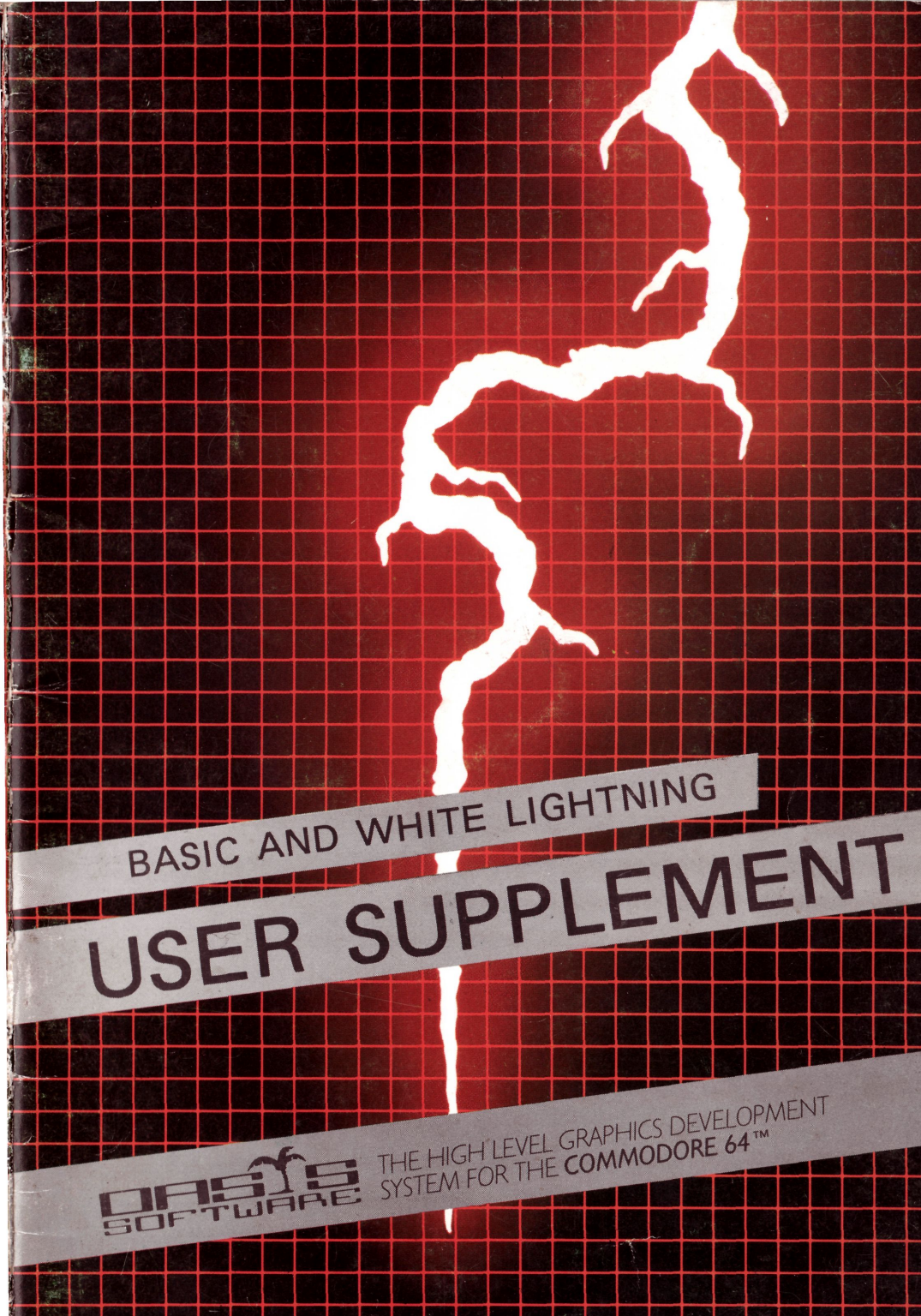
BASIC AND WHITE LIGHTNING
USER SUPPLEMENT
OASIS SOFTWARE
THE HIGH LEVEL GRAPHICS DEVELOPMENT SYSTEM FOR THE COMMODORE 64™

CONTENTS

COMMODORE 64 USER SUPPLEMENT

BASIC AND WHITE LIGHTNING

This manual is supplied as a supplement to the Basic and White Lightning manuals. It is intended to give further explanation and examples of the IDEAL graphics routines covered in those manuals.

The Lightning series of products has been designed in such a way that the IDEAL graphics commands used in Basic and White Lightning not only have the same syntax and operation, but also use the same parameters. Therefore most of the IDEAL commands will be explained with the aid of Basic Lightning, using Basic Lightning examples, as this should make it easier for most users to understand.

LIGHTNING PROGRAM OVERVIEW

BASIC LIGHTNING

This is loaded using RUN/STOP SHIFT from tape or LOAD"BL",8,1 from disk. Basic Lightning is required to be on board whenever a Basic Lightning program is run.

WHITE LIGHTNING

This is loaded using RUN/STOP SHIFT from tape or LOAD"WL",8,1 from disk. Once loaded you will be under the complete control of the Forth operating system.

SPRITE GENERATOR PROGRAM

This is a program written in Basic Lightning that has been provided to enable the user to create and edit software sprites. You will need to load Basic Lightning first. The normal speed tape version is loaded using RUN/STOP SHIFT, the disk version is loaded by typing DLOAD"SPTGEN":RUN the turbo version is loaded as an option when loading the turbo version of Basic Lightning. Type Y to the prompt and once loaded type OLD and then RUN.

DEMO1 SPRITES

This is a file of sprites, listed on page 83 of the Basic Lightning manual, that can be used in your programs. To save memory it would be best to WIPE the sprites you do not want.

The sprites are loaded into Basic Lightning by typing RECALL"DEMO1" for tape and DRECALL"DEMO1" for disk.

The sprites are loaded into White Lightning by typing "DEMO1"RECALL for tape and disk.

DEMO2 SPRITES

This is another file of sprites that was used in the Basic Lightning and White Lightning demos. It is loaded in the same way as the DEMO1 sprites.

WHITE LIGHTNING DEMO

This is a ZAPped demo written using White Lightning. It is a self contained program and will run on its own. To load the tape version type RUN/STOP SHIFT and to load the disk version type LOAD"DEMO",8,1 and then RUN.

BASIC LIGHTNING DEMO (not on tape White Lightning)

This is a demo written using Basic Lightning and you will need Basic Lightning to be loaded first.

Type RUN/STOP SHIFT to load the tape version and DLOAD"DEMO",8,1 and RUN for Basic Lightning disk, or DLOAD"BLDEMO",8,1 and RUN for White Lightning disk.

IDEAL

We will now go into detail explaining some of the IDEAL graphics words. These, as stated, will be explained in Basic Lightning terms since this is easiest to follow and experiment with. The operation of these words are identical for White Lightning users. However, users of White Lightning must remember that Forth requires a different method of passing parameters to and from these commands.

GETTING STARTED

The following examples, using Basic Lightning, will require Basic Lightning to be on board as well as some sprites from the DEMO1 sprite file.

Load Basic Lightning by typing

```
RUN/STOP SHIFT (for tape)
LOAD"BL",8,1   (for disk)
```

Now to load the sprites use the word RECALL or DRECALL for disk, type

```
RECALL"DEMO1"  (for tape)
DRECALL"DEMO1" (for disk)
```

We will only need the first 25 sprites, therefore we can loose the rest by typing

```
FOR I=26 TO 50 : WIPE I : NEXT I
FOR I=100 TO 108 : WIPE I : NEXT I
```

You may wish to store the remaining sprites, do so by typing STORE "SPRITES" or DSTORE "SPRITES" for disk users.

THE TEXT AND HIRES SCREENS

Basic and White Lightning support both a Text screen and a Hires screen (two or four colour mode).

The word HIRES is used to go from a text screen to a 'hires' screen.

When a program is 'running' and the word HIRES is encountered the hires screen will be displayed until either the word LORES (show text screen) or the program ends, then the text screen will be displayed.

Since the 'default' screen is the text screen, while typing in programs or entering words directly, if you type HIRES you will get the hires screen displayed for a fraction of a second and since the program (typing HIRES) has ended the text screen will be displayed again.

Hence HIRES can only be used to display the hires screen during the running of a program.

e.g. HIRES : FOR I=1 TO 1000 : NEXT I

To get over the problem of having to run a program to see what you have done on the hires screen, the word WINDOW can be used.

WINDOW enables the user to split the screen between a top hires screen in the range of 1 to 23 rows and the text screen.

Typing WINDOW 10 will display the top 10 rows of the hires screen.

The hires screen is treated as Sprite number 0, therefore if you carry out any scrolling or sprite operations on sprite 0 you will carry out these operations on the screen.

The hires screen (sprite 0) has the demensions of 0 to 39 columns by 0 to 24 rows and 0 to 319 pixels in the X plane and 0 to 199 pixels in the Y plane.

SYSTEM VARIABLES (SPRITE VARIABLES)

All Basic and White Lightning commands need parameters to operate on. Basic and White Lightning share a group of variables which are listed on page 10 of the Basic Lightning manual.

