WHITE LIGHTNING

CHEAT SHEET
&
MICRODRIVE MANUAL

Do not lose this manual as separate manuals cannot be supplied under any circumstances

CONTENTS

WHITE LIGHTNING

CHEAT SHEET

This sheet is intended as a supplement to the User Manual and is provided only to "get you started". The best way to use it is to type in the source code at the end of this section and execute the appropriate section as you go. You will need the demonstration sprites in memory but sprites 25 to 255 should be deleted to make enough memory available. A word to do this is:

```
: CLRS 256 25 DO I SPN ! TEST IF DSPRITE ENDIF LOOP ; <CR>
```

Type this in then type CLRS <CR>

It is also worth deleting the large sprites 9, 12, 13 and 14 using:

```
 9 SPN ! DSPRITE <CR>
12 SPN ! DSPRITE <CR>
13 SPN ! DSPRITE <CR>
14 SPN ! DSPRITE <CR>
```

where <CR> means press ENTER. After execution CLRS should be FORGOTTEN using FORGET CLRS <CR>

Following the example source there is a listing of a complete game. You won't be able to run it without the sprites but we can provide a tape with the source and sprites for one pound seventy five pence if you think it would to helpful.

Creating Large Sprites

Quite often in games writing large sprites are required which may extend across several screens. The sprite development software can produce sprites with dimensions of up to 15 by 15 characters. Larger sprites need to be constructed in the White Lightning itself.

The sample listing for the Lunar Lander contains a routine which we can use as an example. The Sprite Development Program was used to produce 16 sprites, each was 3 high by 8 wide, having sprite numbers 10 to 25. The following routine sets up sprite 128 which is 3 high by 128 wide, and then fills it with the 16 small sprites before deleting them to save memory.

```
SCR#8
0 : MAKE 128 SPN ! 3 HGT ! 128 LEN ! 0 SROW ! 128 SP2 ! ISPRITE
1 16 0 DO I 10 + DUP SPN ! SP1 ! I 8 * SCOL ! GWBLM GWATTM DSPRITE
2 LOOP ;
```

Line 0 just defines the big sprite in memory and sets SROW to 0.

Line 1 loops round 16 times with I taking the values 0 to 15. I 10 + calculates the sprite number of the smaller sprite and I 8 * calculates the column in the big sprite that this smaller sprite is to be put into. GWBLM GWATTM moves the pixel data, then the attributes, from the individual small sprites into the big sprite. You won't be able to execute this example without the 16 3 by 8 sprites but it illustrates how big sprites are achieved.

Moving Sprites

The chief problem facing the programmer who wants to move sprites around the screen is choosing from the numerous schemes available. We 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

Where an object is to be moved within a screen window that does not contain any other objects the screen itself can be scrolled. This is particularly applicable where movement is either horizontal or vertical. Diagonal movement is also possible however. In some cases the object is constrained to move on the screen because parts of the object scrolled off the screen without wrap are lost.

To begin with let's consider a very simple example - moving an invader (demo sprite 24) 2 characters high and 2 characters wide left and right under keyboard control.

The routine is in three sections: the first sets up the base and initial parameters, the second moves left or right and the third polls the keyboard and controls the movement.

```
: SETUP 0 COL ! 6 ROW ! 24 SPN ! CLS SETAM PUTBLS 2 HGT ! 32 LEN ! ;
: LEFT WRL1V ; : RIGHT WRR1V ;
: KEYS 1 1 KB IF LEFT ENDIF 8 1 KB IF RIGHT ENDIF ;
```

This will move the base by 1 pixel left or right but by changing the words LEFT and RIGHT to be WRR4V and WRL4V or WRR8V and WRL8V movement of 4 or 8 pixels can be achieved.

To try these routines we'll need a small test routine.

```
: TESTA ATTOFF SETUP BEGIN KEYS 8 2 KB UNTIL ;
```

TESTA will loop around until SYMBOL SHIFT is pressed. Pressing CAPS SHIFT will move the base left, pressing space will move the base right and pressing SYMBOL SHIFT will exit the loop.

This routine can also be executed in background using:

```
: TESTB EXX SETUP EXX ' KEYS INT-ON ;
```

To halt, just type INT-OFF.

If you've typed in the source at the end of this section type 6 LOAD to compile and then TESTA or TESTB to execute.

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 2 characters high and 2 characters wide. You will need to construct a 4 by 4 sprite so that the 2 by 2 sprite can be contained with a border around the outside. Make sure background is off by typing INT-OFF.

Suppose for example, sprite 24 is a 2 by 2 sprite (you can use demo sprite 24 for the purposes of this example). We'll use sprite 62 for the sprite to be moved. To set up sprite 62 use:

```
: MAKE 62 SPN ! 4 HGT ! 4 LEN ! TEST 0= IF ISPRITE CLSM ENDIF ;
```

This will create and clear sprite 62. Now use:

```
1 SCOL ! 1 SROW ! 62 SP2 ! 24 SP1 ! GWBLM
```

Sprite 62 is now set up ready for use. We now need four words to move the sprite UP, DOWN, LEFT and RIGHT.

```
: UP    7 1 KB IF ROW @  0 > MINUS ROW +! ENDIF ;
: DOWN  8 1 KB IF ROW @ 20 < ROW +! ENDIF ;
: LEFT  1 1 KB IF COL @  0 > MINUS COL +! ENDIF ;
: RIGHT 1 2 KB IF COL @ 28 < COL +! ENDIF ;
```

The ENTER and SPACE keys will move the sprite up and down and CAPS SHIFT and Z will move the sprite left and right respectively.

The full word to animate the sprite becomes:

```
: TESTC 62 SPN ! 10 COL ! 10 ROW ! BEGIN UP DOWN LEFT RIGHT ADJM PWBLS 6 1 KB
UNTIL ;
```

To exit TEST we press the key at 6 1, this is the "P" key.

If you're using the source at the end of this section type 7 LOAD MAKE TESTC. The great limitation of this routine, however, is that data already on the screen will be replaced by the sprite being PUT and subsequently lost. Before considering the more sophisticated methods available to us which overcome this limitation let's just consider some simpler methods of circumventing this problem, which will work for similar situations.

Supposing the screen holds half a dozen or so fixed objects and we wish to move the invader in the last example through these objects.

First of all let's set up a screen with these objects scattered throughout.

```
: SETUP 4 SPN ! 3 2 4 5 6 3 5 12 10 9 12 14 6 0 DO ROW ! COL ! PUTRS LOOP ;
```

Notice that we use the PUTORS word to OR data to the screen; the reason for this will become clear.

We'll now redefine UP, DOWN, LEFT and RIGHT so that sprite 62 is only 'PUT' if it is moved, the new code becomes:

```
: KCHK KB DUP ROT OR SWAP ;
: UP    7 1 KCHK IF ROW @  0 > MINUS ROW +! ENDIF ;
: DOWN  8 1 KCHK IF ROW @ 20 < ROW +! ENDIF ;
: LEFT  1 1 KCHK IF COL @  0 > MINUS COL +! ENDIF ;
: RIGHT 1 2 KCHK IF COL @ 28 < COL +! ENDIF ;
```

The complete word becomes:

```
: TESTD CLS BEGIN COL @ ROW @ SETUP ROW ! COL ! 62 SPN ! @ UP DOWN LEFT RIGHT IF
ADJM PWBLS ENDIF 6 1 KB UNTIL ;
```

What is happening is that as soon as the moving sprite is PUT to the screen all screen data is immediately "OR"ed so that if any was blotted out, it is immediately replaced. Type "P" to exit.

To use the source code listings type 8 LOAD TESTD

Often it is not practical to repeatedly PUT the screen data which accompanies the 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 2x2 sprite (sprite 24 of the demo sprites will do). 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 we need 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 100, 101, 102 and 103. To create these 4 sprites, type:

```
: MAKE 2 HGT ! 3 LEN ! 104 100 DO I SPN ! ISPRITE CLSM 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 24 needs to be put into sprite 100, then sprites 100 to 103 need to be scrolled and PUT successively to build up the four orientations.

