SPRITE UTILITIES

A game writer's utility pack which allows high speed arcade games to be written in Basic.

This is achieved by using the set of supplied machine code sprite routines to move multicoloured characters (sprites), of your own design, around the screen at high speed.

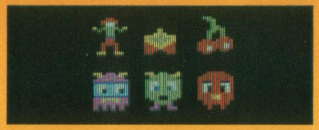
Control of the sprites' movements is by the user-written Basic program. Specific commands to the sprites are very simple.

Special routines are included to detect collisions between sprites and other objects. This is an essential feature for games writing, but is found on few sprite utility packages.

Sprites are generated in mode 5 on a 9 × 16 grid and may include any of the four available colours.

A special super sprite facility allows clones of each sprite to be created, to simulate animation.

The pack contains full instructions and many demonstration programs.



BEEBUGSOFT, PO Box 109, High Wycombe, Bucks HP11 2TD.

SPRITE UTILITIES

BEEBUGSOFT

SPRITE UTILITIES

A Game Writer's Utility Pack

For the Acorn Electron

CASSETTE VERSION

BEEBUGSOFT

SPRITE UTILITIES

A Graphics Package for the Acorn Electron

by
Stephen Allen

BEEBUGSOFT

A game writer's utility pack which allows high speed arcade games to be written in BASIC.

This is achieved by using the set of supplied machine code sprite routines to move multicoloured characters (sprites) of your own design, around the screen at high speed.

Control of the sprites' movements is by the user-written BASIC program. Specific commands to the sprites are very simple.

Special routines are included to detect collisions between sprites and other objects. This is an essential feature for games writing, but is found on few other sprite packages.

Sprites are generated in mode 5 on a 9 x 16 grid and may include any of the four available colours.

A special super sprite facility allows clones of each sprite to be created, to simulate animation.

The pack contains full instructions and many demonstration programs.

BEEBUGSOFT, PO Box 109, High Wycombe, BUCKS HP11 2TD

CONTENTS

1. INTRODUCTION

The major difference between games written in BASIC (or other high level languages) and those written in assembly language is their speed of operation. This is partly due to their graphics handling. In BASIC it is simply not possible to plot intricate objects quickly; even the method of plotting user defined graphics on the screen is clumsy since each colour must be plotted separately. The usual result is that BASIC games tend to have slow jerky graphics, or the game must be kept very simple. Sprites provide an easy way of generating smooth multi-coloured graphics.

The sprite package contains routines for creating and moving sprites around the screen. Each sprite has four images to give enhanced animation. There are also routines to check for collisions, read direction keys, swap images and many other features. All these routines are easily accessible from BASIC or machine code. The package also contains a set of example programs.

2. FEATURES OF THE BEEBUGSOFT SPRITE PACKAGE

1. Up to forty-eight can be quickly created on a 9 by 16 grid, using a special sprite definer program. Each of the 9 by 16 (144) individual parts which make up the sprite may be any of the four colours available in Mode 5.

2. Each sprite is easily displayed or moved using the resident integer variables (A% to Z%). For example the following sequence:

   ```
   W%=2:A%=300:B%=500:Y%=0:CALL S%
   ```

   will draw sprite 2 at location (300,500) on the screen. This corresponds to the normal graphic plotting scale.

3. Plotting a sprite at a new position will automatically delete its previous image, leaving the background unchanged.

4. The sprite routines permit the use of an automatic wrap-around screen. This means that a sprite moving off the left of the screen will reappear on the right, and vice versa. Similarly a sprite moving off the top appears at the bottom, and vice versa.

5. A special machine code routine will check for any collisions with a sprite. Not only will it return the identity of the sprite with which the current sprite has collided, but also the accuracy of the collision and the relative position of the hit sprite.

6. Each sprite has four user-defined images. Each time the sprite is drawn, a different one of the four images is plotted. This is quite automatic and gives the appearance of smooth animation when moving along the screen.