In Basic Lightning they can be used in the normal sense to pass parameters to the Ideal words, e.g. PUTBLK

```
SPN = 1 : COL = 2 : ROW = 3 : PUTBLK
```

or they can be set by just placing the values after the word separated by commas
e.g. PUTBLK 1,2,3

Remember if you type in any word that is given a star in appendix A of the Basic Lightning manual, and do not specify the System Variables, the last values will be used.

PUTTING A SOFTWARE SPRITE ON THE SCREEN

Firstly, clear the hires screen (Sprite 0) using the following lines and set a window:

```
SETATR 0,1,0 : SCLR0,ATR
WINDOW 20
```

The simplest way of putting a software sprite to the screen is to use the word PUT.

There are four versions of PUT these being PUTBLK , PUTOR , PUTXOR and PUTAND .

All these operations take a whole sprite and place it on the screen where the user has specified.

e.g. PUTBLK 1,2,3

This blockputs sprite number 1 and places its top left hand corner at COL position 2, ROW position 3 onto the hires screen.

All these sprite operations do not physically remove a sprite from memory but 'copy' it to the screen, such that, unlike the standard commodore hardware sprites you can have as many of the same or different sprites on the screen at any one time as you like.

Hence, once you have placed a software sprite on the screen it can be said to be dead, or non active. That is to say if you carry out another operation on that sprite you will not see the effect until you re-put that sprite on the screen.

The 3 remaining operations put a sprite to the screen using the logic operations OR, XOR and AND .

For example type PUTXOR 9,10,10 to XOR sprite 9 to the Screen, now type PUTXOR 9,10,10 again, and watch the effect.

GET

The opposite of PUT, i.e. taking data from a sprite and placing it on the screen, altering the screen but leaving the sprite unaffected, is GET. This takes data from the screen, of the dimensions of the sprite and places it into the specified sprite, leaving the screen unaffected but altering the sprite.

The following program puts sprite 2 on the screen and "Gets" that sprite into sprite 1 by reading the data on the screen and then putting sprite 1 on the screen next to sprite 2.

```
10 PUTBLK 2,1,1
20 GETBLK 1,1,1
30 PUTBLK 1,4,1
```

However PUT and GET are limited to the fact that:

1. They only operate to and from the screen (sprite 0), so if you want to copy one sprite into another you have to go via the screen, and

2. There is no way of taking sections of a sprite (these sections of sprites are known as windows) and manipulating them.

MOV

A much more flexible word than PUT is provided, this is known as MOV.

This allows you to operate on a window of data from a sprite (including sprite 0). As before, these operations can be BLK , XOR , OR or AND .

The following program takes random 1 character by 1 character windows out of sprites 3 to 7 and places them on the screen.

```
 5 REM RANMOV
10 HIRES
20 FOR S = 3 TO 7
30 FOR Y = 0 TO 25
40 FOR X = 0 TO 39
50 XP = INT(RND(1)*4)
60 YP = INT(RND(1)*2)
70 MOVBLK S, XP, YP, 1,1,0,X,Y
80 NEXTX : NEXTY : NEXTS
```

X and Y are the column and row positions in Sprite 0, the screen.
XP and YP are the column and row positions in the source sprite.
S is the number of the source sprite.

CPY

CPY is used to 'copy' one sprite into another sprite of the same dimensions. BLK copy as well as XOR , OR and AND are supported, e.g. CPYBLK

In all the above operations PUT , GET , MOV and CPY; the source sprite is always left unaffected while the target sprite is always changed unless an 'OUT of RANGE' error is caused by defining a window larger or outside the dimensions of a sprite, which includes the screen.

Software Sprites are best created using the sprite generator program. However you can create sprites in software using the words ISPRITE or SPRITE and specifying the number, width and height.

e.g. ISPRITE 5,4,3

creates an 'empty' sprite 4 characters wide by 3 characters high. (Remember the number of the sprite must be in the range 1-255 and it must be unique.)

You can also destroy sprites from memory using the word WIPE which 'wipes' a specified sprite from memory.

e.g. WIPE 3 'wipes' sprite 3 from memory

the word RESET will erase all software sprites from memory.

The Word DRAW is used to draw a line in hires from one point in a sprite to another, while the word PLOT is used to set a pixel in a sprite. Both these operations are dependant on the MODE that you are in.

If you type DRAW 0,0,0,319,199 you will draw a hires line from the top left hand pixel to the bottom right hand pixel of the hires screen (sprite 0).

The following program demonstrates the use of the word DRAW by drawing a pattern on the Hires screen.

```
 5 REM STAR BURST
10 WINDOW 23 : SETATR 0,1,0 : SCLR 0, ATR
20 G=INT (RND(1)*30) +4
30 FOR F =0 TO PI * 2 STEP PI/G
40 V=INT (RND(1)*75) +15
50 FOR N =0 TO PI/G STEP PI/100
60 H = SIN(N/ (PI/G)*PI) *V
70 DRAW 0,160,87,160 + (H*SIN (N+F)), 87 + (H*COS(N+F))
80 NEXTN : NEXTF
```

You are, of course, not limited to DRAWing and PLOTing to the screen. All these functions can be carried out on any sprite that exists in memory. The following example fills up a sprite with random PLOTs and then puts the sprite on the screen. Remember to wipe sprite 200 when you have finished.

```
 5 REM RANDOM PLOT
10 SPRITE 200,3,1 : SETATR 0,1,0 : SCRL0,ATR : HIRES
20 SCLR 200, ATR
30 FOR Y = 0 TO 24
40 FOR X = 0 TO 39 STEP 3
50 XP = INT (RND(1)*24)
60 YP = INT (RND(1)*8)
70 PLOT 200,XP,YP
80 PUTBLK 200,X,Y
90 NEXTX : NEXTY
100 GOTO 20
```

Line 10 creates sprite 200, clears the hires screen and goes into hires mode (sprite 3 being 3 wide and 1 high).
Line 20 clears sprite 200.
Lines 30 and 40 calculate the position where sprite 200 is going to be put on the screen.
Lines 50 and 60 calculate the random plotting position.
Line 70 plots the pixel in the sprite.
Line 80 puts sprite 200 on the screen.

CIRCLES

Basic Lightning supports a command known as POLY which enables a polygon of user definable dimensions and sides to be created.

If you state a polygon to have a large number of sides, say more than about 40, a circle will be generated.

The following example uses the polygon command to generate a tunnel of circles in hires multi-colour mode and then using SETA (see later) change the Ink colours to give animation.

```
  5 REM TUNNEL
 10 MULTI: S4COL: HIRES: ATT20N
 20 SETATR BLUE, WHITE, CYAN : HBORDER2: HPAPER2: SCLR0, ATR
 30 FOR R=1 TO 60 STEP9
 40 MODE1: W=R: GOSUB100: W=R+1: GOSUB100: W=R+2: GOSUB100
 50 MODE2: W=R+3: GOSUB100: W=R+4: GOSUB100: W=R+5: GOSUB100
 60 MODE3: W=R+6: GOSUB100: W=R+7: GOSUB100: W=R+8: GOSUB100
 70 NEXTR
 80 SETATR 7,2,5: GOSUB200
 81 SETATR 2,5,7: GOSUB200
 82 SETATR 5,7,2: GOSUB200
 83 GOTO80
100 POLY0,160,100,W,W,80,0:RETURN
200 SETA 0,11,4,18,17,ATR:FORN=1 TO 20: NEXTN: RETURN
```

Type MONO: S2COL when finished

Remember when you are plotting, drawing or using the word Polygon, or for that fact any other operation on a sprite other than the screen. The maximum values depend on the size of the sprite.

e.g. sprite 3 is 2 characters high and 4 characters wide which means you have in effect an area of 32 pixels (0 to 31) in the x plane by 16 pixels (0 to 15) in the y plane to work with. So if you, for example, try and draw outside this range you will get an error message.

SCROLL WITH WRAP

With Basic and White Lightning you have the facility to scroll a window of user definable dimensions in a sprite by 1,2 and 8 pixels left and right with or without 'wrap' around.

Type WINDOW 20 for the following examples.
If you were to type WRL1 0,1,2,3,4 you would scroll a window in sprite 0 (remember the Hires screen) at COL position 1, ROW position 2 of WIDth 3 and HiGhT 4 characters by 1 pixel left with wrap.

This has the effect of scrolling any data (not attributes) on the hires screen by one pixel.

To scroll that window 1000 x 1 pixels you can use the word RPT

```
RPT1000, WRL10,1,2,3,4
```

RPT executes WRL1 1000 times.

If you were to type WRR1 12,0,0,4,2 you would scroll sprite 12 by 1 pixel with wrap in memory. The following program will scroll sprite 12 and then place that sprite on the screen. Looping around to repeat the sequence

```
10 WRL2 12,0,0,4,2
20 PUTBLK 12,1,1
30 GOTO 10
```

You do not have to scroll the whole sprite. You can scroll a section or window within, it if you wish. The following program scrolls the top half of Sprite 2 by 2 pixels left and then puts the sprite on the screen. The program loops around

Contained in the 'DEMO2' sprites, is a Sprite which is 60 characters wide. The following example demonstrates one method of scrolling a landscape which is wider than the physical dimensions of the screen. This example is the simplest, yet slowest method of achieving the effect. (You will have to recall the 'DEMO2' sprites for this example)

```
10 WINDOW 2
20 MOVBLK 35,0,0,40,2,0,0,0
30 WRRl 35,0,0,60,2
40 GOTO 20
```

Line 20 takes a window 40 characters wide at ROW, COL position 0 from sprite 35 and places it on the screen (Sprite 0) at ROW, COL position 0.
Line 30 Scrolls the whole of sprite 35 by 1 pixel with wrap to the right.