```
: SET1 0 SROW ! 0 SCOL ! 24 SP1 ! 100 SP2 ! GWBLM ;
```

This sets up sprite 100 and the remaining 3 orientations are set up from this sprite using:

```
: SET2 103 100 DO I SP1 ! I 1+ DUP SP2 ! SPN ! COPYM WRR1M WRR1M 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 use:

```
: TESTE CLS 0 COL ! 4 0 DO I 100 + SPN ! I DUP + ROW ! PUTBLS LOOP 8 0 AT ;
```

This will place the 4 orientations, one above the other, so that the resolution of the movement can be seen.

To use the source version type 9 LOAD MAKE SET1 SET2 TESTE

This now gives us 2 pixel horizontal resolution so that we now have 128 horizontal plotting positions in the range 0 to 127. 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 ! 100 + SPN ! PUTBLS ;
```

So to PUT at X-position 27 (54 pixels from the left hand column) just use:

```
27 HPLOT
```

The previous example is useful in that it indicates a way of producing high resolution PUTting but as it stands cannot be used for animation because it does not enable the removal of previously placed orientations. Before looking at a scheme for animating these orientations let's generalise this example to cover high resolution vertical movement as well as horizontal movement.

If we're going to give the same resolution of movement (2 pixels) in the vertical plane, we're 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 3x3 as opposed to the previously defined 2x3. If you've typed in the last example you'll need to delete the old sprites numbered 100 to 103. If so, type:

```
100 SPN ! DSPRITE <CR>
101 SPN ! DSPRITE <CR>
102 SPN ! DSPRITE <CR>
103 SPN ! DSPRITE <CR>
```

Now type COLD to clear the dictionary.

To create the 16 new sprites use:

```
: MAKE 3 HGT ! 3 LEN ! 116 100 DO I SPN ! ISPRITE CLSM SETAM  LOOP ;
```

SET1 and SET2 are now used in exactly the same form as in the previous example to set up the first 4 sprites 100 to 103. Each of the horizontally offset orientations needs to be vertically offset by 2 pixels into 4 further orientations. 100 will be offset into 104, 108 and 112; 101 will be offset into 105, 109 and 113 and so on. We'll need a third word SET3 to do this.

```
: SET3 -2 NPX ! 104 100 DO I DUP 12 + SWAP DO I DUP 4 + DUP SP2 ! SPN ! SP1 !
GWBLM SCRM 4 +LOOP LOOP ;
```

Once SET1, SET2 and SET3 have been entered, compiled and executed, the definitions can be forgotten. Since we now have 2 pixel resolution in the vertical and horizontal directions we have 128 horizontal positions and 96 vertical positions. We need a word which can calculate sprite number, column and row from the 2 pixel resolution co-ords X and Y. The following word assumes the vertical then horizontal co-ords have been placed on the stack.

```
: XYPUT 4 /MOD COL ! SWAP 4 /MOD ROW ! DUP + DUP + + 100 + SPN ! ;
```

```
So to PUT at X-position 30 (pixel 60)
             Y-position 17 (pixel 34)

use          17 30 XYPUT PUTBLS
```

Note that the 100 + SPN ! at the end of the definition of XYPUT should be amended so that the number is the sprite number of the first of your 16 sprites.

To use the source type 9 LOAD 10 LOAD MAKE SET1 SET2 SET3 (Ignore MSG# 4's)

Let's now 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 XOR'ed with the screen, the screen 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 (see page 22 of the White Lightning manual).

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 10 sprites in the range 100 to 115, which each have dimensions 3x3, have been set up in the last 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 FSPN.

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

We now need to poll the keyboard:

We use the KCHK word again so that the character is not re-PUT unless a key has been pressed.

```
: KCHK KB DUP ROT OR SWAP ;
: UP 7 1 KCHK IF YC @ 4 > MINUS YC +! ENDIF ;
: DOWN 8 1 KCHK IF YC @ 83 < YC +! ENDIF ;
: LEFT 1 1 KCHK IF XC @ 4 > MINUS XC +! ENDIF ;
: RIGHT 1 2 KCHK IF XC @ 115 < 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 word becomes:

```
: TESTF 10 XC ! 10 YC ! PLACE BEGIN KREAD IF MOVE ENDIF 6 1 KB UNTIL ;
```

Note that this loop assumes no interference with the values of COL, ROW or SPN between cycles, if you are executing another Forth word, for example a word called TRY, then make sure you temporarily stack COL, ROW and SPN, e.g.

```
... BEGIN KREAD IF MOVE ENDIF COL @ ROW @ SPN @ TRY SPN ! ROW ! COL ! 0 1 KB UNTIL
...
```

This word can be easily executed under interrupt using

```
: TESTG KREAD IF MOVE ENDIF ;
: TESTH 10 XC ! 10 YC ! EXX PLACE EXX ' TESTG INT-ON BEGIN 6 1 KB UNTIL INT-OFF
;
```

You will notice, however, that if you do execute this routine under interrupt then the sprite may flicker in passage through certain areas of the screen. This is due to the finite time taken for the dot to scan the screen and can be very annoying.

To execute from source type 11 LOAD 12 LOAD then TESTF or TESTH

Let's look now at some more powerful techniques which not only help with the flickering but also include collision detection facilities.

Collision Detection

To produce the smoothest movement of all, and include collision detection, a six stage operation is used. The technique utilises two dummy sprites, and all intermediate stages of the operation are carried out in memory. For this example let's number the dummy sprites 116 and 117. To set up the sprites use:

```
: MAKED 3 HGT ! 3 LEN ! 116 SPN ! ISPRITE 5 HGT ! 5 LEN ! 117 SPN ! ISPRITE ;
```

This needs to be typed in and executed before executing the Source Code or error MSG# 10 will be produced.

The six stage procedure is as follows:

1. The last orientation PUT, together with a one character surround, are GOT into the 5x5 dummy sprite.

2. The last orientation is GWXRM'ed out of the 5x5, restoring the original screen data.

3. The new orientation is COPYM'ed into the 3x3 dummy.

4. The screen data in the 5x5 is PWNDM'ed into the 3x3 and SCANM performed on the 3x3 to detect any collision. A flag is set.

5. The new orientation is GWXRM'ed into the 5x5 dummy.

6. The 5x5 is PUTBLS'ed onto the screen.

The code for this algorithm is the same as the previous example except that the word MOVE needs to be modified.

Define Sprites as for previous example and execute MAKED then use:

```
: STEP1 -1 COL +! -1 ROW +! SPN @ 117 SPN ! GETBLS ;
: STEP2 1 SCOL ! 1 SROW ! 117 SP2 ! SP1 ! GWXRM ;
: STEP3 116 SP2 ! TSPN @ SP1 ! COPYM ;
: STEP4 COL @ - SCOL ! ROW @ - SROW ! 117 SP2 ! 116 SP1 ! PWNDM 116 SPN ! SCANM
FLAG +! ;
: STEP5 SP1 ! GWXRM 117 SPN ! PUTBLS PSET COL ! ROW ! SPN ! FLAG @ IF 100 100
BLEEP 0 FLAG ! ENDIF ;
: MOVE YC @ XC @ PCAL PSET STEP1 STEP2 STEP3 STEP4 STEP5 ;
```

To use the source code type COLD to clear all previous examples then:

```
11 LOAD FORGET MOVE 14 LOAD 12 LOAD 13 LOAD
```

to compile the words TESTF and TESTH.