7. Each sprite (set of four images) may have any number of clones (exact copies), as long as the total number of sprites does not exceed forty-eight. Each may be used and moved independently around the screen. The clone feature means that less memory is required to store the sprite data.

8. A swap routine allows sprite images to be exchanged.

9. An INKEY facility will detect whether any of the four definable keys have been pressed. It will return one of nine possible directions depending on the key pressed.

10. As well as being able to display sprites you can also delete and display copies. By deleting a sprite the background is restored, but displaying a copy of a sprite its image to be left on the screen.

    This is useful for creating backgrounds and trails.

11. When moving sprites you can either use one of nine pre-programmed directions or use the user-programmable directions. This routine automatically updates the current position of the sprites making the program more efficient.

3. GETTING STARTED

The tape supplied with this package contains a number of programs. These are as follows:

1. An overview with an introductory display of sprites moving about the screen.

2. A sprite definer program.

3. A sprite manipulator program.

4. A machine code routine for use with the manipulator program.

5. A machine code routine with some default sprites already defined.

6. Six demonstration programs.

These features will be explained in the course of the manual, but before starting you may wish to load the introductory program in order to give you an idea of what a sprite is.

To do this type `CHAIN"INTRO"` followed by RETURN.

At this point you may also like to look at the demonstration programs which form a part of the package. In this case please refer to section 9 of the manual.

4. USING SPRITES

4.1 OVERVIEW

When using this sprite package to create a computer game the first stage is to define a number of sprites using the sprite definer program (DEFINE). Once defined, each sprite is individually stored away on cassette.

The next step is to incorporate selected sprites into the machine code sprite manipulation program (MANIP).

You may then proceed to write your program in BASIC. This will access the sprite handling routines, which will need to be co-resident in the machine whenever the BASIC program is run.

In practice the machine code sprite routines supplied, already contain eight pre-defined sprites. We suggest that in the first instance you experiment with these before attempting to define your own sprites. The sprites supplied are as follows:

SPRITE NUMBER	DESCRIPTION
1	Pacman ghost
2	Cherry I
3	Cherry II
4	Laser base
5	Mean monster
6	Nice monster
7	Man I
8	Man II

It is very easy to display and move sprites around the screen, and the next section deals with the precise way in which this is achieved.

In preparation for experiments with the various sprite calls you should first load the machine code routines. This program is repeated a number of times on the tape. Please refer to the APPENDIX A1 to find out where they come on the tape. Once found you can load it by typing in the following:

```
HIMEM=&4600
*RUN M/CODE
```

4.2 MEMORY USAGE

The first thing that the program must do after selecting Mode 5 is to reset HIMEM to the value given in the manipulator program (or to &4600 if you are using the default machine code).

Early in your program a line similar to the following should appear:

```
100 MODE 5:HIMEM=&4600
```

HIMEM should be reset in this way after every mode change in your program. Failure to do this could cause the corruption of the machine code.

4.3 INITIALISATION

One other essential is to initialise the sprite handler. This is achieved by a call to V% and should be present at the start of any program involving sprites. Thus all sprite programs should contain the following two lines:

```
100 MODE 5:HIMEM=&4600
110 CALL V%
```

4.4 USE OF VARIABLES

The sprite routines use the resident integer variables (i.e. A% to Z%) to pass values to and from BASIC. Your BASIC program should not use these for purposes other than those outlined below; and they should not be directly used as a loop parameter in FOR NEXT loops. Of course any other integer variables (eg. AA%, a% and so on) may be freely used.