VERTICAL SCROLLS WITH WRAP

Also provided are vertical scrolls with or without wrap. They work in exactly the same way as the horizontal scrolls, except you are not limited to the 1,2 or 8 pixel increments.

An extra parameter is required which states the number of pixels to be scrolled, a positive number specifies up and a negative number down.

The following program scrolls sprite 17 in memory up by 1 pixel with wrap and places it on the screen looping around.

```
10 WRAP 17,0,0,6,3,1
20 PUTBLK 17,20,5
30 GOTO 10
```

Line 10 scrolls vertically with wrap, a window with a top left hand corner at ROW and COL 0 in sprite 17 of dimensions height 3, width 6 by 1 pixel.

DIAGONAL SCROLLS

By executing a vertical scroll and a horizontal scroll a diagonal scroll can be produced

e.g.
```
10 WRAP 17,0,0,6,3,-2
20 WRL2 17,0,0,6,3
30 PUTBLK 17,20,5
40 GOTO 10
```

Note line 10 scrolls down 2 pixels (-2)

The following routine will place sprite 17 diagonally down the screen, using PUTBLK, twice.

```
 5 WINDON 23: X=0 : FOR Y=0 TO 21
20 PUTBLK 17,X,Y : PUTBLK 17,X+6,Y
30 X=X+1 : NEXTY
```

Not very impressive you may say, but now insert line 10 which will diagonally scroll the sprite in memory in the opposite direction from the movement of the sprite on the screen.

```
10 WRAP 17,0,0,6,3,8 : WRL8 17,0,0,6,3
```

giving a 'diagonal wallpaper' effect.

ATTRIBUTES

All software sprites are built of two distinct groups of data. The first is the pixel data which in 2 colour mode has no bearing on the colours of the sprite, and the second is the attribute data which is all the colour data of the sprite. Pixel colours are referred to as INK colours and non pixel or background colours are referred to as PAPER colour.

ATTON and ATTOFF

These two functions switch ON or OFF a toggle switch which states whether the attribute data of sprites is used.

For example, clear the hires screen to Yellow ink and Black paper.

```
SETATR 0,7,0 : SCLRO,ATR : WINDOW 10
```

Now type ATTOFF and put sprite 13 on the screen by typing PUTBLK 13,0,0. A yellow space ship has appeared on the screen. Yellow is not the colour of the sprite, but the colour of the screen.
Now type ATTON and PUTBLK 13,0,0, to see the true colours of the sprite.

SETATR

In 2 colour or mono mode you are dealing with two colours (Ink and Paper) and in 4 colour or multi colour mode you are using four colours (3 Inks and 1 Paper). A variable known as ATR is used to hold these values. You set ATR using the word SETATR

```
e.g. SETATR 3,1,2
  or SETATR CYAN, WHITE, RED
```

Both forms are correct as colours can be typed as numbers from 0 to 15, or their colour spelt out.

SETATR 3,1,2 in 2 colour mono mode will set the Ink (Pixel or foreground) colour to White and the Paper (Non-pixel or background) colour to RED. CYAN is ignored in this mode.

SETATR 3,1,2, in 4 colour multi mode will set the 3 available Ink colours to 1,2 and 3. The paper colour is defined using a new word HPAPER. e.g. HPAPER 0 or HPAPER BLACK

To use all 4 colours in the multi colour mode you must enable them by typing ATT2ON.

Other words being HBORDER which sets the hiresolution border colour, e.g. HBORDER 2 or HBORDER RED (This border colour will only be seen when a hires program is running).

INK sets the ink colour of the Text e.g INK GREEN
TPAPER sets the paper colour of the Text screen e.g TPAPER 6
TBORDER sets the Text screen border colour e.g TBORDER 7

The Attribute variable ATR is used in such words as SCLR . SCLR clears a sprite of all Pixel data and sets its Attributes, e.g SCLR0,ATR will clear the Hires screen (Sprite 0) and set the Attributes to those contained in ATR .

ATTGET

This word gets the values of Attributes contained in a Sprite and stores it in the Attribute variable ATR .

SETA

This is the opposite to ATTGET, where by a window in a Sprite has its Attributes set to those stored in ATR. But unlike WCLR, which clears the Pixel data, SETA leaves the Pixel data unaffected.

The following program demonstrates this by placing the contents of ATR in a 1 by 4 window randomly selected in the hires screen.

```
 5    REM SETA EXAMPLE
10    SETATR 0,0,1: SCLR0, ATR: WINDOW5
20    STRPLOT 0,1,1, "********************"
30    STRPLOT 0,1,2, "* BASIC AND WHITE *"
40    STRPLOT 0,1,3, "*   LIGHTNING     *"
50    STRPLOT 0,1,4, "********************"
60    FOR I = 2 to 15  :  SETATR 0,I,1
65    X = INT (RND(1)*40)
70    SETA 0,X,1,1,4, ATR
80    NEXT 1
90    GOTO 60
```

Line 10 sets up the original value of ATR and clears the screen using SCLR.
Lines 20 to 50 put text on the screen.
Line 60 selects the ink colour by going through a loop 2 to 15. This value is stored in ATR.
Line 65 selects a random column position.
Line 70 sets the Attributes of a 1 by 4 window on the screen.

MULTI COLOUR MODE

Basic Lightning allows you to use the 4 colour hires graphics facility of the Commodore 64.

To set the hardware in multi-colour mode, you type MULTI . You can then enable the extra Attributes by typing ATT2ON .

In the normal 2 colour mode typing SETATR 0,1,2 would set the INK or Pixel colour to 1 (White) and the Paper or Non-pixel colour to 2 (RED). The result, then of typing SCLR0,ATR would be to set up a RED screen with White Pixels.

However, in multi or 4 colour mode the colours are built up by the four possible combinations of a horizontal pair of Pixels (0,0, 0,1, 1,0, 1,1).

If in multi colour mode you were to type SETATR 3,1,2, you would set the 3 Pixel colours to Cyan, White and Red respectively. But to set the paper or no Pixel per pair colour you would have to use the word HPAPER. For example, typing HPAPER 4 would set the paper colour to purple.

Now, if after typing MULTI and ATT2ON you start plotting or drawing, the colour of the line will be dependant on which Pixels have been set in the respective Pixel pairs, thus you have no way of controlling the colours of the lines.
For example, type in the following program for plotting random lines. As the screen fills up it becomes predominantly yellow in colour as all the pixels become set.

```
 5   MULTI : S2COL
10   SETATR 7,2,3 : SCLR0,ATR : ATT2ON : HPAPER0
20   WINDOW 20
30   LY=INT (RND(1)*199)
31   LX=INT (RND(1)*319)
40   RY=INT (RND(1)*199)
41   RX=INT (RND(1)*319)
50   DRAW 0, LX,LY,RX,RY
60   GOTO 30
```

To gain software control of the colour of lines appearing in multi colour mode you must now type S4COL .

The word mode is used to control the way in which Plotting and Drawing operate to the screen.

In 2 colour mode (MONO, S2COL), modes 0 and 1 will cause pixels to be set in the paper colour, Modes 2 and 3 will cause pixels to be set in the Ink colour, Mode 4 will invert the pixels.

In 4 colour mode (MULTI, S4COL) Mode 1 causes plotting in Ink 1, Mode 2 in Ink 2, Mode 3 in Ink 3, and Mode 0 plots in the paper colour. Mode 4 inverts 0 and 1, and 2 and 3.

By making slight changes to the previous program, setting S4COL and using Mode to control the colours a completely different effect can be obtained.

```
 5   MULTI : S4COL
10   SETATR 7,2,3 : SCLR0,ATR : ATT2ON : HPAPER0
20   WINDOW 20
30   LY = INT(RND(1)*199)
31   LX = INT(RND(1)*319)
35   MODE INT(RND(1)*4)
40   RY = INT(RND(1)*199)
41   RX = INT(RND(1)*319)
50   DRAW0, LX,LY,RX,RY
60   GOTO 30
```

The following listing is a multi-colour version of the RANDOM PLOT program.

```
 5   RANDOM PLOT II
10   SETATR 5,7,0 : SCLR0,ATR : HPAPER 1
20   MULTI : S4COL : ATT2ON
30   SPRITE 200,3,1 : HIRES
40   SETATR INT(RND(1)*15),INT (RND(1)*15),INT(RND(1)*15) : SCLR 200,ATR
60   FOR Y = 0 TO 24
70   FOR X = 0 TO 39 STEP 3
80   MODE INT (RND(1)*5) : XP = INT(RND(1)*24) : YP = INT (RND(1)*8)
90   PLOT 200,XP,YP
100  PUTBLK 200,X,Y
110  NEXT X : NEXT Y
120  GOTO 40
```

Line 10 sets the initial value of the screen.
Line 20 sets up multi colour mode in hardware and software and enables the extra colours.
Line 30 creates Sprite 200 and goes into Hires screen mode.
Line 40 picks random ink colours and clears Sprite 200 to those colours.
Lines 60 and 70 calculate the column and row position for Sprite 200.
Line 80 picks a random plot colour from the 3 inks by selecting a mode and calculates the plot co-ordinate XP and YP.
Line 90 plots the pixel in Sprite 200.
Line 100 puts Sprite 200 on the screen.