The Bleep will sound when the sprite collides with any other screen data.

```
SCR # 6
  0 : SETUP 0 COL ! 6 ROW ! 24 S
PN ! CLS PUTBLS 2 HGT ! 32 LOW ! ;
  1 : LEFT WRL1V ; : RIGHT WRR1V
 ;
  2 : KEYS 1 1 KB IF LEFT ENDIF
8 1 KB IF RIGHT ENDIF ;
  3 : TESTA ATTOFF SETUP BEGIN K
EYS 8 2 KB UNTIL ;
  4 : TESTB ATTOFF EXX SETUP EXX
 ' KEYS INT-ON ;
  5
  6
  7

SCR # 7
  0 : MAKE 62 SPN ! 4 HGT ! 4 LE
N ! TEST 0= IF ISPRITE CLSM ENDI
F
  1 1 SCOL ! 1 SROW ! 62 SP2 ! 2
4 SP1 ! GWBLM ;
  2 : UP 7 1 KB IF ROW @ 0 > MIN
US ROW +! ENDIF ;
  3 : DOWN 8 1 KB IF ROW @ 20 <
ROW +! ENDIF ;
  4 : LEFT 1 1 KB IF COL @ 0 > M
INUS COL +! ENDIF ;
  5 : RIGHT 1 2 KB IF COL @ 28 <
 COL +! ENDIF ;
  6 : TESTC 62 SPN ! 10 COL ! 10
 ROW ! CLS BEGIN UP DOWN LEFT RI
GHT
  7 : ADJM PWBLS 6 1 KB UNTIL ;

SCR # 8
  0 : SETUP 4 SPN ! 3 2 4 5 6 3
5 12 10 9 12 4 6 0 DO ROW ! COL
!
  1 : PUTORS LOOP ; : KCHK KB DUP
ROT OR SWAP ;
  2 : UP 7 1 KCHK IF ROW @ 0 > M
INUS ROW +! ENDIF ;
  3 : DOWN 8 1 KCHK IF ROW @ 20
< ROW +! ENDIF ;
  4 : LEFT 1 1 KCHK IF COL @ 0 >
 MINUS COL +! ENDIF ;
  5 : RIGHT 1 2 KCHK IF COL @ 28
 < COL +! ENDIF ;
  6 : TESTD CLS BEGIN COL @ ROW
@ SETUP ROW ! COL ! 62 SPN !
  7 0 UP DOWN LEFT RIGHT IF ADJM
 PWBLS ENDIF 6 1 KB UNTIL ;

SCR # 9
  0 : MAKE 2 HGT ! 3 LEN ! 104 1
00 DO I SPN ! ISPRITE CLSM LOOP
;
  1 : SET1 0 SROW ! 0 SCOL ! 24
SP1 ! 100 SP2 ! GWBLM ;
  2 : SET2 103 100 DO I SP1 ! I
1+ DUP SP2 ! SPN ! COPYM WRR1M
  3 WRR1M LOOP ;
  4 : TESTE CLS 0 COL ! 4 0 DO I
 100 + SPN ! I DUP + ROW ! PUTBL
S
  5 LOOP 8 0 AT ;
  6 : HPLOT 4 /MOD COL ! 100 + S
PN ! PUTBLS ;
  7

SCR # 10
  0 : MAKE 3 HGT ! 3 LEN ! 116 1
00 DO I SPN ! ISPRITE CLSM SETAM
  1 LOOP ;
  2 : SET3 -2 NPX ! 104 100 DO I
 DUP 12 + SWAP DO I DUP 4 + DUP
SP2
  3 ! SPN ! SP1 ! GWBLM SCRM 4
 +LOOP LOOP ;
  4 : XYPUT 4 /MOD COL ! SWAP 4
/MOD ROW ! DUP + DUP + + 100 + S
PN
  5 ! ;
  6
  7

SCR # 11
  0 0 VARIABLE TSPN 0 VARIABLE T
COL 0 VARIABLE TROW 0 VARIABLE
XC
  1 0 VARIABLE YC 100 CONSTANT F
SPN 0 VARIABLE FLAG
  2 : PLOAD COL ! ROW ! SPN ! PU
TXRS ;
  3 : PSET TSPN @ TROW @ TCOL @
;
  4 : PCAL 4 /MOD TCOL ! SWAP 4
/MOD TROW ! DUP + DUP + FSPN + +
  5 TSPN ! ;
  6 : PLACE YC @ XC @ PCAL PSET
PLOAD ;
  7 : MOVE YC @ XC @ PCAL PSET P
UTXRS PLOAD ;

SCR # 12
  0 : KCHK KB DUP ROT OR SWAP ;
  1 : UP 7 1 KCHK IF YC @ 4 > MI
NUS YC +! ENDIF ;
  2 : DOWN 8 1 KCHK IF YC @ 83 <
 YC +! ENDIF ;
  3 : LEFT 1 1 KCHK IF XC @ 4 >
MINUS XC +! ENDIF ;
  4 : RIGHT 1 2 KCHK IF XC @ 115
 < XC +! ENDIF ;
  5 : KREAD 0 UP DOWN LEFT RIGHT
 ;
  6 : TESTF 10 XC ! 10 YC ! PLAC
E BEGIN KREAD IF MOVE ENDIF 6 1
KB
  7 UNTIL ;
```

```
SCR #13
  0 : TESTG KREAD IF MOVE ENDIF
;
  1 : TESTH 10 XC ! 10 YC ! EXX
PLACE EXX ' TESTG INT-ON BEGIN 6
 1
  2 KB UNTIL INT-OFF ;
  3
  4
  5
  6
  7

SCR # 14
0 : STEP1 -1 COL +! -1 ROW +!
SPN @ 117 SPN ! GETBLS ;
  1 : STEP2 1 SCOL ! 1 SROW ! 11
7 SP2 ! SP1 ! GWXRM ;
  2 : STEP3 116 SP2 ! TSPN SP1
! COPYM ;
  3 : STEP4 COL @ - SCOL ! ROW @
 - SROW ! 117 SP2 ! 116
  4 SP1 ! PWNDM 116 SPN ! SCANM
FLAG +! ;
  5 : STEP5 SP1 ! GWXRM 117 SPN
! PUTBLS PSET COL ! ROW ! SPN !
  6 FLAG @ IF 100 100 BLEEP 0 FL
AG ! ENDIF ;
  7 : MOVE YC @ XC @ PCAL PSET S
TEP1 STEP2 STEP3 STEP4 STEP5 ;
```

NOTE

Make sure that you delete the unwanted sprites 25 to 255, 9, 12, 13 and 14 BEFORE loading source code from tape or the source code will over-run the sprites.

LUNAR LANDER

Variables

PH	Horizontal Phase of scrolling landscape
SPD	Horizontal Velocity of strolling landscape
DOWN	Set to 1 if Lander crashes
FU	Remaining fuel
XP	Vertical Position of Lander
VEL	Vertical Velocity of Lander
SX	Phase of X-Velocity dial
SY	Phase of Y-Velocity dial
SFU	Phase of Fuel Gauge
IX	Phase of Horizontal Position Dial

Sprites

NUMBERS	HEIGHT	LENGTH	DESCRIPTION
1	1	2	POINTER
3	1	5	LANDING PAD
4-9	1	1	BLOCKS FOR LANDSCAPE SPRITES
10-25	3	8	LANDSCAPE SECTIONS
26,28,29,31	1	1	PANEL SPRITES
32,33	1	8	MINI LANDSCAPE
43	3	3	EXPLOSION
44	4	5	LANDED LUNAR LANDER
45	3	3	CRASHED LUNAR LANDER
100	6	3	LUNAR LANDER
101-107	6	3	LUNAR LANDER ORIENTATIONS (CONSTRUCTED)
128	3	128	COMPLETE LANDSCAPE (CONSTRUCTED)

Sprites 1 to 100 are produced using the sprite generator program. Sprites 101 to 107 are created in the main program using the word SET. Sprite 128 is constructed from sprites 10 to 25 in the main program using the word MAKE.

The Lunar Lander

This listing is provided as an example of White Lightning programming. In order to run the game you will need to enter the sprites as described in the next section. This is a fairly laborious task so we can offer the fainthearted, sprites and demo on tape for one pound seventy five pence. We recommend, however, that you take the time to build up the sprites yourself as an exercise in self discipline if nothing else!

The program executes one word in foreground and one word in background. The program can be roughly sub-divided the following way:

Screens 6,7,8

These set up the screen display, dials etc.

Screens 9,10,11

These three screens form the routine which scrolls the landscape at one of three speeds. This routine is executed in background to give smoother movement.

Screens 11,12,13,14,15,16,17

These control the flight of the lander, manipulate the dials, execute the crashes and so on.

