

LASER BASIC FOR THE COMMODORE 64



by

OASIS SOFTWARE

LASER BASIC FOR THE COMMODORE 64

COPYRIGHT NOTICE

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

This MANUAL

Piracy has reached epidemic proportions and it is with regret that we are forced to reproduce this manual in a form which cannot be photocopied.

Our apologies for any inconvenience this may cause to our genuine customers. A reward will be paid for information leading to the successful prosecution of parties infringing this Copyright Notice.

NOTE

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

ENQUIRIES

If you have any queries on the use of Laser BASIC, please send them to us in a letter, ensuring you enclose the Enquiry Card printed on the last page of this manual. A new card will be returned to you with your reply. Please note that enquiries not accompanied by the card will not be answered

Copyright c by Oasis Software

https://69.60.118.202/commodore/applications/Laser_BASIC_Instruction_Manual.pdf

CONVERSION FROM SCAN TO PDF NOTE

The original scanned document had an orange background to prevent 1980's photocopiers from being able to reproduce the manual. As a result, printed copies of the manual are painfully red-ink wasteful.

Changes:

The table of contents has been renumbered to match the new document. Effort was made to keep the general format of the original document as much as possible. Spelling errors were corrected when found. Examples of code were keyed in, run, debugged and saved to disk image using [vice 3.3](#). Some examples were enhanced to follow rules of structured programming. Also added to this document where screenshots of the examples.

Examples were also tested on the [C64 Mini](#) except for the light pen functionality which will not work with the HDMI output.

Document Revisions

Version	Revised by	Revision Description	Date
1.0	David Hunter	Original Publication	Cira 1985
2.0		Converted to MS Word format and reformatted.	December 2019

LASER BASIC FOR THE COMMODORE 64

Table of Contents

Document Revisions.....	2
INTRODUCTION	10
LOADING LASER BASIC	12
Tape Users:	12
Disk Users:	12
STRUCTURED PROGRAMMING	14
IF-THEN-ELSE.....	14
CIF-CElse-CEND	14
LABELS.....	15
REPEAT-UNTIL.....	16
WHILE-WEND.....	16
EXIT	17
TOOLKIT COMMANDS AND MISCELLANEOUS KEYWORDS.....	21
HEXADECIMAL NUMBERS	21
DOUBLE-BYTE PEEK AND POKE	21
TURNING OFF THE STOP KEY.....	22
LOADING AND SAVING PROGRAMS WITH DISK.....	22
POPPING A RETURN ADDRESS	22
PI	22
PRINTING A DISK DIRECTORY (DIR)	22
REMOVING A NEWED PROGRAM (OLD).....	22
AUTOMATIC LINE NUMBERING (AUTO)	22
RENUMBERING A PROGRAM (RENUM).....	23
GRAPHICS COMMANDS	23
GRAPHICS MODES	23
BITS AND BYTES.....	24
TWO-COLOUR TEXT MODE (LORES MODE)	24
EXTENDED BACKGROUND COLOUR MODE	24
MULTI-COLOUR TEXT MODE	25
TWO-COLOUR HIRES (BIT-MAPPED) MODE	25
MULTI-COLOUR HIRES MODE	25
SPRITES	26

LASER BASIC FOR THE COMMODORE 64

WHAT IS A SPRITE?	26
HIRES SPRITES.....	26
CHARACTER SPRITES	26
SPRITE NUMBERS	27
SPRITE UTILITIES.....	27
DELETING ALL SPRITES (RESET)	27
CREATING A HIRES SPRITE (SPRITE).....	27
CREATING CHARACTER SPRITES (CSprite).....	27
DELETING A SPRITE (WIPE).....	28
FINDING A SPRITE'S ADDRESS.....	28
SPRITE VARIABLES	29
WHAT ARE SPRITE VARIABLES?	29
WHY ARE SPRITE VARIABLES USED?.....	30
USING COMMANDS WITHOUT PARAMETERS.....	30
SAVING and LOADING Sprites.....	31
SAVING SPRITES.....	31
LOADING SPRITES (RECALL, DRECALL, MERGE and DMERGE)	31
RENUMBERING SPRITES - (RESEQ).....	32
RESERVING MORE SPACE FOR SPRITES (RESERVE)	32
SETTING UP THE DISPLAY.....	32
DISPLAY INITIALISATION (INIT)	32
DISPLAYING HIRES GRAPHICS (HIRES)	32
GOING BACK INTO TEXT MODE (LORES).....	33
DISPLAYING EXTENDED BACKGROUND COLOUR MODE (EBACK)	33
COLOUR CONSTANTS	33
HOW TO CHANGE THE BORDER COLOUR (TBORDER , HBORDER).....	34
HOW TO CHANGE THE BACKGROUND COLOUR (TPAPER, HPAPER)	34
HOW TO CHANGE THE EXTENDED BACKGROUND COLOURS	35
SETTING UP THE PRINTING COLOUR (INK)	36
GOING INTO MULTI-COLOUR MODE (MULTI , MONO).....	36
MIXING DISPLAY MODES (WINDOW)	37
SCREEN MODES SUMMARY	37
SETTING UP SOFTWARE MODES.....	38