Remember to wipe Sprite 200 and go into 2 colour mode when finished, by typing

```
WIPE 200 : MONO : S2COL
```

SOFTWARE SPRITE CRASH DETECTION

Whenever you use such words as PUT or MOV you can detect whether your sprite has been placed on the screen or into another sprite over some existing data (crash detection).

The following program will demonstrate this by moving sprite 1 across the screen from left to right until a collision is detected with the stationary sprite 2.

```
10    SETATR 0,1,0 : SCLR0,ATR : WINDOW 20
20    PUTBLK 2,37,10
30    DTCTON
40    FOR I = 0 TO 39
50    PUTXOR 1,I,10
60    IF CCOL <>-1 THEN STRPLOT 0,12,4,"COLLISION", 1024 : DTCTOFF : STOP
70    FOR N = 1 TO 10 : NEXTN
80    PUTXOR 1,I,10
90    NEXT I
```

Crash detection has to be enabled using the word DTCTON (line 30). After every sprite data movement (line 50) system variables CCOL and CROW are set with either -1 if no crash has occured, or the values of Row and Col if it has.

In the above listing, line 60 checks CCOL to see if the value contained within ever changes fron -1. If it does, a crash has occured and 'collision' is printed on the screen as well as switching off the crash detection using DTCTOFF .

Line 70 is a delay routine to slow the sprite movement, but only for visual reasons.

Line 80 XOR'S sprite 1 over sprite 1, removing it from the screen. This of course will cause crash conditions, but we only scan for a crash after the next PUTXOR in the new COL position.

These commands are used to expand a software Sprite in either the x or y planes into another sprite.

For a successful expansion, the sprite that is expanded into must be of large enough dimensions to accommodate the expanded sprite.

In the following example we will expand sprite 18 in both the x and y planes and place it in sprite 201, via sprite 200.

```
  5   WINDOW 10
 10   SPRITE 200,4,2
 20   XPANDX 18,0,0,2,2,200,0,0
 30   SPRITE 201,4,4
 40   XPANDY 200,0,0,4,2,201,0,0
 50   PUTBLK 18,1,1 : PUTBLK 200,4,1 : PUTBLK 201,9,1
```

Line 10 creates a sprite (200) twice as wide as sprite 18 to accommodate sprite 18 expanded in the x plane.
Line 20 expands a window in sprite 18 which is as large as the whole sprite. Therefore, we expand the data starting from ROW and COL position 0 with a window of HGT and WID 2 characters (the whole sprite) into sprite 200 at ROW and COL position 0 of that sprite.
Line 30 creates a sprite (201) twice as high as sprite 200 to accommodate sprite 200 expanded in the y plane.
Line 40 expands sprite 200 into sprite 201.
Line 50 places the 3 phases of expansion on the screen.

When finished wipe sprites 200 and 201 by typing:

```
WIPE 200 : WIPE 201
```

The following routine takes successive characters from the character set, characters 0 to 767, and expands them from 1 by 1 characters to 8 by 8 characters displaying the result on the screen.

```
  5   REM BIG CHARS
 10   SETATR 0,1,0 : SCLR0,ATR : HIRES
100   SPRITE 100,2,1 : SPRITE 101,2,2 : SPRITE 102,2,4 : SPRITE 103,4,4 :
      SPRITE 104,8,4
101   SPRITE 105,8,8
200   FOR N = 0 TO 767
220   CHAR 100,0,0,(1024 + N)
230   XPANDY 100,0,0,2,1,101,0,0
240   XPANDY 101,0,0,2,2,102,0,0
250   XPANDX 102,0,0,2,4,103,0,0
260   XPANDX 103,0,0,4,4,104,0,0
270   XPANDY 104,0,0,8,4,105,0,0
290   CHAR0,14,6,N : PUTBLK105,16,6 : A$=STR$(N) : STRPLOT 0,25,6,A$,0
300   NEXTN
```

Line 10 clears the screen.
Line 100 sets up the various sprites needed for the expansion.
Line 220 places the original character (double width) in the first sprite.
Lines 230 to 270 expands the sprite.
Line 290 puts it on the screen.

Wipe sprites when finished.

THE KEYBOARD

A special word known as KB is used to scan the keyboard to see if a key or series of keys has been pressed. If the required key has been pressed then a TRUE value is given.

The syntax of KB is KB(n) where n is a number in the range 0 to 63 which corresponds to a particular key (see pages 23/24 of the Basic Lightning manual).

If you wished to write a program to print "HELLO" every time the H key was presses, an 'n' value of 43 would be used.

```
10 IF KB(43)=TRUE THEN PRINT "HELLO"
20 GOTO 10
```

The advantage of KB is that the keyboard can be scanned to see if more than one key has been pressed.

Say you wished to write a program to print "HELLO" every time the 'Q', 'W' and 'E' keys were all pressed together, it would look like this:

```
10 IF KB(55) AND KB(9) AND KB(49)=TRUE THEN PRINT "HELLO"
20 GOTO 10
```

The function KB can also scan to see if a key has not been pressed, using FALSE.

Say you wished to write a program that prints "HELLO" when, and only when, the 'Q' key is pressed and the 'W' is not:

```
10 IF KB(55)=TRUE AND KB(9)=FALSE THEN PRINT "HELLO"
20 GOTO 10
```

THE JOYSTICK FIRE BUTTON

A word FIREn (where n is the number of the control port 1 or 2) is used to scan the fire button on the joystick, giving a true value if it has been pressed. Say you wished to print the word "FIRE" when the fire button on a joystick in control port 2 was pressed, the program would look like this:

```
10 IF FIRE2=TRUE THEN PRINT "FIRE"
20 GOTO 10
```

THE JOYSTICK

Two functions JS1 and JS2 will produce a number in the range 0 to 8 (see diagram on page 24 of the Basic Lightning manual) depending on the state of a joystick in the respective control port.

The following program will print out the values obtained by a joystick in control port 1 using JS1:

```
10 A=JS1 : PRINTA : GOTO 10
```

The following program is a simple drawing program to demonstrate the use of a joystick in port 1:

```
  5 REM JOYSTICK IN PORT 1
  6 REM DRAW PROGRAM
 10 SETATR0,1,0 : SCLR0,ATR : HBORDER6 : HIRES
 20 X=160 : Y=100
 30 A=JS1
 40 CASEA
 50 OF1,2,8 : X=X+1
 60 OF4,5,6 : X=X-1
 70 OF8,7,6 : Y=Y+1
 80 OF4,3,2 : Y=Y-1
 90 CASEND
200 IF X=320 THEN X=1
201 IF X=0 THEN X=319
202 IF Y=0 THEN Y=199
203 IF Y=200 THEN Y=1
205 MODE3
206 IF FIRE1 THEN MODE1
207 PLOT0,X,Y
210 GOTO 30
```

Line 10 sets up the screen.
Line 20 sets up the X and Y values.
Line 30 reads the value of the joystick.
Lines 40 to 90 calculate the new X and Y values.
Lines 200 to 203 check that X and Y lie on the screen.
Line 206 if the fire button is pressed then pixels are removed.
Line 207 plots the pixels.

THE LIGHTPEN

For those who have a lightpen, two functions are provided to read the X and Y positions of a lightpen in control port 1, these being LPX and LPY.

The following program prints out the values of a lightpen:

```
 10 A=LPX
 20 B=LPY
 30 PRINT"X=";A;" Y=";B
 40 GOTO 10
```

The following program is a simple drawing program using the lightpen, note the LPX and LPY values must be adjusted for drawing.

```
  5 REM LIGHTPEN IN PORT 1
  6 REM DRAW PROGRAM
 10 SETATR0,0,1 : SCLR0,ATR : HBORDER6 : HIRES
 20 X=LPX : X=(X-25)*2
 21 Y=LPY : Y=Y-51
 30 A=LPX : A=(A-25)*2
 31 B=LPY : B=B-51
 40 DRAW0,X,Y,A,B
 50 X=A : Y=B
 60 GOTO 30
```

Line 10 sets up the screen.
Lines 20 and 21 set up the initial values of X and Y.
Lines 30 and 31 read the new value of the joystick.
Line 40 draws a line from the old value of the joystick position (X,Y) to the new position (A,B).
Line 50 makes the new joystick position the old one.

HARDWARE SPRITES

In this example we will make Hardware Sprite number 1 from Software Sprite number 2 of 'DEMO 1' sprites.

All the hardware sprite commands begin with '.' e.g. .XPOS .

Firstly set up a hires window so that we can see what is happening, so type:

```
WINDOW 20
```

We can clear the screen by typing SETATR0,1,0 : SCLR0,ATR and put sprite 2 on it at ROW and COL position 1 by typing PUTBLK 2,1,1

Hardware sprites require their data to be stored somewhere in RAM, usually this will involve POKEing numbers into memory. In Basic Lightning there is 2k of memory set aside for storing the various character sets. Now this area of memory can also be used to store the data for hardware sprites. Since each block of data takes 63 bytes, you can have 32 such sets of data in the character RAM. Using the word SPRCONV pixel data is read from a sprite and stored in one of these 32 blocks.