4.5 REFERENCING A SPRITE

Each of the forty-eight individual sprites use integer variables reserved for their X and Y screen co-ordinates. These are as follows:

Sprite Number						Co-ordinate Variables	
						X	Y
1	9	17	25	33	41	A%	B%
2	10	18	26	34	42	C%	D%
3	11	19	27	35	43	E%	F%
4	12	20	28	36	44	G%	H%
5	13	21	29	37	45	I%	J%
6	14	22	30	38	46	K%	L%
7	15	23	31	39	47	M%	N%
8	16	24	32	40	48	O%	P%

The X and Y co-ordinates correspond to those of the normal graphic scale, if the value lies outside of the screen boundary the sprite will wrap-around.

4.6 DISPLAYING A SPRITE

This is very easy to do. Y% is set to 0 and W% is used to tell the computer which sprite you wish to display. The routine is called using S% as follows:

```
150 W%=2
160 C%=300:D%=800
170 Y%=0:CALL S%
```

This will display sprite 2 at (300,800). If the sprite was already on the screen at some other location, displaying it in a new location will automatically delete the previous image. Hence you do not need to worry about deleting old images, as you cause movement it is all done for you.

If you wish to try this, `*RUN M/CODE` as described in section 4.1, then run the following program:

```
  0 REM PROG 1
 10 REM USES S% TO POSITION
 20 REM SPRITE NO 2
100 MODE 5:HIMEM=&4600
110 CALL V%
120 W%=2:Y%=0
130 C%=300:D%=800
140 CALL S%
```

4.7 DELETING A SPRITE

This is very similar to displaying a sprite. But instead of giving Y% a value of 0, set it to 2. If you wish to try this add the following lines to the previous program.

```
150 PRINT"PRESS ANY KEY TO DELETE SPRITE"
160 G$=GET$
170 REM DELETE SPRITE
180 Y%=2:CALL S%
```

4.8 DISPLAYING A COPY SPRITE

Again this uses the same cell, but with Y% set to 1. To try this add the following lines to program 'PROG 1' (See section 4.6).

```
190 REM PRESS ANY KEY
200 G$=GET$
210 REM DISPLAY A ONE-OFF SPRITE
```

```
220 Y%=1:CALL S%
230 C%=500:D%=200
240 Y%=1:CALL S%
```

If the sprite was already on the screen at some other location, displaying a copy at a new location will have no effect on the previous image.

4.9 MOVING A SPRITE

There are two ways of doing this:

1. Updating the Integer Variables

Plot the sprite on the screen, update the integer variables controlling its location, and simply plot it on the screen elsewhere - as described in section 4.6 above.

The program below uses this principle in conjunction with a FOR NEXT loop to move a sprite across the screen.

If you have pressed BREAK (thus resetting the machine) you will need to `*RUN M/CODE` before running the program, as described in 4.1.

```
  0 REM PROG 2
 10 REM USES S% TO MOVE
 20 REM SPRITE NO 2
100 MODE 5:HIMEM=&4600
110 CALL V%
120 W%=2:Y%=0
130 C%=120:D%=200
140 FOR loop = 1 TO 50
150 REM DELAY
160 A$=INKEY$(1)
170 REM CHANGE POSITION VARIABLES
180 C%=C%+8:D%=D%+4
190 REM DISPLAY SPRITE
200 CALL S%
210 NEXT
```

2. Special Routines

The second way to move a sprite is using the Special Routines.

Use the special programmable directions, and CALL T%. This is achieved as follows:

```
W%        = Sprite Number
```

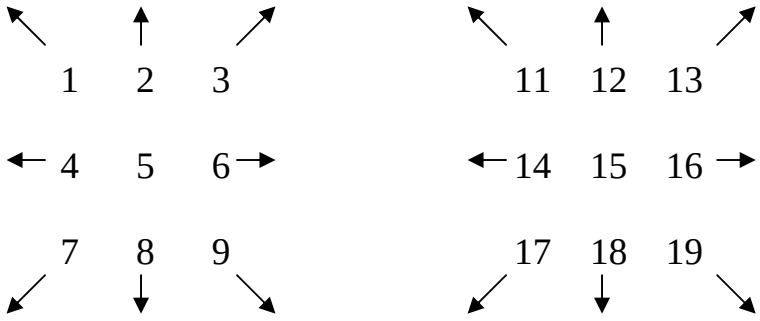
```
Z%          = Direction Number
CALL T%
```

e.g. Change the lines in PROG 2 to:

```
120 W%=2:Z%=3
130 C%=120:D%=200
140 REM DISPLAY SPRITE
150 Y%=0:CALL S%
160 FOR loop = 1 TO 50
170 REM DELAY
180 A$=INKEY$(1)
190 REM MOVES SPRITE
200 CALL T%
210 NEXT
```