Screen 18

This executes all the previous definitions in the right order to produce the final game.

Let's now look at the program in more detail.

```
SCR # 6
  0 : COLOUR 0 ROW ! 16 COL ! 16
 LEN ! 23 HGT ! 7 INK 1 BRIGHT
  1 SETAV ATTON ;
  2 : VTSC COL ! 10 2 DO I ROW !
 PUTBLS LOOP ;
  3 : SCLE 26 SPN ! 18 VTSC 26 V
TSC MIRM 21 VTSC 29 VTSC MIRM ;
  4 : VTCL ROW ! COL ! LEN ! HGT
 ! PAPER SETAV ;
  5 : BARS 4 6 1 20 2 VTCL 2 2 1
 20 8 VTCL 5 4  1 28 2 VTCL
  6 2 3 1 28 7 VTCL 5 1 16 16 14
 VTCL ;
  7 -->

SCR # 7
  0 : LND 6 COL ! 17 ROW ! 44 SP
N ! PUTBLS BEGIN 7 1 KB UNTIL
  1 20 ROW ! 43 SPN ! 7 COL ! 20
 0 DO PUTXRS 100 100 BLEEP LOOP
  2 44 SPN ! 17 ROW ! 6 COL ! PU
TXRS ;
  3 : PTST SPN ! COL ! ROW ! PUT
BLS ;
  4 : BARST 14 23 31 PTST MIRM 1
4 24 31 PTST MIRM 6 28 28 PTST
  5 19 16 32 PTST 19 24 33 PTST
;
  6 : LETR 7 INK 0 PAPER 1 18 AT
 ." FUEL" 0 26 AT ." VERT" 1 26
AT
  7 ." VEL" 11 20 AT ." HORZ VEL
" ; -->

SCR # 8
  0 : HRSC 32 16 DO 12 I 29 PTST
 17 I 29 PTST LOOP 16 16 1 PTST
;
  1 : MARK 152 159 PLOT 7 0 DRAW
 216 127 PLOT 7 0 DRAW 128 71 PL
OT
```

```
  2 0 -7 DRAW ;
  3 : PANEL 0 PAPER COLOUR SCLE
BARS BARST HRSC LETR MARK ;
  4 : MAKE 128 SPN ! 3 HGT ! 128
 LEN ! 0 SROW ! 128 SP2 ! ISPRIT
E
  5 16 0 DO I 10 + DUP SPN ! SP1
 ! I 8 * SCOL ! GWBLM GWATIM DSP
RITE
  6 LOOP ;
    0 VARIABLE PH
  7 256 VARIABLE SPD 0 VARIABLE
DOWN 1008 VARIABLE FU -->

SCR # 9
  0 : S1 1023 AND 8 / SCOL ! 1 L
EN ! PUTBLS 16 LEN ! ;
  1 : NBR PH @ S1 ;
    : NBL PH @ 128 + S1 ;
  2 : OPEN 0 PAPER 5 INK CLS 0 P
H ! EXX 128 SPN ! 16 LEN ! 0 COL
 !
  3 3 HGT ! 21 ROW ! 0 SCOL ! 0
SROW ! PWBLS PWATTS 2 HGT ! EXX
  4 0 PAPER 1008 FU ! 0 BORDER ;
    : SH8 PH @ DUP 7 AND 0= ;
  5 : FUEL -1 FU +! ;
    : SR SH8 IF NBR ENDIF ;
  6 : SL SH8 IF NBL ENDIF ;
    : SH4 PH @ DUP 3 AND 0= ;
  7 : SP SPD +! ;
    : SS SPD @  ; -->

SCR # 10
  0 : POLL FU @ IF 8 1 KB IF SS
-252 > MINUS SF FUEL ENDIF
  1 1 1 KB IF SS 256 < SF FUEL E
NDIF ENDIF ;
  2 : -P - PH ! POLL ;
    : +P + PH ! POLL ;
  3 : SR1 SR WRR1V 1 -P ;
    : SL1 SL WRL1V 1 +P ;
  4 : SR4 SH4 IF SR WRR4V 4 -P E
LSE SR1 ENDIF DROP ;
  5 : SL4 SH4 IF SL WRL4V 4 +P E
LSE SL1 ENDIF DROP ;
  6 : SO DOWN @ IF ELSE SS ABS 2
56 < IF POLL ENDIF ENDIF ;
  7 : SR8 SH8 IF NBR WRR8V 8 -P
ELSE SR4 ENDIF DROP ; -->

SCR # 11
  0 : SL8 SH8 IF NBL WRL8V 8 +P
ELSE SL4 ENDIF ;
  1 : UR SS 200 > IF SR8 ELSE SR
4 ENDIF ;
  2 : LR SS 7 > IF SR1 ELSE SO E
NDIF ;
  3 : RT SS 100 > IF UR ELSE LR
ENDIF ;
  4 : UL SS -200 < IF SL8 ELSE S
L4 ENDIF ;
  5 : LL SS -7 < IF SL1 ELSE SO
ENDIF ;
  6 : LF SS -100 < IF UL ELSE LL
 ENDIF ;
  7 : DEC SS 0< IF LF ELSE RT EN
DIF ;

SCR # 12
  0 : SET -1 NPX ! 3 LEN ! 6 HGT
 ! 107 100 DO I SP1 ! I 1+ DUP S
P2 !
  1  SPN ! ISPRITE COPYM WCRM LO
OP ;
  2 40 VARIABLE XP 8 VARIABLE V
EL
  3 : PREP 7 COL ! 0 DOWN ! 40 X
P ! ;
  4 : TICK VEL @ 255 > IF ELSE 1
VEL +! ENDIF ;
  5 : THRUST FU @ IF 7 1 KB IF V
EL @ -252 > IF -4 VEL +! FUEL
  6 ENDIF ENDIF ENDIF ;
  7 -->

SCR # 13
  0 : MV VEL @ XP @ + DUP 5631 >
 IF DROP 5631 1 DOWN ! ENDIF DUP
  1 XP ! 32 / 8 /MOD 5 - ROW ! 7
 AND 100 + SPN ! ROW @ 0< IF ADJ
M
  2 PWBLS ELSE VEL @ 0< IF 1 SRO
W ! ROW @ 15 > IF 4 HGT ! ELSE
  3 5 HGT ! ENDIF ELSE 0 SROW !
5 HGT ! ENDIF ROW @ DUP SROW @ +
  4 ROW ! PWBLS ROW ! ENDIF ;
  5
  6
  7 -->

SCR # 14
  0 : BANG DOWN @ DUP IF 19 ROW
! 43 SPN ! -5 NPX ! 7 HGT ! 3 LE
N !
  1 40 10 DO PUTXRS I 20 + I DO
J I BLEEP LOOP PUTXRS 17 ROW !
  2 SCRV 19 ROW ! 5 +LOOP SS ABS
 8 < IF 21 ROW ! 45 SPN ! PUTBLS
  3 ENDIF ENDIF XP @ 5631 = IF D
OWN @ 0= IF 7 COL ! LND 7 COL !
  4 MV 0 VEL ! ENDIF INT-OFF BEG
IN 7 1 KB UNTIL
  5  ' DEC INT-ON ENDIF ;
  6 : OK 0 DOWN ! ;
  7 -->

SCR # 15
  0 : LAND SS ABS 8 < IF VEL @ 3
2 < IF PH @ 1023 AND 8 / CASE 12
 OF
```

```
  1 OK ENDOF 13 OF OK ENDOF 30 O
F OK ENDOF 31 OF OK ENDOF 58 OF
OK
  2 ENDOF 59 OF OK ENDOF 91 OF O
K ENDOF 92 OF OK ENDOF ENDCASE
  3 ENDIF ENDIF BANG ;
  4 128 VARIABLE SX 32 VARIABLE
SY 63 VARIABEL SFU
  5 : XG SX +! 16 COL ! 13 ROW !
 1 HGT ! 16 LEN ! ;
  6 : RSET1 7 COL ! 3 LEN ! ;
    : WLEFT -1 XG WRR1V RSET1 ;
  7 : WRIGHT 1 XG WRL1V RSET1 ;
     -->

SCR # 16
  0 : XVEL SPD @ 256 + 4 / SX @
 - DUP 0< IF WLEFT DROP ELSE 0 >
IF
  1 WRIGHT ENDIF ENDIF ;
  2 : YG DUP MINUS NPX ! SY +! 2
7 COL ! 2 ROW ! 8 HGT ! 1 LEN !
  3 WCRV RSET1 ;
  4 : WUP -1 YG ;
    : WDOWN 1 YG ;
  5 : YVEL VEL @ 256 + 8 / SY @
- DUP 0< IF WUP DROP ELSE 0 > IF
  6 WDOWN ENDIF ENDIF ;
  7 -->

SCR # 17
  0 : FVEL FU @ 16 / SFU @ - 0<
IF 19 COL ! 2 ROW ! 8 HGT ! 1 LE
N !
  1 -1 NPX ! -1 SFU   +! WCRV RSE
T1 ENDIF ;
  2 0  VARIABLE LX
  3 : MXG 18 ROW ! 16 COL ! 16 L
EN ! 1 HGT ! ;
  4 : MLEFT MXG WRL1V -1 LX +! ;
    : MRIGHT MXG WRR1V 1 LX +! ;
  5 : MON PH @ 8 / LX @ - DUP 0
> IF DROP MRIGHT ELSE
  6 0< IF MLEFT ENDIF ENDIF RSET
1 ;
  7 -->

SCR # 18
  0 : OFF PANEL PREP ' DEC INT-O
N BEGIN TICK THRUST MV MON XVEL
  1 YVEL FVEL MON LAND MON UNTIL
 INT-OFF ;
  2 : TST 256 SPD ! 0 PH ! 1008
FU ! 40 XP ! 8 VEL ! 128 SX !
  3 32 SY ! 63 SFU ! 0 LX ! OPEN
 OFF ;
  4
  5
  6
  7
```