Now, in our example, we are going to read the data around sprite 7 which is now in sprite 0 (i.e On the screen), and store it in data block 16 of the 32 available. Since the dimensions of a hardware sprite are 24 pixels by 21 pixels and sprite 2 is only 16 by 16 pixels, the window that reads the data is offset so as to put sprite 2 in the middle of the hardware sprite data.

Sprite 2 is at ROW and COL position 1 on the screen, or x position 8 and y position 8 (in pixel co-ordinates). If we now type:

```
SPRCONV 0,3,6,16
```

a window or area of data of 24 by 21 pixels is read from sprite 0 (the screen) at x pixel position 3, y pixel position 6, thus capturing software sprite 2 in the middle, and storing it in the character RAM, data block number 16. We now have to tell hardware sprite 1 where its data is by typing:

```
.SET 1,16
```

To refresh, the program for doing the above would be

```
10    WINDOW 20
20    SETATR0,1,0 : SCLR0,ATR
30    PUTBLK 2,1,1
40    SPRCONV 0,3,6,16,
50    .SET 1,16
```

In two colour mode we can set the colour of hardware sprite 1 by storing the colour using .COL. Say for this example we want the sprite to be GREEN (6) we would type .COL 1,6

The hardware sprite screen is larger than that of the Physical display screen such that the top left hand corner has an x value of 24 and a y value of 50 and the bottom right has an x value of 344 and a y value of 250. To position a hardware sprite you use the words .XPOS and .YPOS, so to position hardware sprite 1 in the top left hand corner of the screen you would type .XPOS1,24 :.YPOS1,50

This means to keep a non-expanded sprite on the display screen .XPOS values should be in a range of 24 to 320, or 24 to 296 fo an 'x' expanded sprite and .YPOS values should be in the range of 50 to 229, or 50 to 208 for a 'y' expanded sprite.

To make sprite 1 appear you must switch it on by typing .ON 1 . You can switch it off by typing .OFF 1 .

The rest of the program to put hardware sprite 1 in the middle of the screen would be:

```
60 .COL 1,6
70 .XPOS 1,160
80 .YPOS 1,120
90 .ON 1
```

By changing the values in .XPOS and .YPOS in FOR-NEXT loops etc. you can move the hardware sprites smoothly around the screen.

```
e.g  100 FOR Y = 50 to 229 STEP 2
     110 FOR X = 24 to 320 STEP 2
     120 .XPOS 1,X : .YPOS 1,Y
     130 NEXTX : NEXTY
```

OTHER HARDWARE SPRITE COMMANDS

.XPAND and .SHRINK

Hardware sprites can be expanded in the x and y directions.
So for hardware sprite 1 you would type .XPANDX 1 to expand in the x plane and .XPANDY 1 to expand in the y plane (you can only expand a hardware sprite once).
To bring an expanded sprite back to its original dimensions you use the word .SHRINK e.g .SHRINKX,1 .SHRINKY,1

.OVER and .UNDER

These commands state whether a hardware sprite will pass over or under pixel data that is on the screen.

Typing .OVER 1 will mean that hardware sprite 1 will pass over data while .UNDER 1 states that the sprite will pass under the pixel data on the screen.

PRIORITY

When two hardware sprites paths cross, the sprite with the higher number will pass behind the sprite with the lower number.

CHARACTER SET MANIPULATION

CHAR allows you to place a character (8x8 pixels) from one of the 3 normal size character sets or a double width character (16x8 pixels) from one of the 3 double width character sets into a sprite.

Each character is identified by a number in the range 0 to 767 for normal size characters and 1024 to 1791 for double width characters (See Basic Lightning manual).

The following program puts the characters from 0 to 767 onto the screen (Sprite 0), printing its numbers in the text screen.

```
 5   WINDOW 22
10   ROW = 0 : COL = 0
20   FOR I = 0 TO 767
30   CHAR 0, COL, ROW,I
40   COL = COL+1
50   PRINT@ 0,0; "CHARACTER ="; I
60   IF COL = 40 THEN COL = 0 : ROW = ROW +1
70   NEXT I
```

Try this after typing UCASE and then LCASE.

PUTCHAR

This allows you to redefine the character set by taking data stored in a sprite (or on the screen) and then placing it in the character set ram.

In the example below we will store the data in sprite 2 in the character set starting with character 65 (A).

```
10   PUTCHR 2,0,0,65
20   PUTCHR 2,1,0,66
30   PUTCHR 2,0,1,67
40   PUTCHR 2,1,1,68
```

However, if you type UCASE or LCASE the character set is refreshed and will have to be re-redefined. Therefore, alternative character sets could be stored in sprites and loaded when required.

Typing UCASE or LCASE will also wipe any hardware sprite data stored.

STRPLOT

This allows you to put strings of characters on the Hires screen. These strings can be inputed directly or stored in a variable.

The following example demonstrates its use.

```
10   A$ = "                    THIS IS A DEMONSTRATION OF 'STRPLOT' IN "
11   B$ = "BASIC LIGHTNING.                                           "
12   A$ = A$ + B$
20   WINDOW 20
30   FOR N = 1 TO (LEN (A$)) - 10
40   B$ = MID (A$,N,10)
50   STRPLOT 0,10,10,B$,1024
60   NEXT N
```

STRUCTURED BASIC (For Basic Lightning Users)

Basic Lightning has the facility to support structured programming. Structured programming leads to more compact and tidier programs produced using languages such as Pascal.

IF-THEN-ELSE

Say, for example, we wanted to write a program to print whether every number between 1 and 200 was divisible by 13 or not, using just IF-THEN. The program could look like this.

```
10    FOR I = 1 TO 200
20    PRINT I;
30    IF I/13 = INT (I/13) THEN PRINT "IS";
40    IF I/13 <> INT (I/13) THEN PRINT "IS NOT";
50    PRINT "DIVISABLE BY 13" : NEXT I
```

Line 30 would be the true line and line 40 the false. However, the 2 lines 30 and 40 can be replaced by one line using the ELSE statement.

```
30   IF I/13 = INT (I/13) THEN PRINT "IS" ELSE PRINT "IS NOT";
```

Note how much more tidy the program has become by replacing lines 30 and 40 by the new line 30.

CIF-CESLE-CEND

However, IF-THEN-ELSE only allows the use of one program line. By using CIF-CESLE-CEND conditional programs can be spread over many lines.

```
10    FOR I = 1 TO 200
20    CIF I/13 = INT(I/13)
30    PRINT I; "IS DIVISABLE BY 13"
40    CELSE
50    PRINT I; "IS NOT DIVISABLE BY 13"
60    CEND
70    NEXT I
```

Line 30 contains the 'THEN' statements and line 50 contains the 'ELSE' statements. Remember to end the routine using CEND .

REPEAT UNTIL

Now say, for example, we wished to set the program in an indefinate loop waiting for a string of characters, "FRED" to be entered, the program could look like this.

```
10    INPUT A$
20    IF A$ = "FRED" THEN GOTO 40
30    GOTO 10
40    STOP
```

This program can be replaced by the following, removing the need for GOTO

```
10    REPEAT
20    INPUT A$
30    UNTIL A$ = "FRED"
40    STOP
```

Any program contained between REPEAT and UNTIL will be REPEATED until the condition after UNTIL is satisfied.

WHILE-WEND

Say, for example, we wish to write a program that keeps adding a random number in the range 1 to 5 to a variable until that variable is greater than 100, it could look like the following.

```
10    A = 0
20    A = A + RND(1)*5
30    PRINT A
40    IF A < 100 THEN GOTO 20
50    STOP
```

This program could be re-written using WHILE-WEND

```
10    A = 0
20    WHILE A < 100
30    A =A+ RND(1)*5
40    PRINT A
50    WEND
60    STOP
```

Any program contained within WHILE and WEND will be repeated while the condition after WHILE is satisfied.

PROCEDURES

One of the most useful functions of Basic Lightning is the ability to support Procedures and Labels.

Labels are used to give a procedure a name as it can be called using the word PROC . Every procedure must begin with the word LABEL and end with the word PROCEND

```
e.g  10    LABEL FRED
     20    PRINT "HELLO FRED"
     30    PROCEND
```

Now every time you type PROC FRED, the procedure labelled Fred will be executed, returning after completion.

You may say that all these are, are fancy sub-routines that could have been called using the standard GOSUB. However, Procedures allow the use of local variables, which are independant from the rest of the program. These variables have to be declared using the word LOCAL.

The following program demonstrates the use of the word LOCAL by storing a number in a variable A and then going into a procedure, creating a local variable, also A and re-using it.

```
 10    A = 1.23456789
 20    PRINT A
 30    PROCCOUNT
 40    PRINT A
 50    STOP
 60    '    100    LABEL COUNT
110    LOCAL A
120    FOR A = 1 TO 9
130    PRINT A
140    NEXT A
150    PROCEND
```

SPRITE INTERROGATION

The functions DFA, AFA, and AFA2 are used to interrogate software sprites in memory.

DFA is used to find the start address of the pixel data. This is useful since you may wish to store a re-locatable machine code sub-routine in a sprite and you would need to know the entry address.

AFA and AFA2 find the start address of the attributes of the sprites.