This would move sprite 2 in direction 3. This call assumes that the sprite has already been displayed on the screen. If this is not the case a statement to display the sprite should be made before the call:
e.g. `W%=4:G%=500:H%=100:Y%=0:CALL S%`

The directions for the call to T% are initially defined as follows, but directions 11 to 19 may be re-programmed using the sprite manipulator program (See section 7):



Note that moving the sprites using this method will automatically update the relevant position variables. Thus in the above example with sprite 2, C% and D%, would have been automatically updated to reflect the sprite's new position.

As an example the following program uses this call to move a sprite randomly around the screen.

```
 0 REM PROG 3
10 REM USES T% TO MOVE
20 REM SPRITE NUMBER 2
```

```
 100 MODE 5:HIMEM=&4600
 110 CALL V%
 120 Y%=0:C%=30:D%=40:W%=2:CALL S%
 130 REPEAT
 140 REM DELAY
 150 A$=INKEY$(1)
 160 PROCRAND
 170 UNTIL FALSE

1000 DEF PROCRAND
1010 IF RND(20)=1 THEN Z%=RND(9)
1020 CALL T%
1030 ENDPROC
```

4.10 DELAY ROUTINES

Because of the high speed of movement provided by the sprites, you may need to slow them down a little to prevent them from shooting across the screen too quickly. This is fairly easy to do in BASIC with simple REPEAT loops.

5. ADVANCED FEATURES

It is advisable that the features covered in this section should only be used once familiarity has been gained with those outlined in the previous sections.

5.1 CLONES

Each sprite may have any number of clones (i.e. exact copies) as long as the total number of sprites and clones does not exceed forty-eight. The clones can be used and moved in exactly the same way as their parent, and since they use the same data require no memory of their own.

Each clone is allocated a number (See section 7.1: Load Sprites), identifying it in the same way as a sprite. In practice sprites and their clones are indistinguishable.

Sprite And Clone Numbers						Co-ordinate Variables	
						X	Y
1	9	17	25	33	41	A%	B%
2	10	18	26	34	42	C%	D%
3	11	19	27	35	43	E%	F%
4	12	20	28	36	44	G%	H%
5	13	21	29	37	45	I%	J%
6	14	22	30	38	46	K%	L%
7	15	23	31	39	47	M%	N%
8	16	24	32	40	48	O%	P%

As you will observe, the variables used to define the position of the sprites and clones may be the same. For example A% and B% are used to indicate the position of sprites 1, 9, 17, 25, 33 and 41. Hence to move all these sprites will require care on your behalf to ensure that you do not call, say, sprite 1 and then sprite 9, without updating the values of A% and B% to the new position of sprite 9.

5.2 ANIMATION

If the sprite that you are moving is, say, a man walking along, simply redrawing him in a different position will give the appearance of him gliding - not actually walking. The Beebugsoft Sprite pack takes care of this automatically.

When using the sprite definer (as explained later in this manual) it is possible to create four different versions of the same sprite. (Do not confuse this with clones). The different images of the individual sprite will be displayed as it is placed at different positions along the x-co-ordinate, giving the impression of movement.

5.3 CHECKING FOR A COLLISION

In most games, it is essential to know when one object on the screen hits another, for example, a bullet hitting an alien or, a pacman hitting a monster.