WORD DESCRIPTIONS

COLOUR

Sets the attributes in the right hand half of the screen.

VTSC

Produces a row of sprites with the current sprite number at the column on the stack. Used to build up the gauges.

SCLE

Uses VTSC to build up the gauges.

VTCL

Sets up a specified window with specified attributes.

BARS

Uses VTCL to set the attributes for the gauges.

LAND

This word controls the landing sequence. The landed sprite is placed on the pad and sits until ENTER is pressed. The explosion is then produced beneath the lander to simulate take off and the landed lander is then exclusively OR'ed off of the pad.

PTST

General purpose word which sets ROW, COL and SPN from the stack and then performs a PUTBLS.

BARST

Adds the finishing touches to the gauges by putting sprites 28,31,32 and 33 in the appropriate positions. Uses PTST.

LETR

Places the gauge titles above the gauges.

HRSC

Puts the horizontal scale on the screen.

MARK

Draws the indicators used in the gauges.

PANEL

Execution word to set up the whole right hand side of the screen ie. all previous words.

MAKE

See 'Creating Large Sprites'.

S1

Used to calculate the next column in the large landscape sprite, to be put to the screen. LEN is set back to 16 for the next operation.

NBR

Gets the appropriate column (calculated by S1) when the landscape is moving right.

NBL

Gets the appropriate column when the landscape is moving left.

OPEN

Builds up the initial picture for the left of the screen. Notice the use of EXX to set up background variables for execution under interrupt.

SH8

PH describes the phase of the landscape with pixel resolution. SH8 checks to see if this phase is a multiple of 8 and if so sets a flag to indicate that a fresh column should be GOT from the landscape sprite. PH is also left on the stack.

FUEL

Decrements the amount of fuel left.

SR

Checks to see if a character boundary is crossed (see SH8) and if so executes NBR.

SL

As SR but checks when movement is left.

SH4

Checks to see if phase is crossing a half character boundary, scrolling can only increase from one pixel to four pixel movements on such a boundary or scrolling will go out of phase.

SF

Accelerates horizontal speed (decelerates if negative) by the amount on the stack.

SS

Puts the current horizontal speed on the stack.

POLL

Checks to see if there's any Fuel left (a non zero value of FU will act as a true flag) then first checks SPACE to accelerate right if not travelling too fast then check CAPS SHIFT to accelerate left if not travelling too fast. If a key is pressed fuel is decremented.

-P

Updates phase and does a POLL when moving right.

+P

Updates phase and does a POLL when moving left.

SR1

Moves landscape 1 pixel right and adjusts pointers.

SR4

Moves landscape 4 pixels right and adjusts pointers provided a half character boundary has been reached. If not, a further 1 pixel movement must be made.

SL4

As SR4 when moving left.

SO

When speed is less than 7, keyboard is polled but no scrolling of the screen is executed. The lander is treated as horizontally stationary.

SR8

Moves landscapes 8 pixels (1 character) right and adjusts pointers provided a full character boundary has been reached. If not a further 4 pixel scroll is executed.

SL8

As SR8 when movement is to the left.

UR

If speed is greater than 200, try and scroll right 8 pixels right, if not, try and scroll by 4 pixels right.

LR

If speed is greater than 7 then scroll 1 pixel right else no scroll.

RT

If speed is greater than 100 then do a UR if not do an LR.

UL

As UR when speed is negative.

LL

As LR when speed is negative.

LF

As RF when speed is negative.

DEC

The execution word which does all the scrolling logic. The words UR to LF are effectively the nodes of a tree which produce one of 7 possible scrolls from -8 pixels to +8 pixels. A detailed understanding of the workings are not necessary as long as you can adapt the routine to serve your needs.

SET

Creates 7 new orientations of the lander from the original in sprite 100, making 8 in all, each lander being 1 pixel shifted vertically from the one before. This enables single pixel resolution in the lander movement.

PREP

Used to set up initial values.

TICK

Increments vertical velocity (acts like gravity) unless terminal velocity has been reached. If you want to make the game more difficult change 1 VEL +! to 2 VEL +! and thus double the planet's gravity.

THRUST

If the lander still has fuel and hasn't reached terminal upward velocity then increase upward velocity.

MV

A fairly complicated word which moves the lander vertically. The velocity is added to the position (physicists note that unit time has elapsed etc). If the lander goes under the base its position is put equal to the base and DOWN is SET to 1. The row and orientation are then calculated.

BANG

Another fairly involved word which executes a crash if DOWN=1. It checks to see if a safe landing was made and if not decides what sort of crash is required.

OK

A short word which sets DOWN to 0, indicating a safe landing.

LAND

If the lander has zero sideways velocity then vertical velocity and horizontal position are checked for a safe landing or a crash. DOWN is set accordingly.

XG

Updates X-VEL PHASE and sets window for scroll.

RSET1

Sets COL and LEN back after XG.

XVEL

Controlling routine for XVEL gauge.

YG

Used to adjust Y-VEL gauge.

WUP

Move up Y-VEL gauge.

WDOWN

Move down Y-VEL gauge.

YVEL

Controlling routine for Y-VEL gauge.

FVEL

Controlling routine for fuel gauge.

MXG

Used to set up window for small screen movement.

MLEFT

Move pointer left on small screen.

MRIGHT

Move pointer right on small screen.

MON

Control routine for scroll screen.

OFF

Main program loop.

TST

Final execution word. Initialises parameters and then executes main program loop.

THE GAME ITSELF

CREATION OF SPRITES

Load up the sprite development package and create all the sprites listed in table 1.

Once the sprite development package has loaded execute a cold start by pressing the C key and hit Y for yes and then N for the change buffer size prompt.

Set the attribute switch to 1 by pressing the A key and then 1.

With reference to table 1 set the sprite number to the required value by pressing the S key and then inputting the required value. Input the dimensions (Height and Length) of the sprite by pressing the H or C keys and then inputting the appropriate values. Set the respective ink, paper, flash and bright values using the X, C, V and B keys.

Position the sprite screen X and Y pos cursors to their settings using the symbol shift 5, 6, 7 or 8 keys.