When one of these functions is executed WID and HGT are loaded with the dimensions of the sprite under test.

The following program demonstrates the use of AFA, AFA2 and DFA by scanning through sprite memory and printing out details on all sprites that exist.

```
 10    REM SCAN MEMORY FOR SPRITES
 20    FOR N = 1 TO 255 : PRINT " CLEAR SCREEN "
 30    IF N = 255 THEN PRINT "SEARCH COMPLETE." : STOP
 40    A = DFA(N) : B = AFA(N) : C = AFA2(N)
 50    IF A<1 THEN NEXT N
 60    PRINT "SPRITE NUMBER =" ; N
 70    PRINT "WID =" ; WID;"  "; "HGT ="; HGT: PRINT
 80    PRINT "                    PIXEL DATA ADDRESS ="; A
 90    PRINT "     PRIMARY ATTRIBUTE ADDRESS ="; B
100    PRINT "SECONDARY ATTRIBUTE ADDRESS ="; C
110    FOR T = 1 TO 2000: NEXT T
120    NEXT N
```

THE SPRITE GENERATOR PROGRAM (SPTGEN)

Users of the early version of the Sprite Generator Program have the RUN/STOP key disabled so they cannot break into the program.

If you load in the Sprite Generator program, using LOAD"SPTGEN", remove the word DISABLE from line 2 and then resave the program SAVE"SPTGEN2" you will have a version in which you can break into once you have finished with the program. Typing NEW will clear the memory leaving a fresh Basic Lightning.

LISTING BASIC LIGHTNING PROGRAMS TO PRINTER

Before listing programs in Basic Lightning to a printer, always make sure you are in upper case by typing UCASE because if you are in lower case, LCASE, the keywords will not be printed.

'PRINT AT'

You can position the cursor for printing using '@'
e.g. PRINT@ X,Y

So to print the cursor at COL position 10, Row position 5 you would type:

```
PRINT@ 10,5
```

REM

Basic Lightning supports a second REM command ' (Shift 7) which is quicker to type and much neater. It is used in exactly the same manner as REM.

```
e.g.    510 PROCEND
        511 '
        512 ' THIS PROCEDURE FIRES BOMB
        520 LABEL FIRE
```

You write forth programs in a series of colon definitions building a lattice of definitions which are all eventually called up from one definition.

Colon definitions are in the form : name definition ;. Every definition must begin with : and end with ; and be given a name or label.

e.g : FRED ."FRED IS HERE" ;

You can give a definition a name that has already been used, this means that only the latest version of the definition with that name will be used, printing a warning error message MSG #4.

To run or execute a defintion you simply type the name of the definition and hit RETURN e.g FRED

SOURCE CODE

These definitions can be said to be the source code and can be entered in two ways.

You can enter colon definitions directly by typing for example : JOHN ." JOHN IS NOT HERE" ; This is a very quick and easy way of doing things but has several drawbacks.

Once you have typed in your colon definition and hit RETURN, if the definition is no longer on the screen and you wish to change it, you will have to type in the whole definition again.

Secondly, the colon definition must be less than two lines long.

By far the best way of creating forth programs is to use the screens. You must first, before using a screen, clear it, e.g using screen 1 type 1 CLEAR Now you can list it by typing 1 LIST

You can now insert your definition by using the editor. Say you want to use line 0, type 0 EDIT

You do not have to complete a definition on line one, i.e : and ; do not have to be on the same line.

You can now, of course, edit the definitions using the normal Commodore editing functions.

Once your definitions have been typed in it is best to save the screens. This is done automatically for disk users but tape uses should use SCRSAVE (see page 5 of the White Lightning manual).

Screens are 1k sections of memory where source is stored. On the disk based system, these screens are stored on disk and you have access to screens 1 to 88. However, the tape based version of White Lightning has to have a 'RAM based' disk system whereby the screens are stored in RAM, giving you access to screens 1 to 12. The source code stored in these screens is not the program that is actually run, the actual code that executes is stored in the Forth Dictionary.

When you type in definitions directly or LOAD a screen, e.g 1 LOAD SOURCE code is compiled into OBJECT code (machine code) and stored in the dictionary. It is this code that executes and runs. You can see a list of all the definitions in the dictionary by typing VLIST , this will include all of your definitions.

Since the program is stored in the dictionary once source code has been compiled from one screen, that screen can be used again for more source.

Once your program has been completed and is contained in the dictionary and sprites are in memory you can use the command ZAP to produce a stand alone package that contains your program, your sprites, the ideal graphics routines and a Forth runtime package.

DISK AND TAPE

At the present time Disk White Lightning can only save and load source screens, sprites and zapped programs to and from disk. While, likewise tape users are restricted to tape I/O operations. Basic Lightning supports both disk and tape; however, disk and tape versions of White Lightning are different products, especially in the way source screens are stored but they do support two common 'flags' that point to the storage device. If these flags are set to 1 the device is the data-recorder while if the flag is 8 the device is the disk drive.

The addresses of these two flags are:

12588 and 17568

So if you were to type 8 12588 C! 8 17568 C! Sprites could be stored and recalled from disk. Source screens could be SCRSAVED and BLKLOADED, not flushed to or loaded from disk and your finished game zapped to disk.

LISTING SOURCE SCREENS TO A PRINTER

The following routine has been written to enable source screens to be listed to a standard Commodore printer.

```
 0    (PRINTER WORDS)
 1    FORTH DEFINITIONS HEX
 2    CREATE PROUT
 3    0286 , 3898 , 1AA0 , FD71 ,
 4    (CHANGE 04A2 ON LINE 5 TO 06A2 IF USING 1520)
 5    FD91 , 06A9 , 04A2 , 00A0 ,
 6    BA20 , A9FF , 2000 , FFBD ,
 7    C020 , A2FF , 2006 , FFC9 ,
 8    02A6 , 00B5 , 7F29 , D220 ,
 9    20FF , FFCC , 06A9 , C320 ,
10    A6FF , 4C02 , 13EE , SMUDGE
11
12    : PRT-ON ' PROUT ' EMIT CFA ! ;
13    : PRT-OFF 25F1 ' EMIT CFA ! ;
14    CACA 261F ! 0DA9 2621 ! 0095 2623 !
15    6C 2625 C! 14D7 2626
```

To enable the printer type PRT-ON and to disable using PRT-OFF

e.g PRT-ON 1 LIST 2 LIST PRT-OFF

SAVING A 'PACKED' PROGRAM

When you save off an extended White Lightning using PACK you save off all the code that makes up White Lightning, which includes your dictionary definitions and the Ideal Graphics routines. Once you have loaded in the code you can execute the program by typing SYS 4608. Users, however, may wish to save off a loader program that will auto run their PACKed programs.

Disk Users - You can use the loader program on page 6 of the White Lightning manual that is used to load and execute disk zapped programs, making sure 'filename' is the same as 'filename' of your PACKed program.

Tape Users - The same routine as for disk users can be used by modifying line 0 to read "filename",1,1 If you save that routine to tape and save the PACKed program after it. Typing RUN/STOP shift will load your packed program and auto run it.

PLAY, RPLAY and TRACK

Although these are not documented in the White Lightning manual they are implemented in the package. Their means of operation is the same as explained in the back of the Basic Lightning manual pages 65 and 66.

PREPARING SPRITES FOR WHITE LIGHTNING EXAMPLES

The following examples are written in Forth for White Lightning. They have equivalent Basic Lightning listings for Basic Lightning users.

In these examples only the first 10 sprites of the 'DEMO1' sprite set will be used for the various examples. It would be best to prepare these sprites using Basic Lightning.

Load in Basic Lightning and then load in the Demo1 sprites by typing RECALL "DEMO1" since we only need to retain sprites 1 to 10 type,

```
FOR N = 11 TO 50 : WIPE N : NEXT N
```

to remove the unwanted sprites 11 to 50. Now save the remaining sprites by typing STORE "SPRITES" (disk users use DRECALL and DSTORE)

GETTING STARTED IN WHITE LIGHTNING

Clear the machine and load in White Lightning, now type:

```
HEX 7C00 LOMEN DECIMAL (tape users only)
```

to reserve room for the sprites, which can now be loaded by typing

```
"SPRITES" RECALL
```

We suggest that the following examples be typed in using the screens and editor. Before using a screen you should clear it, e.g 1 CLEAR , then list it, e.g. 1 LIST , use the editor to display the line, e.g. 0 EDIT , and hit return to flush the line once typed in, remembering you can only have 64 characters per screen line.

Before you run the examples it is wise to save the screens to tape or disk, using SCRSAVE , recalling screens using BLKLOAD

MOVING SOFTWARE SPRITES

The chief problem facing the programmer who wants to move software sprites around the screen is choosing from the numerous schemes available. We will now consider some of these methods, each with its own merits for speed, simplicity, smoothness and storage. We'll begin with the easiest to implement and then work up to some of the more elaborate techniques.

SCREEN SCROLLING UNDER KEYBOARD CONTROL

This is where a sprite on the screen is moved within a window on the screen that does not contain any other sprites or data.