This routine will check for a collision between a set sprite and any other, and this is achieved as follows:

```
W%      = sprite number to be checked
Y%      = range of sprites to be checked
CALL Q%
```

After this call the results are stored in the following variables:

```
X%  = number of the sprite that was hit
Y%  = accuracy of the collision (0 = no collision)
Z%  = relative direction of the hit sprite
```

The reason for using a range is to save time when checking. For instance if you are only using the first six sprites then set the range to 6. To try this, type in the following program. If the machine code has not been loaded refer to section 4.1 and load it with `*RUN M/CODE`

```
  0 REM PROG 4
 10 REM MOES TWO SPRITES RANDOMLY
 20 REM AND CHECKS FOR CRASHES
100 MODE 5:HIMEM=&4600
110 CALL V%
120 Y%=0:VDU 23,1,0;0;0;0;
130 A%=100:B%=200:W%=1:CALL S%
140 C%=500:D%=800:W%=2:CALL S%
150 REPEAT
160   time%=TIME+RND(600)
170   Z1%=RND(9)
180   X2%=RND(9)
190   REPEAT
200     PROCmove1
```

```
 210     PROCmove2
 220     PROCcrash
 230   UNTIL time%<TIME
 240 UNTIL FALSE
 250 END

1000 DEF PROCmove1
1010 W%=1:Z%=Z1%
1020 CALL T%
1030 ENDPROC
1040 DEF PROCmove2
1050 W%=2:Z%=Z2%
1060 CALL T%
1070 ENDPROC
1080 DEF PROCcrash
1090 W%=1:Y%=2
1100 CALL Q%
1110 IF Y%<>0 THEN SOUND &10,-15,6,4:time%=0
1120 ENDPROC
```

5.4 SWAPPING SPRITE IMAGES

You may reach a point in your game when you need to change the appearance of a sprite. As previously mentioned there are 48 available sprites which may be defined as required. Each of these sprite images may be swapped around at will. This feature can be used to give spectacular explosions or to allow you to make the sprite face different directions. All this can be achieved by a simple call to U%

W% = sprite number
X% = sprite number of image to be copied
CALL U%

5.5 DISPLAYING SPRITE VARIABLES

As sprites have to share the same variables it is necessary to include a routine to allow you to find out the exact position of any sprite at any time. This is achieved as follows:

W% = sprite number
X% = 0
CALL U%

The X and Y co-ordinates are returned in the respective variables for that sprite.

5.6 DEFAULT VALUES

When writing a program the first thing to do after selecting 'MODE 5' and resetting 'HIMEM' is tell the computer that all previously positioned sprites are now not present on the screen. To do this use:

```
CALL V%
```

5.7 DIRECTING A DIRECTION KEY

This INKEY routine is very useful in games where the player has direct control over a sprite. By setting X% to a value in the range of (0-3) and calling R%, you can select any one of four options. These options are as follows:

X%=0 will return any direction in Z% depending on the key pressed. It will continue to return this value until another direction key is pressed.

X%=1 will again return any direction in Z% depending on the key pressed. But this time it will return a value of 5 when the key is released.

X%=2 will not return any diagonal directions in Z%. This value will continue to be returned until another direction key is pressed.

X%=3 will not return any diagonal directions in Z%. But it will return a value of 5 when the key is released.

The default direction keys are as follows:

Z - Left, X - Right, : - Up, / - Down

This routine can easily be used with the move routine.

```
  0 REM PROG 5
 10 REM USES INKEY ROUTINE TO
 20 REM MOVE A SPRITE
100 MODE 5:HIMEM=&4600
110 CALL V%
120 VDU 23,1,0;0;0;0;
130 A%=100:B%=200
140 Y%=0:CALL S%
150 X%=0:W%=1
160 REPEAT
170   CALL R%
180   CALL T%
190 UNTIL FALSE
```

6. DEFINING SPRITES

6.1 USING THE SPRITE DEFINER

Each sprite is defined separately and saved as a separate file. It is therefore possible to build up a library of sprites, which can be selected independently for use in programs. The selection is done using the Sprite Manipulator program (See section 7). To load the definer program, type `CHAIN"DEFINE".`

Each sprite has four separate images which are defined independently of each other. These images are defined in the large squares which form the centre of the screen. Above these squares are four smaller squares which represent the final shape and size of the finished sprite.