LASER BASIC FOR THE COMMODORE 64

SCREEN MODES SUMMARY WITH SOFTWARE MODES	39
CLEARING SPRITES (SETA, WCIR, SCIR)	39
HIRES GRAPHICS COMMANDS	43
HOW TO SET UP THE PLOTTING COLOUR (MODE)	43
SETTING INDIVIDUAL PIXELS	44
DRAWING RECTANGLES (BOX)	47
DRAWING LINES (DRAW)	48
DRAWING CIRCLES AND POLYGONS	52
FILLING IN BLANK AREAS (FILL)	56
EXAMINING INDIVIDUAL PIXELS	61
SPRITE 'BLOCK MOVE' COMMANDS	62
DISPLAYING A SPRITE ON THE SCREEN	62
CONTROLLING ATTRIBUTES	64
LOGICAL OPERATIONS	64
OTHER BLOCK-MOVE COMMANDS	66
SUMMARY	68
SOME SIMPLE PROGRAMMING EXAMPLES	69
TWO-WAY BLOCK MOVE COMMANDS	74
COLLISION DETECTION	76
ATTRIBUTE BLOCK-MOVE COMMANDS	78
SPRITE TRANSFORMATIONS AND OTHER MISCELLANEOUS COMMANDS	80
HORIZONTAL REFLECTION	80
VERTICAL REFLECTION	81
HORIZONTAL EXPANSION	82
VERTICAL EXPANSION (EXY)	82
ROTATING A SPRITE WINDOW (SPIN)	83
INVERTING A WINDOW (INV)	84
SCANNING AN AREA OF THE SCREEN	86
INSPECTING ATTRIBUTES	87
SCROLLING COMMANDS	87
SCROLLING PIXEL DATA HORIZONTALLY	87
REPEATING A GRAPHICS COMMAND	88
AVOIDING FLICKER	89

LASER BASIC FOR THE COMMODORE 64

SCROLLING PIXEL DATA VERTICALLY	91
SCROLLING ATTRIBUTES.....	92
TWO PROGRAMMING EXAMPLES	93
READING THE KEYBOARD, JOYSTICK AND LIGHTPEN.....	96
DETECTING KEY DEPRESSIONS (KB)	96
READING THE JOYSTICK.....	97
USING A LIGHTPEN	99
CHARACTER SET MANIPULATION	101
LOWER CASE AND UPPER CASE	101
ASCII AND DISPLAY CODES.....	101
PUTTING A CHARACTER INTO A SPRITE.....	104
PUTTING A STRING INTO A SPRITE (TEXT).....	107
REDEFINING A CHARACTER.....	108
HARDWARE SPRITES.....	111
CREATING A HARDWARE SPRITE DEFINITION	111
ASSOCIATING A HARDWARE SPRITE WITH ITS DEFINITION	112
SETTING UP A HARDWARE SPRITE'S COLOUR.....	112
SWITCHING A HARDWARE SPRITE ON AND OFF.....	112
POSITIONING A HARDWARE SPRITE ON THE SCREEN.....	112
EXPANDED HARDWARE SPRITES	116
HARDWARE SPRITE DISPLAY PRIORITIES.....	118
Sprite to Screen Display Priorities	119
MOVING HARDWARE SPRITES AUTOMATICALLY	121
TRACKING SPRITES	124
FOUR-COLOUR MODE HARDWARE SPRITES.....	129
COLLISION DETECTION WITH HARDWARE SPRITES	130
MISCELLANEOUS COMMANDS	133
DISPLAY BLANKING	133
DISABLING KEYBOARD SCANNING	133
CHARACTER SPRITES AND SMOOTH SCROLLING	133
USING ATTRIBUTE COMMANDS WITH CHARACTER SPRITES	134
PUTTING AND GETTING CHARACTER SPRITES.....	136
SMOOTH SCROLLING.....	139