Using the direct data input function, key D, input the 8 bytes of data. Move the X and Y pos cursors to the next position and input the data until the sprite is complete on the screen. Set both the X and Y position cursors to 1 and then GET the sprite into memory by pressing the G key. Clear the sprite screen by pressing symbol shift Q and then create the next sprite.

CREATION OF THE 64x3 CHARACTER LANDSCAPE SPRITE

This sprite will be made up of 8 8x3 character sprites, which will be joined together into one large sprite in the White Lightning program itself.

Set the ink to 7, the paper to 0, the flash to 0 and the bright to 1. Clear the CHR$ square by pressing the Q key. Using the sprites 4,5,6,7,8,9 and the CHR$ square (referred to as 0) build up sprites 10 to 25 as laid out in diagram 1.

Position the X and Y Pos cursors to the appropriate co-ordinates. Input the relevant sprite number and put the sprite to the screen by pressing the D key and then 1. In the case of 0 press J to DUMP the empty CHR$ square to the sprite screen.

The landing pad, sprite 3, is also placed in these sprites.

Once the 8 by 3 sprites have been created on the screen, position the X and Y pos cursors to the top left corner of the sprite. Set the sprite number to the appropriate value, set the length to 8 and the height to 3 then press G to get the sprite.

Note that the left hand column of sprite 10 must have a 0 ink value as well as the right hand column of sprite 11.

Once all the sprites have been created save them off to tape using the symbol SHIFT S key.

Load in White Lightning, load in the lunar sprites. Carefully type in the Lunar Lander program as listed and check your program against the original. It is now best to save your source off to tape.

```
Exit to BASIC using:                PROG <CR>

then save to tape using:            SAVE"LUNAR" CODE 52224,6656

Go back into White Lightning using: PRINT USR 24836

Now type     6 LOAD <CR>
             MAKE <CR>  to create the landscape
             SET  <CR>  to create the landers

To run the program type  TST <CR>
```

Please note that if there is an error in your source the last few screens can no longer be listed or compiled, since the creation of extra sprites has overwritten the end screens, thus the source would have to be reloaded for editing purposes.

PLAYING THE GAME

The game itself is more of a simulation than a game. The idea is to land on all four bases without running out of fuel or crashing. The gauges are self explanatory.

The controls are:		
	CAPS SHIFT	Thrust to the left
	BREAK SPACE	Thrust to the right
	ENTER	Vertical thrust

Once the game is over, hit the ENTER key to escape and then TST <CR> for a new game.

FUEL

If fuel runs out the controls will no longer function.

YVEL

A safe landing is only made if the VEL gauge registers a velocity in the "safe" region of the centre of the gauge.

XVEL

The horizontal velocity is represented by one of 3 scroll speeds but safe landings can only be made if the pointer is in the "safe" region in the centre of the gauge.

THE SMALL SCREEN

This gives a macroscopic view of the 8 screens. The bases are marked. The gauge sometimes cannot keep up with the lander movement but at scroll speeds of a pixel it will soon "catch up" with the real positions. This is not a "feature" we must admit, keeping up with the gauge slows the foreground program down a lot.

TABLE 1

SPRITE	H	L	INK	PAPER	FLASH	BRIGHT	XPOS	YPOS	1	2	3	4	5	6	7	8
1	1	2	7	0	0	1	1	1	29	29	28	15	7	3	0	0
1	1	2	7	0	0	1	2	1	92	92	28	120	112	96	128	128
3	1	6	7	0	0	1	1	1	255	227	127	0	60	24	60	126
3	1	6	7	0	0	1	2	1	255	142	255	63	31	63	31	63
3	1	6	7	0	0	1	3	1	255	28	255	255	255	255	255	255
3	1	6	7	0	0	1	4	1	255	28	255	255	255	255	255	255
3	1	6	7	0	0	1	5	1	255	113	255	254	252	254	252	254
3	1	6	7	0	0	1	6	1	255	199	254	0	60	24	60	126
4	1	1	7	0	0	1	1	1	128	64	160	80	168	84	170	85
5	1	1	7	0	0	1	1	1	0	1	2	5	10	21	42	85
6	1	1	7	0	0	1	1	1	170	85	170	85	170	85	170	85
7	1	1	7	0	0	1	1	1	128	65	162	85	170	85	170	85
8	1	1	7	0	0	1	1	1	0	0	0	0	8	20	42	85
9	1	1	7	0	0	1	1	1	0	0	0	0	0	16	40	85
26	1	1	6	0	0	1	1	1	254	2	2	2	30	2	2	2
28	1	1	2	4	0	1	1	1	0	0	0	0	255	255	255	255
29	1	1	6	0	0	1	1	1	128	128	128	136	136	136	255	0
31	1	1	5	1	0	1	1	1	254	254	254	254	254	254	254	254
32	1	8	3	0	0	1	1	1	0	1	155	155	255	255	255	255
32	1	8	3	0	0	1	2	1	0	0	226	230	255	255	255	255
32	1	8	3	0	0	1	3	1	63	33	127	127	255	255	255	255
32	1	8	3	0	0	1	4	1	0	0	59	251	255	255	255	255
32	1	8	3	0	0	1	5	1	3	2	223	223	255	255	255	255
32	1	8	3	0	0	1	6	1	240	16	254	254	255	255	255	255
32	1	8	3	0	0	1	7	1	0	24	127	127	255	255	255	255
32	1	8	3	0	0	1	8	1	0	0	231	231	255	255	255	255
33	1	8	3	0	0	1	1	1	126	66	254	255	255	255	255	255
33	1	8	3	0	0	1	2	1	0	16	126	126	255	255	255	255
33	1	8	3	0	0	1	3	1	0	0	143	143	255	255	255	255
33	1	8	3	0	0	1	4	1	0	0	254	254	255	255	255	255
33	1	8	3	0	0	1	5	1	126	66	127	255	255	255	255	255
33	1	8	3	0	0	1	6	1	18	18	255	255	255	255	255	255
33	1	8	3	0	0	1	7	1	0	0	255	255	255	255	255	255
33	1	8	3	0	0	1	8	1	0	0	167	167	255	255	255	255

TABLE 1 (Continued)

SPRITE	H	L	INK	PAPER	FLASH	BRIGHT	XPOS	YPOS	1	2	3	4	5	6	7	8
43	3	3	6	0	0	1	1	1	96	114	88	46	181	26	21	90
43	3	3	6	0	0	1	2	1	16	41	88	104	212	174	85	170
43	3	3	6	0	0	1	3	1	128	2	14	20	40	208	98	224
43	3	3	6	0	0	1	1	2	13	10	53	234	61	10	21	42
43	3	3	6	0	0	1	2	2	85	170	85	170	85	170	85	170
43	3	3	6	0	0	1	3	2	64	160	80	174	120	192	64	226
43	3	3	6	0	0	1	1	3	85	170	255	2	70	13	42	12
43	3	3	6	0	0	1	2	3	85	170	89	168	208	49	16	0
43	3	3	6	0	0	1	3	3	96	180	168	216	40	20	74	7
44	4	5	5	0	0	1	1	1	0	0	0	0	0	0	0	0
44	4	5	5	0	0	1	2	1	0	0	0	0	0	0	0	0
44	4	5	5	0	0	1	3	1	0	0	0	0	0	0	0	0
44	4	5	5	0	0	1	4	1	0	0	0	0	0	0	0	0
44	4	5	5	0	0	1	5	1	0	0	0	0	0	0	0	0
44	4	5	5	0	0	1	1	2	0	0	0	0	0	0	0	0
44	4	5	5	0	0	1	2	2	0	0	0	0	1	1	7	6
44	4	5	5	0	0	1	3	2	0	0	0	0	165	255	66	36
44	4	5	5	0	0	1	4	2	0	0	0	0	128	128	224	96
44	4	5	5	0	0	1	5	2	0	0	0	0	0	0	0	0
44	4	5	5	0	0	1	1	3	0	0	0	0	0	0	0	0
44	4	5	5	0	0	1	2	3	12	15	5	0	7	13	213	47
44	4	5	5	0	0	1	3	3	24	255	90	0	255	90	90	255
44	4	5	5	0	0	1	4	3	48	240	160	0	224	176	171	244
44	4	5	5	0	0	1	5	3	0	0	0	0	0	0	0	0
44	4	5	5	0	0	1	1	4	0	0	1	6	9	22	72	48
44	4	5	5	0	0	1	2	4	237	221	127	140	48	192	0	0
44	4	5	5	0	0	1	3	4	36	165	255	129	44	94	191	191
44	4	5	5	0	0	1	4	4	183	187	254	49	12	3	0	0
44	4	5	5	0	0	1	5	4	0	0	128	96	144	104	18	12
45	3	3	7	0	0	1	1	1	0	0	18	9	9	55	33	65
45	3	3	7	0	0	1	2	1	0	0	12	24	56	112	231	207
45	3	3	7	0	0	1	3	1	0	32	80	8	4	58	196	201