Type in the following colon definitions

```
: SETUP BLACK YELLOW 0 SETATR 0 SPN ! SCLR HIRES S2COL
  0 COL !
  6 ROW ! 1 SPN ! PUTBLK 2 HGT ! 0 WID ! 0 SPN ! ;
: LEFT WRL1 ;
: RIGHT WRR1 ;
: KEYS 61 KB IF LEFT ENDIF 37 KB RIGHT ENDIF ;
: EXA SETUP BEGIN KEYS 47 KB UNTIL ;
```

The above program scrolls a ghost left or right under keyboard control.

'SETUP' sets up the screen, puts it into hires mode and places the sprite on the screen.
'LEFT' and 'RIGHT' are the scroll words and could be WRL2 or WRL8 etc. They scroll the whole of the 40 x 2 character window.
'KEYS' if the '<' key is pressed (61 KB) 'LEFT' is executed or if the '>' key is pressed (37 KB) 'RIGHT' is executed.
'EXA' runs the program and sees if the Commodore key is pressed to exit from the program.

Type EXA to run the program '<' to move left and '>' to move right. The Commodore key exits the program.

The equivalent Basic Lightning program of EXA would be:

```
 10    LABEL SETUP
 20    SETATR0, YELLOW, BLACK : SCLR0,ATR: HIRES: S2COL: PUTBLK1,0,6
 30    PROCEND
 40    '
 50    LABELLEFT : WRL10,0,6,40,2 : PROCEND
 60    '
 70    LABELRIGHT : WRR10,0,6,40,2 : PROCEND
 80    '
 90    LABELKEYS
100    IF KB(61) =-1 THEN PROCLEFT
110    IF KB(37) =-1 THEN PROCRIGHT
120    PROCEND
130    '
140    LABELEXA
150    PROCSETUP
160    REPEAT : PROCKEYS : UNTIL KB(47) =-1
170    PROCEND
```

To run this program type PROCEXA

SIMPLE PUTTING

Another fairly simple means of moving sprites around the screen is to simply PUT sprites with a blank border around them. Suppose the sprite you want to move is sprite number 2, which is two characters wide and two characters high.

Firstly a border 1 character wide will have to be constructed around it, making it a 4 by 4 character sprite. This can be done either in the sprite generator pragram or via the following program.

We will create a 4 by 4 sprite and give it the number 11. Type the following colon definitions.

```
: SETUP BLUE WHITE 0 SETATR 0 SPN ! SCLR ;
: MAKE 11 SPN ! 4 HGT ! 4 WID ! ISPRITE SCLR ATTOFF
2 SPN ! 0 COL ! 0 ROW ! 2 WID ! 2 HGT ! 11 SPN2 !
1 COL2 ! 1 ROW 2 ! MOVBLK ATTON ;
```

'SETUP' sets up the attributes of the screen ready for the sprite.
'MAKE' creates the 4x4 sprite and moves sprite 2 into it. Now type in the rest of this example

```
:   UP 32 KB IF ROW @ 0 > MINUS ROW +! ENDIF ;
:   DOWN 24 KB IF ROW @ 21 < ROW +! ENDIF ;
:   LEFT 40 KB IF COL @ 0 > MINUS COL +! ENDIF ;
:   RIGHT 48 KB IF COL @ 36 < COL +! ENDIF ;
:   EXB HIRES S2COL 11 SPN ! 19 COL ! 11 ROW ! BEGIN UP DOWN LEFT RIGHT
    PUTBLK 47 KB UNTIL ;
```

The definitions 'UP' and 'DOWN', 'LEFT' and 'RIGHT' control the movement of the character on the screen.

'EXB' sets up the screen and scans the commodore key to see if it is pressed while executing UP,DOWN,LEFT and RIGHT.

To create sprite 11 type SETUP MAKE and to run the program type EXB. The keys are f1 up, f3 left, f5 right, f7 down and Commodore to exit.

The great limitation of this routine, however, is that data already on the screen will be replaced by the sprite being PUT over it.

The equivalent Basic Lightning program will be.

```
 10     LABELSETUP
 20     SETATR0, WHITE, BLUE : SCLR0, ATR
 30     PROCEND
 40     '
 50     LABELMAKE
 60     SPRITE 11,4,4:SCLR11,ATR:ATTOFF:MOVBLK 2,0,0,2,2,11,1,1:ATTON
 70     PROCEND
 80     '
 90     LABELKEYS
100     IF KB (32)=-1 ANDR>0 THENR=R-1
110     IF KB (24)=-1 ANDR<21 THENR=R+1
120     IF KB (40)=-1 ANDC>0 THENC=C-1
130     IF KB (48)=-1 ANDC<36 THENC=C+1
140     PROCEND
150     '
160     LABELEXB
170     HIRES:S2COL: R=11: C=19: REPEAT: PROCKEYS: PUTBLK11,C,R: UNTIL
        KB(47)=-1
180     PROCEND
```

Type PROC SETUP: PROCMAKE to create sprite 11 and PROCEXB to run the program.

Before considering the more sophistocated methods available to overcome this problem, let's consider a simpler method of circumventing this problem. Supposing the screen holds half a dozen or so fixed objects and we wish to make the man in the last example move through these objects. First let's scatter some sprites about the screen.

```
: SETUP WHITE BLACK 0 SETATR 0 SPN ! SCLR 4 SPN !
3 2 4 5 6 3 5 12 10 9 12 14 6 0 DO ROW ! COL ! PUTORS LOOP ;
```

Notice that we use the PUTOR word to OR data to the screen, the reason for this will become clear later.
We will now redefine UP, DOWN, LEFT, RIGHT so that sprite 11 is only 'PUT' if it is to be moved. The new code becomes:

```
: KCHK KB DUP ROT OR SWAP ;
: UP 32 KCHK IF ROW @ 0 > MINUS ROW +! ENDIF ;
: DOWN 24 KCHK IF ROW @ 21 < ROW +! ENDIF ;
: LEFT 40 KCHK IF COL @ 0 > MINUS COL +! ENDIF ;
: RIGHT 48 KCHK IF COL @ 36 < COL +! ENDIF ;
```

The complete definition then becomes.

```
: EXC BLUE WHITE 0 SETATR 0 SPN ! SCLR HIRES S2COL ATTOFF
BEGIN COL @ ROW @ SETUP ROW ! COL ! 11 SPN ! UP DOWN LEFT RIGHT
IF PUTBLK ENDIF 47 KB UNTIL ;
```

What happens is that as soon as the sprite is PUT to the screen, all screen data is immediately 'ORed' so that if any data was blotted out it is immediately replaced.

The Basic Lightning program would be:

```
 10 LABELSETUP
 20 PUTOR 4,3,2 : PUTOR 4,4,5 : PUTOR 4,6,3 : PUTOR4,5,12 : PUTOR4,10,9 :
    PUTOR4,12,14
 30 PROCEND
 40 '
 50 LABELKEYS : FG=0
 60 IF KB(32) =-1 AND R>0 THEN R=R-1 : FG=1
 70 IF KB(24) =-1 AND R<21 THEN R=R+1 : FG=1
 80 IF KB(40) =-1 AND C>0 THEN C=C-1 : FG=1
 90 IF KB(48) =-1 AND C<36 THEN C=C+1 : FG=1
100 PROCEND
110 '
120 LABEL EXC
130 SETATR0,WHITE,BLUE : SCLR0,ATR : HIRES : S2COL : ATTOF : R=10 : C=10
140 REPEAT : PROCSETUP : PROCKEYS : IFFG=1 THEN PUTBLK 11,C,R
150 UNITL KB(47)=-1 : PROCEND
```

To run type PROC EXC

MORE ADVANCED TECHNIQUES

Quite often it is not practical to repeatedly PUT the screen data which accompanies a moving sprite and more frequently movement is required with a higher resolution than one character.

To begin with let's consider the problem of improving the resolution of the movement. Let's work again with a 2 x 2 sprite (Sprite 8).

You may if you wish type COLD to clear previous examples.

Suppose we wish to move the sprite around the screen with 2 pixel resolution. This means that between 2 successive columns there are 4 intermediate orientations, each successive orientation being 2 pixels right shifted. This means that 4 sprites in all before the cycle is repeated at the next column position.

To begin with let's set up the 4 sprites and number them 20, 21, 22 and 23. To create the sprites type:

```
: MAKE BLACK WHITE 0 SETATR 2 HGT ! 4 WID ! 24 20
DO I SPN ! ISPRITE SCLR LOOP ;
```

This will define and clear the 4 sprites and we can now put the character in its various orientations into these sprites. There are two stages to this operation. Firstly sprite 8 needs to be put into sprite 20 then sprites 20 to 23 need to be scrolled and put successively to build up the four orientations.

```
: SET1 8 SPN ! 0 COL ! ROW ! 2 WID ! 2 HGT !
20 SPN2 ! 1 COL2 ! 0 ROW2 ! MONBLK ;
```

This sets up sprite 20 and the remaining 3 orientations are set up from this sprite, using

```
: SET2 23 20 DO I SPN ! I 1+ DUP SPN2 ! CPYBLK
SPN ! 0 COL ! 0 ROW ! 4 WID ! 2 HGT ! SCR2
LOOP ;
```

It's worth putting these sprites on the screen to see what they look like, assuming you've executed the words MAKE, SET1 and SET2.

Firstly type

```
: EXD 20 COL ! 4 0 DO I 20 + SPN ! I DUP + ROW ! PUTBLK LOOP ;
```

Now once EXD has been compiled type

```
BLACK WHITE 0 SETATR 0 SPN ! SCLR
```

to clear the hires screen.