LASER BASIC FOR THE COMMODORE 64

SOUND COMMANDS	143
INITIALISING THE SID	143
SETTING THE MASTER VOLUME	143
SETTING THE FREQUENCY	143
SETTING THE ENVELOPE	143
SETTING THE WAVEFORM	144
SAWTOOTH WAVEFORM (SAW)	144
TRIANGLE WAVEFORM (TRI)	145
PULSE WAVEFORM (PULSE)	145
WHITE NOISE (NOISE)	146
MAKING A SOUND (MUSIC)	146
PLAYING MUSIC UNDER INTERRUPT	148
Format for Storing Music inside Sprites	150
FILTERING	152
SETTING THE CUTOFF FREQUENCY	152
SELECTING THE FILTER	152
RESONANCE (RESONANCE)	154
ENABLING THE FILTER	154
RING MODULATION	155
SYNCHRONISATION	157
DYNAMIC EFFECTS	157
MULTI TASKING	159
ALLOCATING MEMORY	159
STARTING A CONCURRENT TASK	159
ERROR MESSAGES WITH MULTI-TASKING	161
COMMUNICATION BETWEEN TASKS	161
USING MULTI-TASKING WITH GRAPHICS	162
THE SPRITE GENERATOR PROGRAM	163
GLOSSARY	163
THE ARCADE SPRITE SET	163
THE DEMONSTRATION SPRITES	163
THE CHR\$ SQR	163
THE HI-RES SCREEN	163

LASER BASIC FOR THE COMMODORE 64

THE CHR\$ SQR CURSOR.....	163
HI-RES WINDOW.....	163
SPRITE LIBRARY.....	163
SOFTWARE SPRITES.....	164
HARDWARE SPRITES.....	164
INKS and PAPER.....	164
THE COMMODORE KEY.....	164
THE FUNCTION KEYS.....	164
MODE:1.....	164
MODE:2.....	166
MODE:3.....	167
MODE:4.....	169
MODE:5.....	173
DISK COMMANDS.....	173
A SAMPLE SESSION WITH THE SPRITE GENERATOR.....	174
FUNCTION KEY SUMMARY.....	180
MODE:1.....	180
MODE:2.....	180
APPENDICES.....	183
APPENDIX A : LASER BASIC KEYWORD SUMMARY.....	183
Display Table.....	197
Offset Table.....	197
APPENDIX B: ERROR MESSAGES.....	198
APPENDIX C : DIFFERENCES BETWEEN BASICS.....	199
APPENDIX D: GLOSSARY OF TERMS.....	201
APPENDIX E : SCREEN DISPLAY CODES.....	203
APPENDIX F : ASCII AND CHR\$ CODES.....	204
APPENDIX G: MEMORY MAP.....	205
APPENDIX H : SPRITE LIBRARIES.....	206
ARCADE SPRITE LIBRARY - "SPRITESA".....	206
DEMO SPRITE LIBRARY - "SPRITESB".....	207
CHARACTER SPRITES AND MUSIC SPRITES - "SPRITESC".....	208
APPENDIX I: MUSIC NOTE VALUES.....	208

LASER BASIC FOR THE COMMODORE 64

Appendix J: C64 and Vice information	209
US PC keyboard	209
Appendix K: Package Information	210
NOTES.....	211
TECHNICAL ENQUIRY CARD.....	212

LASER BASIC FOR THE COMMODORE 64

LASER BASIC FOR THE COMMODORE 64 by David Hunter

INTRODUCTION

Laser BASIC is an easy to use and versatile utility which lets you make full use of the Commodore 64's outstanding graphics capabilities without using machine code. Although it was originally designed with video games in mind, it can be used in any application which requires high quality graphics.

