	506	1	250
16	C-1	536	2	24
17	C#-1	568	2	56
18	D-1	602	2	90
19	D#-1	637	2	125
20	E-1	675	2	163
21	F-1	716	2	204
22	F#-1	758	2	246
23	G-1	803	3	35
24	G#-1	851	3	83
25	A-1	902	3	134
26	A#-1	955	3	187
27	B-1	1012	3	244
32	C-2	1072	4	48
33	C#-2	1136	4	112
34	D-2	1204	4	180
35	D#-2	1275	4	251
36	E-2	1351	5	71
37	F-2	1432	5	152
38	F#-2	1517	5	237
39	G-2	1607	6	71
40	G#-2	1703	6	167
41	A-2	1804	7	12

MUSICAL NOTE		OSCILLATOR FREQ		
Note	Octave	Decimal	Hi	Low
42	A#-2	1911	7	119
43	B-2	2025	7	233
48	C-3	2145	8	97
49	C#-3	2273	8	225
50	D-3	2408	9	104
51	D#-3	2551	9	247
52	E-3	2703	10	143
53	F-3	2864	11	48
54	F#-3	3034	11	218
55	G-3	3215	12	143
56	G#-3	3406	13	78
57	A-3	3608	14	24
58	A#-3	3823	14	239
59	B-3	4050	15	210
64	C-4	4291	16	195
65	C#-4	4547	17	195
66	D-4	4817	18	209
67	D#-4	5103	19	239
68	E-4	5407	21	31
69	F-4	5728	22	96
70	F#-4	6069	23	181
71	G-4	6430	25	30
72	G#-4	6812	26	156
73	A-4	7217	28	49
74	A#-4	7647	29	223
75	B-4	8101	31	165
80	C-5	8583	33	135
81	C#-5	9094	35	134
82	D-5	9634	37	162
83	D#-5	10207	39	223
84	E-5	10814	42	62
85	F-5	11457	44	193
86	F#-5	12139	47	107
87	G-5	12860	50	60
88	G#-5	13625	53	57
89	A-5	14435	56	99
90	A#-5	15294	59	190
91	B-5	16203	63	75
96	C-6	17167	67	15
97	C#-6	18188	71	12
98	D-6	19269	75	69
99	D#-6	20415	79	191
100	E-6	21629	84	125
101	F-6	22915	89	131
102	F#-6	24278	94	214
103	G-6	25721	100	121
104	G#-6	27251	106	115
105	A-6	28871	112	199
106	A#-6	30588	119	124
107	B-6	32407	126	151
112	C-7	34334	134	30
113	C#-7	36376	142	24
114	D-7	38539	150	139
115	D#-7	40830	159	126
116	E-7	43258	168	250
117	F-7	45830	179	6
118	F#-7	48556	189	172

MUSICAL NOTE		OSCILLATOR FREQ		
Note	Octave	Decimal	Hi	Low
119	**G-7**	51443	200	243
120	**G#-7**	54502	212	230
121	**A-7**	57743	225	143
122	**A#-7**	61176	238	248
123	**B-7**	64814	253	46

LOCATION	CONTENTS
54293	Low Cutoff frequency (0-7)
54294	High Cutoff frequency (0-255)
54295	Resonance (bits 4-7) Filter voice 3 (bit 2) Filter voice 2 (bit 1) Filter voice 1 (bit 0)
54296	High pass (bit 6) Bandpass (bit 5) Low pass (bit 4) Volume (bits 0-3)

BASIC LIGHTNING

Create your own, professional arcade-type games on the Commodore 64.

Although there are a number of extended BASIC's available which aim to make better use of the 64's powerful hardware, Basic Lightning offers you more. Basic Lightning brings any professional, arcade-type game, or any game your imagination can create within your reach with these extensive and unique features:

▶ Basic Lightning is totally dedicated to writing very fast, commercial quality video games.

▶ Basic Lightning is multi-tasking so one foreground and four background programs can be run simultaneously. Ideal for those scrolling backdrops.

▶ More than 100 graphics and sound commands, more than 20 structured programming commands, including procedures with true parameter passing, plus general purpose utility commands and improved editing facilities, put this product in a class of its own.

▶ Up to 255 software sprites (up to 6 screens wide!) can be scrolled, spun, enlarged, contracted, mirrored and rotated. The Commodore's own hardware sprites are fully supported.

▶ Basic Lightning is supplied with a full sprite development package which comes complete with a library of predefined arcade characters.

▶ A comprehensive manual is also included.

BASIC LIGHTNING IS JUST ONE IN A SERIES OF LIGHTNING PACKAGES FROM OASIS SOFTWARE.