Along the top of the screen are the twelve options available to you. These are selected by holding down the SHIFT key down and pressing the 'cursor control' keys, this will cause the marker to move. When the marker is in the correct position press SHIFT and RETURN keys together to select that option. A brief description of those options follow:

ANIMATION
Will cause the computer to display the images one after another in a set sequence. This mimics the final movement in your program. By pressing the DELETE key before selecting this option the user can switch 'single step' on or off.

NEW COLOUR
Will run through a set number of colour combinations. These combinations have been selected to ensure that the screen is always readable. Each time the NEW COLOUR option is selected the four basic colours will be changed to a new set. Finally the computer will complete the sequence and revert back to the original colours.

WIPE GRID
Will clear the current image grid in the colour that the editing cursor (the small dot) is currently in.

QUIT
Will leave the Sprite Definer program.

COPY
Will ask you to key in the number of the image that you wish to copy. It will then proceed to copy it into the image grid that your editing cursor is currently in.

MIRROR
Will cause the image in the grid to reflect itself.

INVERT
Will turn the image upside down.

ROTATE
Will ask you to press one of the 'cursor control' keys and then it will move the whole image one place in that direction.

LOAD
Will produce on the screen a rectangular box, in which you may enter the seven letter filename. It will then load that file and display the sprite in the image grids.

SAVE
Will produce the same box, but will save the sprite onto the tape.

PRINT CAT
Will display the sprite filenames to aid you in positioning the tape. To leave this option, press ESCAPE.

DISPLAY CAT
Will display the sprite images in the smaller squares giving you a visual catalogue of the tape. Again press ESCAPE to leave this option, but you will have to wait while the original images are being restored.

6.2 DEFINING A SPRITE

First position the editing cursor and then press the number corresponding to the colour required. (Remember that you can physically change the actual colour using VDU 19 in your program - refer to User Guide). Repeat this process until the correct shape is obtained.

Next move the cursor to another grid. This is done by holding down CTRL and pressing either the left or right 'cursor control' key. Now copy the first image using the COPY option. You can then use the editing cursor (or any of the other options) to make any alterations.

Once all the images have been defined, select the ANIMATION option and see if the desired effect has been obtained. If not make any alterations using the editing cursor. (Remember that the basic colours can be changed at any time using the New Colour option).

When you are satisfied use the SAVE option to save the image onto your Sprite Library tape. This is now ready to be loaded into the machine code using the Sprite Manipulator program.

7. SPRITE MANIPULATOR PROGRAM

7.1 USING THE MANIPULATOR PROGRAM

Included in the package is a Sprite Manipulator program which is designed to take the hard work out of customising your own machine code routines. The options available are:

LOAD SPRITES

This is the only option that must be selected before continuing with the next program. After selecting this option the screen is presented with 48 numbers in the top right-hand corner. The number written in green represent the undefined sprites. At the bottom of the screen you are asked to enter the filename of the sprite followed by its number.

Once the number has been entered the corresponding green number turns yellow meaning that it has been defined. You are then asked if clones of that sprite are required. (A clone is an identical copy of that sprite that uses that same data and so requires no extra memory.)

If you enter 'Y' to that question you are then asked to enter the clone sprite numbers followed by RETURN, this is terminated by just pressing RETURN on its own. Finally you are asked if you want another sprite. If the answer is 'N' then any undefined sprites will be defined as clones.

SCREEN BOUNDARIES

A screen boundary is similar to a graphics window, except that any sprite moving out of the boundary wraps-around to the other side. The instructions for this are self-explanatory.