Laser BASIC consists of three compatible parts:

- a) The SPRITE GENERATOR program. You can use this to create the graphics used in your program. This is documented in Chapter 7 of this manual.
- b) The Laser BASIC interpreter. This is a multi-tasking extension to the Commodore's resident BASIC which adds 256 extra keywords. If you don't understand any of the following description, don't worry because everything is explained in this manual. The new commands can be split into four types:
 - 1) Structured programming commands
 - 2) Toolkit commands
 - 3) Graphics commands
 - 4) Sound commands

The structured programming commands add PASCAL-like control structures to BASIC:

- IF-THEN-ELSE
- REPEAT -UNTIL
- WHILE-WEND
- Procedures
- Multiple-line functions
- CASE-OF-CASEND
- Local variables

These commands make GOTO-less programming possible.

The toolkit commands make it easier for you to write and edit your program . The following are included:

- RENUM - renumber a BASIC program, including GOTOs and GOSUBs
- AUTO - auto line-numbers for use when you type in a program
- OLD - recover a NEWed program
- DIR - disk directory

Every graphics feature of the 64's powerful hardware is supported by the graphics commands. This means that you can:

- Plot points

LASER BASIC FOR THE COMMODORE 64

- Draw lines or polygons
- Fill in areas of the screen
- Use up to 254 software sprites
- Use the 64's own hardware sprites

The software sprite commands are the most important part of Laser BASIC. There are commands which let you do the following, in any of the 64's five screen modes:

- Display sprites on the screen
- Scroll sprites
- Rotate sprites
- Invert sprites
- Expand sprites
- Mirror sprites
- Create smooth -scrolling backgrounds
- Redefine the character set
- Change a software sprite into a hardware sprite

If you are using the 64's own hardware sprites, there is a facility to move them around the screen automatically under interrupt.

The sound commands let you use every facility of the 64's sound chip, the S.I.D or Sound Interface Device. You can even play tunes under interrupt while your program does something else.

The Laser BASIC Interpreter is documented in Chapters 2 to 6 of this manual.

- c) The Laser BASIC Compiler. This must be bought separately from the Laser BASIC Interpreter. Once you have finished your program, you can use the compiler to translate it into a fast machine code program which can be run Independently of the Laser BASIC software. You may market the compiled program without paying royalties, although we do ask that you include a credit on the packaging of the finished product.

There are several options available to you when using the compiler:

- Integer or floating-point arithmetic
- Error checking on or off
- Laser BASIC or ordinary Commodore Basic programs

This manual is also included with Machine Lightning, The Machine Lightning manual does not give any details of what the graphic s routines do since the commands are all documented here.

To make full use of this manual and Laser BASIC, it is absolutely essential that you can already program using the 64's resident (non-extended) BASIC. If you have not already done so, you should read the User' s Guide which was supplied with your 64. You can miss out chapters 6 and 7.

LASER BASIC FOR THE COMMODORE 64

LOADING LASER BASIC

Tape Users:

There are two tapes supplied with Laser BASIC; tape 1 is recorded in fast loading turbo format, and the tape 2 is in the normal slow format. If you have problems loading the turbo format on your tape recorder, you can use the normal format tape instead. The programs are arranged on the tapes as follows:

SIDE A:	Demo	"LBI"
	Demo Sprites	"DEMO"
	Compiled Demo	"SPRITESB"
	Sprite Generator	"CDEMO"
SIDE B:	Laser BASIC Interpreter	"SPTGEN "
	Arcade Sprites	"SPRITESA"
	Demo Sprites	"SPRITESB"
	Character Sprites	"SPRITESC"

So, to load in the interpreter, insert one of the tapes into the recorder at SideA, press SHIFT-RUN / STOP and start the recorder in the normal way.

Disk Users:

Put the disk into the drive and type `LOAD "LBI",8,1`.

After about ten seconds, the title screen will appear, and the interpreter takes just over a minute to load.

It is essential that before loading the interpreter from disk or tape, the computer is completely reset. The best way to do this is to switch it off and on again.

Once the interpreter has loaded, a menu giving four options is printed; you must press either 1, 2, 3 or 4 to select how much memory you want.