TABLE 1 (Continued)

SPRITE	H	L	INK	PAPER	FLASH	BRIGHT	XPOS	YPOS	1	2	3	4	5	6	7	8
45	3	3	7	0	0	1	1	2	67	71	46	60	56	17	131	64
45	3	3	7	0	0	1	2	2	191	44	78	187	119	46	25	123
45	3	3	7	0	0	1	3	2	18	37	194	5	138	197	138	5
45	3	3	7	0	0	1	1	3	160	81	162	82	170	84	170	85
45	3	3	7	0	0	1	2	3	6	230	248	248	244	163	82	85
45	3	3	7	0	0	1	3	3	42	21	42	85	170	85	170	85
100	6	3	5	0	0	1	1	1	0	0	0	0	0	0	0	0
100	6	3	5	0	0	1	2	1	0	0	0	0	0	0	0	0
100	6	3	5	0	0	1	3	1	0	0	0	0	0	0	0	0
100	6	3	5	0	0	1	1	2	1	1	7	6	12	15	5	0
100	6	3	5	0	0	1	2	2	165	255	66	36	24	255	90	0
100	6	3	5	0	0	1	3	2	128	128	224	96	48	240	160	0
100	6	3	5	0	0	1	1	3	7	13	213	47	237	221	11	21
100	6	3	5	0	0	1	2	3	255	90	90	255	36	165	255	129
100	6	3	5	0	0	1	3	3	224	176	171	244	183	187	208	168
100	6	3	5	0	0	1	1	4	34	68	248	144	160	64	144	96
100	6	3	5	0	0	1	2	4	44	94	191	191	0	0	0	0
100	6	3	5	0	0	1	3	4	68	34	31	9	5	2	9	6
100	6	3	5	0	0	1	1	5	0	0	0	0	0	0	0	0
100	6	3	5	0	0	1	2	5	0	0	0	0	0	0	0	0
100	6	3	5	0	0	1	3	5	0	0	0	0	0	0	0	0
100	6	3	5	0	0	1	1	6	0	0	0	0	0	0	0	0
100	6	3	5	0	0	1	2	6	0	0	0	0	0	0	0	0
100	6	3	5	0	0	1	3	6	0	0	0	0	0	0	0	0

DIAGRAM 1

0 INK ↓ (column 4)

	4	5	6	7	8	9	A	B
6	4	0	0	8	9	0	5	6
7	6	4	5	6	6	7	6	6
8	6	6	6	6	6	6	6	6

SPRITE 10

↓ 0 INK (column B)

	4	5	6	7	8	9	A	B
6	4	8	9	0	0	9	8	0
7	6	6	6	4	5	6	6	7
8	6	6	6	6	6	6	6	6

SPRITE 11

	4	5	6	7	8	9	A	B
6	0	9	SPRITE 3					
7	7	6	6	6	6	6	6	6
8	6	6	6	6	6	6	6	6

SPRITE 12

	4	5	6	7	8	9	A	B
6	9	0	5	4	8	0	8	8
7	6	6	6	6	6	7	6	6
8	6	6	6	6	6	6	6	6

SPRITE 13

	4	5	6	7	8	9	A	B				
6	5	4	0	9	5	4	SPRITE 3					
7	6	6	7	6	6	6	6	6				
8	6	6	6	6	6	6	6	6				

SPRITE 14

	2	3	4	5	6	7	8	9	A	B
6	SPRITE 3						8	5	4	0
7			6	6	6	6	6	6	6	4
8			6	6	6	6	6	6	6	6

SPRITE 15

	4	5	6	7	8	9	A	B
6	0	9	5	6	6	4	8	9
7	5	6	6	6	6	6	6	6
8	6	6	6	6	6	6	6	6

<u>SPRITE</u> 16

	4	5	6	7	8	9	A	B
6	5	4	8	0	0	5	7	4
7	6	6	6	7	7	6	6	6
8	6	6	6	6	6	6	6	6

<u>SPRITE</u> 17

	4	5	6	7	8	9	A	B
6	9	**SPRITE 3**						0
7	6	6	6	6	6	6	6	4
8	6	6	6	6	6	6	6	6

<u>SPRITE</u> 18

	4	5	6	7	8	9	A	B
6	0	8	5	6	4	8	9	0
7	5	6	6	6	6	6	6	7
8	6	6	6	6	6	6	6	6

<u>SPRITE</u> 19

	4	5	6	7	8	9	A	B
6	8	0	0	0	5	7	7	4
7	6	7	4	5	6	6	6	6
8	6	6	6	6	6	6	6	6

<u>SPRITE</u> 20

	4	5	6	7	8	9	A	B
6	8	5	4	5	4	8	9	0
7	5	6	6	6	6	6	6	4
8	6	6	6	6	6	6	6	6

<u>SPRITE</u> 21

DIAGRAM 1 CONTINUED

	4	5	6	7	8	9	A	B
6	0	**SPRITE 3**						9
7	5	6	6	6	6	6	6	6
8	6	6	6	6	6	6	6	6

SPRITE 22

	4	5	6	7	8	9	A	B
6	8	9	5	6	7	7	6	4
7	6	6	6	6	6	6	6	6
8	6	6	6	6	6	6	6	6

SPRITE 23

	4	5	6	7	8	9	A	B
6	9	5	4	8	9	9	5	7
7	6	6	6	6	6	6	6	6
8	6	6	6	6	6	6	6	6

SPRITE 24

	4	5	6	7	8	9	A	B
6	4	0	8	0	0	8	5	7
7	6	7	6	4	5	6	6	6
8	6	6	6	6	6	6	6	6

SPRITE 25

INTRODUCTION

In order to make maximum use of the Spectrum's 48k of memory the tape based White Lightning was located at 24832 decimal. This leaves room for a small BASIC loader program. If, however, Interface 1 is fitted, the execution of any of the shadow ROM commands will cause BASIC to be relocated upwards and result in insufficient memory for the tape based loader program.

It was therefore decided, on completion of the tape based program, to develop another version which would not only be microdrive compatible but would also utilise the drives to compile source code. Unfortunately, because of the way that the shadow ROM operates, the BASIC interface part of the software is no longer practical and therefore Microdrive White Lightning no longer has this feature.

The editing buffers have now been moved up to occupy the old screens 6, 7 and 8, and an extra 1k of dictionary space is now available. Source code is now compiled directly from microdrives and so sprites can be stored from 53760 onwards. Microdrive screens 1 to 19 are utilised by the system but you can edit any of the screens 20 to 150. This means that Microdrive White Lightning can handle six times as much source code without any troublesome reloading from tape.

IMPLEMENTING ON MICRODRIVE

1. Insert Tape 1 in your cassette and rewind to Side A.

2. Type LOAD"" and when "MWL" has loaded it will auto-run, format your microdrive and save the BASIC loader and the machine code.

3. To run the microdrive version just type LOAD *"M";1;"MWL" and it will load and execute.

PREPARING A CARTRIDGE FOR SOURCE CODE

Before using the microdrive version it is necessary to set up a separate cartridge for storing and loading source code. This version has been designed to work exclusively with microdrive number 1.

To set up the cartridge, insert it in microdrive 1 and execute the following:

```
FORMAT"M";1;"name":OPEN#4;"M";1;"a":
FOR I=1 TO 100000: PRINT#4;CHR$ 32 ; : NEXT I
```

Note the lower case "a" used in the filename.