DIRECTION KEYS

This option allows you to enter your own keys for use with the INKEY routine.

PROGRAMMABLE DIRECTIONS

Will ask you to enter the X and Y steps for each direction. These steps can range from 0 to 120 and again they correspond to the normal graphics scale.

COLLISION RANGE

This option will allow for different widths or heights of sprites. For instance if all the sprites have a width of 8, enter this value. But if all the

widths are different, enter the largest value. The maximum values are, a width of 9 and a height of 16.

NEXT PROGRAM
When selected the second part of the Manipulator Program will be loaded.

This in turn loads the default machine code. It will ask you to position your tape containing the sprite data (as saved by the Sprite Definer) telling you which filename to find.

Once you have approximately positioned the tape press RETURN and the normal 'Searching' response will appear. You can now accurately position the tape. While the sprite data is being loaded the actual sprite will be displayed.

After all the sprites have been loaded, the computer will ask you to position the tape to save the customised machine code 'M/CODE' you will use in your programs. As mentioned previously it is advisable to save the machine code at the start of a *separate* tape.

Finally, write down the new value of HIMEM. Its value will need to be changed in your program so that it can accommodate the customised machine code, `'M/CODE'.`

7.2 MAKING A BACKUP COPY OF THE MACHINE CODE

You will probably require backup copies of the machine code, this is done as follows:

```
MODE 5
*OPT 1,2
*LOAD M/CODE
```

(After the code has been loaded the screen will look something like this:)

```
Loading

M/CODE    11 1200 00004600 00005499
```

(Using this information you can now make your backup copy by typing)

```
*SAVE M/CODE 4600 +1200 5499
```

8. USING SPRITES IN ASSEMBLY LANGUAGE

Sprites can also be used in assembly language. To do this you must first know the addresses of the static integer variable. A full list of these follow:

Variable	Low address	High address
A% B% C% D%	&404 &408 &40C &410	&405 &409 &40D &411
E% F% G% H%	&414 &418 &41C &420	&415 &419 &41D &421
I% J% K% L%	&424 &428 &42C &430	&425 &429 &42D &431
M% N% O% P%	&434 &438 &43C &440	&435 &439 &43D &441
W% X% Y% Z%	&45C &460 &464 &468	&45D &461 &465 &469

To call the routines from assembly language use the following list:

Normal Call Variable	Assembly Language Command
CALL Q% CALL R% CALL S% CALL T% CALL U% CALL V%	JSR !&444 JSR !&448 JSR !&44C JSR !&450 JSR !&454 JSR !&458

When using sprites from within assembly language it is essential that you initialise the sprite routine. This is done with the following command:

```
JSR !&458
```

You must also remember to preserve the Accumulator and the X and Y registers. This is done by placing them on the stack.

`PHA`	pushes the accumulator
`TXA`	transfers the X register to the accumulator
`PHA`	
`TYA`	transfers the Y register to the accumulator
`PHA`	
`JSR !address`	runs sprite routine
`PLA`	pulls the accumulator off of the stack
`TAY`	transfers the accumulator to the Y `register`
`PLA`	
`TAX`	transfers the accumulator to the X register
`PLA`	

The following example will display sprite 3 at the position 500,800.

```
JSR !&458
PHA
TXA
PHA
TYA
PHA
LDA #&500 DIV 256
STA &414
LDA #&500 MOD 256
STA &415
LDA #&800 DIV 256
STA &418
LDA #&800 MOD 256
STA &419
LDA #&00
STA &464
STA &465
LDA #&03
```

```
STA &45C
JSR !&44C
PLA
TAY
PLA
TAX
PLA
```

The sprite routines use zero page memory from &70 to &8F. This means that your program must not use any of these locations.