Normally, you should press "1" unless you are writing a very large program. If you want more memory, you must do without either the multitasking (see Chapter 6) or the turbo-tape facility, or both. If you select options 3 or 4, all the multi-tasking commands, as well as `RENUM`, `AUTO` and `OLD` (see Chapter 2) cannot be used.

If you select options 1 or 3, all programs and sprites are saved and loaded on tape at ten times the normal speed. Programs and sprites saved in the normal slow format can still be loaded. If you wish to save to tape at normal speed, you must use a secondary address of 1, i.e.:

```
SAVE "filename",1 ,1
```

or

```
STORE "filename",1 ,1
```

This fast tape format is compatible with the Laser BASIC Compiler, and any compiled BASIC programs.

Once you have made the selection, the title is printed, followed by "ready" and the flashing cursor, just as when the computer is first switched on. The screen background is light grey, with an emerald green border and blue characters. The interpreter is now waiting for you to type in a command.

LASER BASIC FOR THE COMMODORE 64

We suggest that you have the computer beside you with the interpreter loaded while you read this manual, so that you can tryout the various examples.

A demo program is also included with the package, and you might like to run it before proceeding so that you can see what can be done with Laser BASIC.

For tape, first load in the interpreter, then load in the demo program, which is recorded after it, using SHIFT-RUN / STOP in the normal way.

For disk, load in the interpreter as before, then type DLOAD " DEMO". Once the program has loaded, type RUN. There will be a delay of a few seconds while sprites are loaded in from disk.

The Laser BASIC interpreter is an updated version of an earlier program called BASIC Lightning. A list of the new commands which have been added can be found in Appendix C.

The compiled version of the demo program is also supplied. To load this from tape, first reset the computer, insert one tape at side A and type LOAD "CDEMO". To load it from disk, type LOAD "CDEMO",8. When it has loaded, type RUN.

LASER BASIC FOR THE COMMODORE 64

STRUCTURED PROGRAMMING

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

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

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

IF-THEN-ELSE

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

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

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

1 is printed if 'a' is true, irrespective of the value of 'b'.

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

3 is printed if 'a' is false and 'b' is false.

However, the following is ambiguous:

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

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

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

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

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

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

CIF-CELSE-CEND

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

Example:

```
100 LABEL STCHECK  
110 `  
120 CIF T <> 0  
130 OPEN 15,8,15 `SUBROUTINE TO CHECK ST  
140 INPUT# 15,W$,X$,Y$,Z$  
150 PRINT W$; ", "; X$; ", "; Y$; ", "; Z$
```

LASER BASIC FOR THE COMMODORE 64

```
160 ER= 1
170 CLOSE 15
180 CELSE
190 ER=0
200 CEND
210 RETURN
```

If you have typed in any of the examples so far, you will have noticed that all reserved words are printed in upper case when listing, and everything else is printed in lower case. This makes the program easier to read, particularly if many statements are put on one line without any spaces. If you want to list to a printer, you must type "UCASE" first to disable this feature. To restore it type "LCASE".

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

LABELS

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

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

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

```
z9    grid3%    i3v1  v$62
```

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

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

Example:

```
10 INPUT "SKILL LEVEL (1,2 OR 3)";I
20 ON I RESTORE EL,ML,HL
30 READA$
40 PRINT "LOADING"; A$; "..."
50 READB$
60 LOADB$,8,1
70 LABEL EL:DATA "EASY LEVEL" , "LEVEL 1"
80 LABEL ML:DATA "MEDIUM LEVEL", "LEVEL 2"
90 LABEL HL:DATA "HARD LEVEL", "LEVEL 3"
```

Labels are also used to define procedures and multiple-line functions; this is explained fully later. Also, labels can be used with the LIST command, e.g. "LIST chsde,".