Now type 10 WINDOW EXD to see the sprites.

Now this gives us 2 pixel horizontal resolution so that we now have 160 horizontal plotting positions in the range 0 to 159. We need a simple formula which will calculate the sprite number and the column from the horizontal plotting position. This turns out to be very simple.

```
: HPLOT 4 /MOD COL ! 20 + SPN ! PUTBLK ;
```

So to PUT at x-position 27 (54 pixals from the left hand border) just use ;

```
27 HPLOT
```

A program that will move the sprites 20 to 23 under keyboard control is given below. The program uses HPLOT and has the < key to move left, the > key to move right and the commodore key to break. To run the program type EXE

```
   80 VARIABLE XPOS
: LEFT 61 KB IF XPOS @ 0 > MINUS XPOS +! ENDIF ;
: RIGHT 37 KB IF XPOS @ 144 < XPOS +! ENDIF ;

: EXE BLACK WHITE 0 SETATR 0 SPN ! SCLR HIRES S2COL BEGIN LEFT RIGHT XPOS @
HPLOT 47 KB UNTIL ;
```

The variable XPOS stores the position of the sprite.

The equivalent Basic Lighting Routines would be.

```
 10 LABELMAKE
 20 SETATR 0, WHITE, BLACK : FORN=20TO23 : SPRITEN,4,2 : SCLRN, ATR : NEXTN :
    PROCEND
 30 '
 40 LABELSET1
 50 MOV BLK 8,0,0,2,2,20,1,0 : PROCEND
 60 '
 70 LABELSET2
 80 FORN=20TO22 : CPYBLKN, N+1 : SCR2N+1,0,0,4,2 : NEXTN : PROCEND
 90 '
100 LABELEXD
110 FORN=0 TO 3 : PUTBLKN+20,20,N*2  NEXTN : PROCEND
120 '
130 LABELHPLOT
140 X=INT(XP/4) : Y=XP-(X*4) : PUTBLKY+20,X,0 : PROCEND
150 '
160 LABEL KEYS
170 IF KB(61)=-1 AND XP>0 THEN XP=XP-1
180 IF KB(37)=-1 AND XP<144 THEN XP=XP+1
190 PROCEND
200 '
210 LABEL EXE
220 SETATR0, WHITE, BLACK : XP=80 : SCLR0, ATR : HIRES : S2COL
    REPEAT : PROCKEYS : PROCHPLOT
230 UNTILKB(47) =-1 : PROCEND
```

To run the program type PROC EXE

If we are going to give the same resolution of movement (2 pixels) in the vertical plane, as well as the horizontal plane, we are going to need 4 vertically shifted orientations for each of the horizontally offset orientations (16 sprites in all). This time they will need to be 4 x 4. If you have finished with the last example you will need to delete the old sprites numbered 20 to 23, by typing

```
20 SPN ! WIPE
21 SPN ! WIPE
22 SPN ! WIPE
23 SPN ! WIPE
```

You can now type COLD to clear the dictionary. Type in MAKE which creates 16 4 x 4 sprites numbered 20 to 35.

```
: MAKE BLACK WHITE 0 SETATR 4 HGT ! 4 WID ! 36 20 DO I SPN ! ISRITE SCLR LOOP ;
```

Each of the horizontally offset orientations needs to be vertically offset by 2 pixels into 4 further orientations 20 will be offset into 24,28,32. 21 will be offset into 25,29,33 and so on. We will need a new word SET3 to do this.

```
: SET3 10 SPN ! 0 COL ! 0 ROW ! 2 WID ! 2 HGT ! 20 SPN2 !
1 COL2 ! 1 ROW2 ! MOVBLK 23 20 DO I SPN ! I 1+ DUP
SPN2 ! CPYBLK SPN ! 4 WID ! 4 HGT ! SCR2 LOOP -2
NUM ! 24 20 DO I DUP 12 + SWAP DO I DUP 4 + SPN2 !
DUP SPN ! CPYBLK 4 + SPN ! SCROLL 4 +LOOP LOOP ;
```

Once SET3 has been entered, compiled and executed; the definitions can be forgotten. Since we now have 2 pixel resolutions in the vertical and horizontal directions, we have 160 horizontal positions and 100 vertical positions. We need a word which can calculate the sprite number, column and row from the 2 pixel resolution co-ords X and Y. The following word assumes that the vertical and then the horizontal co-ords have been placed on the stack.

```
: XYPUT 4 /MOD COL ! SWAP 4 /MOD ROW ! DUP + DUP + + 20 + SPN ! ;
```

So to put at X-position 30 (60 pixel)
and Y-position 17 (84 pixel)

you use 17 30 XYPUT PUTBLK

Note that the 20 at the end of the definition XYPUT is the first of the 16 sprites.

Now lets deal with the animation of the sprite itself.

Perhaps the most powerful method of Sprite animation is via the XOR operation. The usefulness of this operation stems from the fact that when an object is XORed with the screen, the screen data can be restored simply by repeating the operation. The area of screen is restored to the same state as it was before the first operation.

We can now write a routine which moves a sprite around the screen under keyboard control using a slightly amended form of the word XYPUT. The routine below assumes the 16 sprites in the range 20 to 35, which each have demensions 4 x 4, have been set up in the example.

The following variables will be used.

TSPN	Temporary value for SPN
TCOL	Temporary value for COL
TROW	Temporary value for ROW
XC	X co-ordinate
YC	Y co-ordinate
FLAG	Collision flag

We'll also use a constant FSPN which holds the number of the first sprite in the series of 16. To adapt these routines for your own use just change the value of FSPN.

```
0 VARIABLE TSPN 0 VARIABLE TCOL 0 VARIABLE TROW.
0 VARIABLE XC 0 VARIABLE YC 20 CONSTANT FSPN
0 VARIABLE FLAG

: PLOAD COL ! ROW ! SPN ! PUTXOR ;
: PSET TSPN @ TROW @ TCOL @ ;
: PCAL 4 /MOD TCOL ! SWAP 4 /MOD TROW ! DUP + DUP + FSPN + + TSPN ! ;
: PLACE YC @ XC @ PCAL PSET PLOAD ;
: MOVE YC @ XC @ PCAL PSET PUTXOR PLOAD ;
```

We now need to poll the keyboard. We use the word KCHK again so the character is not re-PUT unless a key has been pressed.

```
: KCHK KB DUP RIOT OR SWAP ;
: UP 32 KCHK IF YC @ 4 > MINUS YC +! ENDIF ;
: DOWN 24 KCHK IF YC @ 83 < YC +! ENDIF ;
: LEFT 40 KCHK IF XC 4 > MINUS XC +! ENDIF ;
: RIGHT 48 KCHK IF YC @ 140 < XC +! ENDIF ;
: KREAD 0 UP DOWN LEFT RIGHT ;
```

'KREAD' will leave 0 on the stack if no key was pressed or 1 if a key was pressed.

The complete colon definition becomes

```
: EXF BLACK WHITE 0 SETATR 0 SPN ! SCLR HIRES S2COL
10 SC ! 10 YC ! PLACE BEGIN KREAD IF MOVE ENDIF
47 KB UNTIL ;
```

The function keys are used in the same way as in the previous example. The commodore key breaks the program.

The Basic Lightning version of EXF is as follows:

```
 10 LABEL MAKE
 20 SETATR 0, WHITE, BLACK : FORN = 20 TO 35 : SPRITEN,4,4 :
    SCLRN, ATR : NEXTN
 30 PROCEND
 40 '
 50 LABEL SET3
 60 MOVBLK 10,0,0,2,2,20,1,1
 70 FORN = 20 TO 22 : CPYBLK N, N+1 : SCR2N+1,0,0,4,4 : NEXTN
 80 FORN = 20 TO 23
 90 FORI = NTO(N+11) STEP4 : CPYBLKI,(I+4) : SCROLL (I+4),0,0,4,4,-2 :
    NEXTI : NEXTN
100 PROCEND
110 '
120 LABEL KEYS : FG=0
130 IF KB(32) = -1 AND YC>4 THEN YC = YC-1 : FG=1
140 IF KB(24) = -1 AND YC<83 THEN YC = YC+1 : FG=1
150 IF KB(40) = -1 AND XC>4 THEN XC=XC-1 : FG=1
160 IF KB(48) = -1 AND XC<40 THEN XC=XC+1 : FG=1
170 PROCEND
180 '
190 LABELPCAL
200 TC=INT(XC/4) : TR=INT (YC/4) : TS = ((XC-(TC*4)) + ((YC-(TR*4))+20)
    : PROCEND
210 '
220 LABELPLOAD
230 COL = TC : ROW=TR : SPN=TS : PROCEND
240 '
250 LABELEXF
260 SETATR0, WHITE, BLACK : SCLR0,ATR : HIRES : S2COL : XC=10 : YC=10
270 PROCCAL : PUTXOR TS,TC,TR
280 REPEAT : PROCKEYS
290 IF FG=1 THEN PUTXOR : PROCPCAL : PROCPLOAD : PUTXOR TS, TC, TR
    : PRINT TS,XC,YC
300 UNTIL KB(47) =-1
310 PROCEND
```

To run the program type PROCEXF.