After several minutes, the error message "MICRODRIVE FULL" will be printed. You should now key in:

```
CLOSE#4
```

to close this file. To check that you have a correctly prepared cartridge, type:

```
CAT 1
```

"a" should be printed on the screen to indicate a single file called "a" and 0 to indicate zero bytes free.

Please note that this cartridge should be clearly labelled and used exclusively for SAVEing and LOADing Forth source code while you are editing Forth screens. Sprites, BASIC and blocks of machine code should be SAVEd on a separate cartridge or cartridges. There are no commands within White Lightning to manipulate microdrives, therefore all such commands (e.g. FORMAT, ERASE, OPEN#) are executed from BASIC after exiting via the PROG command.

EDITING FORTH SCREENS

Forth source code is still EDITed in exactly the same way as the tape version (using EDIT,P,S,D etc.) except that EDIT will not automatically execute a FLUSH. This means that your EDITs will not be updated on microdrive until you type FLUSH, so be sure to remember to do this before moving on to another screen.

THE EDIT BUFFERS

To give some idea of how White Lightning uses the microdrive for Forth source, the following brief description may be helpful.

There exists in RAM an area of 1024 bytes called the edit buffer, which can hold two 512 byte Forth screens. If you issue a command which requires the use of a screen (LIST, CLEAR or INDEX) then this screen will be read from the microdrive into the edit buffer. If the edit buffer already contains two screens and they have been altered in any way since they were last loaded, then they must first be saved (using FLUSH) back to the drive in order to allow the currently required screen to be loaded in.

Note that before a screen is first edited it will need to be cleared using the CLEAR command as it will probably contain garbage. For example, before using screen 20 for the first time, type:

```
20 CLEAR 20 LIST
```

TRANSFERRING OLD SOURCE

If you have already written a sizeable program with the tape based White Lightning, then you will want to transfer it to your microdrive-based White Lightning without having to completely re-type it. To do this, use the following:

1. Type OLD <CR>.

2. Type PROG <CR> and load your old source code from the tape in the normal way. Then re-enter Forth as normal.

3. Transfer each screen from its old number in memory to its new number on the microdrive using:

   OLDSCREEN NEWSCREEN TRANS <CR>

   So, for example, to transfer the old screen 6 to microdrive screen 25, use:

   6 25 TRANS <CR>

   If a bad sector is encountered you will get error message 8. Skip over this sector and try the next one (see next section on BAD SECTORS).

4. Finally type NEW <CR> to restore the editing buffers to their microdrive addresses.

BAD SECTORS

Regrettably, at the time of writing, whilst the microdrive cartridge costs about twice as much as a standard 5 1/4 inch floppy, the number of bad sectors is still extremely high. Forth screens map directly to microdrive sectors, so screen 25 uses sector 25 and so on. So as we shall see, some screens may be unusable.

A bad sector will be identified by White Lightning the first time a read or write operation is carried out, and error 8 generated. Note that executing CLEAR will mean that you can find bad sectors before editing into them. If you do find a bad sector, keep a note of it and don't use that screen. There is a simple way around this problem as we shall see in the following example. Assume we want to edit into screens 25 to 28 and that we did the following:

1. Type 25 CLEAR 0 EDIT

2. Key in text for lines 0 to 6.

3. Type 7 EDIT then key in --> to indicate continue with next screen when LOADing.

4. Type 26 CLEAR 0 EDIT.

5. Key in text for lines 0 to 6.

6. Type 7 EDIT and key in --> .

Now suppose when we typed in 27 CLEAR, that we got error 8, indicating drive error.
This would mean that sector 27 was a bad sector and therefore that screen 27 was unusable.

Remember that in screen 26 the last line was:

```
7 -->
```

which tells Forth to continue LOADing on the next screen. The next screen is screen 27 which is unusable, so we have to change the last line of screen 26 to become:

```
7 28 LOAD
```

which tells Forth to continue LOADing at screen 28. This will then skip over the bad sector.

ADDITIONAL ERROR MESSAGES

3 - Incorrect Addressing Mode

7 - Stack Overflow

8 - Microdrive read/write error (bad sector).

SUMMARY OF MICRODRIVE WHITE LIGHTNING

1. Only use specially prepared cartridges for EDITing and use them exclusively for storing screens.

2. CLEAR a screen before using it for the first time and change the previous screen to skip over it if it is a bad sector. Do not use this screen again.

3. Microdrive commands such as ERASE, VERIFY etc. can be executed after entering BASIC using PROG.

4. Do NOT break into the program (SHIFT and SPACE) while the microdrive is running.

5. RESERVE no longer executes.

6. There are some additional error messages (listed above).

7. Always execute a FLUSH after editing of a screen is complete.

THE MICRODRIVE SPRITE GENERATOR

INTRODUCTION

The microdrive Sprite Generator Program is upwardly compatible with the current tape based version and tape LOADing and SAVEing is still supported. An extra command has been added to make the creation of large sprites easier and the arcade character set has been re-organised to give extra sprite space.

IMPLEMENTING ON MICRODRIVE

The first thing to do is to transfer the program onto a microdrive cartridge.

1. Insert Tape 1 in your cassette and rewind to Side A.

2. Type LOAD"":LOAD""CODE

3. Place a formatted cartridge in microdrive 1.

4. Type GOTO 9998.

This will save and verify the generator onto the microdrive. Now type PRINT USR 0 to clear memory.

The microdrive version can now be LOADed and RUN by typing:

```
LOAD *"M";1;"S"
```

SAVING AND LOADING SPRITES

The microdrive version of the Sprite Development Program still allows sprites to be loaded and saved from and to tape as described in the manual.

A separate cartridge is required to store sprites. The program will allow you to save five files of sprites per cartridge, these being numbered 1 to 5.

Before a cartridge can be used to store sprites, it has to be specially formatted. This is done using the Sprite Generator Program by typing SYMBOL SHIFT F (TO). This will format the cartridge and set up five dummy files, numbered 1 to 5. From now on, whenever you save a file of sprites, the old file of that number will be erased to conserve cartridge storage space.

For example, if you wished to save a file of sprites currently in memory, to file 1, use:

1. Type SYMBOL SHIFT S (save sprites).

2. Type N (we don't want tape).

3. Type Y (save to drive).

4. Insert the formatted cartridge.

5. Press any key.

6. Type 1 (save to file 1).

To load sprites just press SYMBOL SHIFT J and then follow the same sequence as that used to save.

AVAILABLE MEMORY

You have 13595 bytes available for sprites. Please note that the bottom 2816 bytes, locations 51685 to 54501 are used to store the arcade character library accesses by the 'Z' key. If, by creating lots of sprites, you overwrite this area of memory, you should not try to access any of these characters.

CREATION OF LARGE SPRITES

The microdrive version of the Sprite Development Package allows the creation of large sprites (larger than the 15x15 screen) in memory. These sprites can be said to be empty when created and have to be filled by placing smaller sprites into them using the 'place sprite into sprite window' function (BREAK SPACE key).

To create a large sprite hit CAPS SHIFT C and enter the dimensions as instructed.

MERGING SPRITES FROM MICRODRIVE

The actual microdrive file that contains the sprite data has the capital letter B after it (CHR$ 66) such that, for example, the sprite data for sprite file '5' is file '5B'.

So with reference to line 5 on page 81 of the White Lightning manual: to merge from microdrive the sprites of sprite file 5 in White Lightning, exit to BASIC and then type:

```
LOAD *"M";1;"5B"CODE
```

TRANSCRIBER'S NOTE

While OCRing and proofreading the scans of the White Lightning manuals, I have tried to preserve, as best as I was able, the original pagination, layout, spacing and formatting of the originals - while still incorporating all errata noted and published in previous versions, and with a few (small) edits and corrections of my own.

This transcription is still a work in progress. If you discover any further mistakes, please create a pull request (or otherwise report the errors) against the manuals' source repository at:

https://github.com/richmilne/white-lightning-manuals