9. DEMONSTRATION PROGRAMS

Included in the sprites package is a series of six demonstration programs. Each program is listable, and the user is strongly recommended to experiment with these programs, trying to alter speeds, directions of movement, sprites plotted and so on. These programs are an ideal way to increase familiarity with sprites quickly. The user will then be better equipped to design programs of his own.

Users should note that the final SPACE bar pressing in each demonstration causes the program to be listed on the screen. To repeat the demonstration, just type `RUN`. When examining the program, please note that the function of the last five lines is to perform the auto-listing.

APPENDIX
A.1 PROGRAMS SUPPLIED ON THE CASSETTE

The programs are located on the tape in a convenient order. Where two programs are required in sequence, they are placed in order. The files are as follows:

Introduction	`INTRO`
	`(M/CODE)`
	`(PART_1)`

Sprite definer	`DEFINE`
	`(DEF_1)`
	`(DEF_2)`

Manipulator Program	`MANIP`
	`(MAN_1)`
	`(MAN_2)`
	`(CODE)`

Demonstrations	`M/CODE`
	`DEMO_1`
	
	`M/CODE`
	`DEMO_2`
	
	`M/CODE`
	`DEMO_3`
	
	`M/CODE`
	`DEMO_4`
	
	`M/CODE`
	`DEMO_5`
	
	`M/CODE`
	`DEMO_6`

If the program name is enclosed in brackets this means that it is automatically called by a previous program.

The introduction should be loaded by: `CHAIN "INTRO"`

The Sprite Definer by: `CHAIN "DEFINE"`

The Manipulator Program by: `CHAIN "MANIP"`

The Demonstration Programs by:

```
HIMEM=&4600
*RUN M/CODE
 CHAIN"DEMO_1"
```

etc.

A.2 SUMMARY OF DEFINING YOUR OWN SPRITES

i) CHAIN program 'DEFINE'
ii) Define four images of your sprite
iii) SAVE your sprite on tape
iv) Repeat (ii)-(iii) as necessary
v) CHAIN program 'MANIP'
vi) Using OPTION 1 (LOAD SPRITES), specify those sprites you have saved on tape that you want to use in your program
vii) Use OPTIONS 2-5 if required
viii) Select OPTION 6 (NEXT PROGRAM)
   a) Wait while program 'MAN_2' loads
   b) Wait while program 'CODE' loads
ix) LOAD the sprite(s) you have requested when the prompts appear
x) SAVE the customised machine code 'M/CODE' that you will use in your programs
xi) Write down the new value of HIMEM
xii) Write your own programs which should include:

```
10 *RUN M/CODE
20 MODE5:HIMEM=&<your new value>
30 CALL V%
```

If you are developing a program, line 10 can be deleted, but `'M/CODE'` must be `*RUN` at least once.

A.3 SUMMARY OF INTEGER VARIABLE USE

A% - P% Sprite (X,Y) co-ordinates

CALL Q% Test for collision
W% = sprite number
Y% = range of sprites

Results
X% = sprite hit
Y% = collision accuracy
Z% = relative direction of sprite hit

CALL R% Return direction key in Z%
X% = 0 Return same value until new key pressed
= 1 Returns a value of 5 when no key pressed
= 2 As X% = 0, but returns non-diagonal directions only
= 3 As X% = 1, but returns non-diagonal directions only

CALL S% Draw or delete sprite
A% - P% = (X,Y) co-ordinates
W% = sprite number
Y% = 0 (draw sprite, deleting old one)
= 1 (draw sprite)
= 2 (delete sprite)

CALL T% Move sprite
W% = sprite number
Z% = direction

CALL U% X = 0 Find co-ordinates of sprite W%
Results in A% - P%
X > 0 Swap sprite W% with write X%

CALL V% Sprite routine initialisation

W% (1-48) Sprite number
X% Parameter used with CALL Q%, CALL R%, CALL U%
Y% Parameter used with CALL Q%, CALL S%
Z% Parameter used with CALL Q%, CALL R%, CALL T%