LASER BASIC FOR THE COMMODORE 64

REPEAT-UNTIL

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

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

```
10 REPEAT
20 INPUT "4-CHARACTER CODE"; AS
30 UNTIL LEN (AS) = 4
```

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

WHILE-WEND

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

Try running this program:

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

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

```
WHILE-WEND
HILE-WEND
ILE-WEND
LE-WEND
E-WEND
-WEND
WEND
END
ND
D
.... END ....
```

However, if you type in a null string (two quotes and RETURN), you will see that the loop itself is not executed at all since `x$<>""` gives a FALSE value the first time around. Since TRUE and FALSE return values of -1 and 0 respectively, it is possible to set up an infinite loop using:

```
REPEAT . . . . UNTIL FALSE
```

or

LASER BASIC FOR THE COMMODORE 64

```
WHILE TRUE . . . . WEND
```

EXIT

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

```
10 REPEAT
20 INPUT "4-CHARACTER CODE"; AS
30 UNTIL LEN (AS) = 4
```

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

```
10 REPEAT
20 INPUT "4-CHARACTER CODE"; A$
25 IF LEN (A$) <> 4 PRINT "4 CHARACTERS PLEASE-TRY AGAIN"
30 UNTIL LEN(A$) = 4
```

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

```
10 REPEAT
20 INPUT "4-CHARACTER CODE"; A$
25 IF LEN (AS) = 4 EXIT
27 PRINT "4 CHARACTER PLEASE-TRY AGAIN"
30 UNTIL FALSE
```

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

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

CASE-OF-CASEND

The CASE statement is used to select between a number of alternative courses of action, depending on the value of a variable which must be numeric.

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

Example:

```
10 INPUT A
20 CASE A
30 OF 3 : PRINT "THREE FRENCH HENS,"
40 OF 2,3 : PRINT "TWO TURTLE DOVES,"
50 OF 1,2,3 : PRINT "AND A PARTRIDGE IN A PEAR TREE."
60 OF OR : STOP
70 CASEND
```

LASER BASIC FOR THE COMMODORE 64

PROCEDURES

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

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

```
10 LABEL TEST(I)
20 LOCAL J
30 FOR J=1 TO I
40 PRINT J;
50 NEXT J
60 PRINT
70 PROCEND
```

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

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

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

```
FOR J=1 TO 12 : PROCTEST(J) : NEXT
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10 11
1 2 3 4 5 6 7 8 9 10 11 12
```

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

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

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

```
100 LABEL CENTRE (AS,I)
110 PRINT@, ; " " '40 SPACES
```

LASER BASIC FOR THE COMMODORE 64

```
120 PRINT@20 - LEN(AS) / 2,I; AS
130 PROCEND
```

Note that this procedure has two parameters instead of one as in the previous example - the number of parameters a procedure can have is only limited by the maximum line length of 80 characters. In lines 110 and 120, PRINT@ is used: PRINT@X,Y; will position the cursor on screen column X and row Y before printing the data following it.

Type:

```
proccentre ("Laser Basic", 10)
```

and " Laser Basic" will be printed on the middle of the tenth line of the screen .

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

VAR PARAMETERS

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

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

```
100 LABEL INP (AS,VAR BS)
110 PRINT@0, 10; AS;
120 INPUT BS
130 PRINT@0, 10;"      " '40 spaces
140 PROCEND
```

Now clear the screen using SHIFT -CLR and type:

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

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

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

ARRAYS AS PARAMETERS

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

LASER BASIC FOR THE COMMODORE 64

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

```
DIM x(1 0,20)
```

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

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

```
10 INPUT N 'GET THE NUMBER OR RECORDS
20 DIM AS(N)
30 PROC ENTER(AS()) 'GET RECORDS
40 PROC SORT (AS()) 'SORT
50 PROC OUT (AS()) 'AND PRINT
60 END
70 '
80 LABEL ENTER (VAR X$( ))
90 LOCAL I
100 FOR I=1 TO SIZE (X$, 1) - 1
110 INPUT XS(I)
120 NEXT I
130 PROCEND
140 '
150 LABEL SORT (VAR Y$( ))
160 LOCAL I,J,K$
170 REPEAT
180 J=TRUE
190 FOR I=1 TO SIZE (YS,1) - 2
200 CIF YS (I) > YS(I+1)
210 K$ = Y$(I) 'SWAP TWO ELEMENTS
220 YS(I) = YS(I+1) 'IF OUT OF SEQUENCE
230 YS(I+1) = K$
240 J=FALSE
250 CEND
260 NEXT I
270 UNTIL J 'UNTIL ELEMENTS IN ORDER
280 PROCEND
290 '
300 LABEL OUT (VAR Z$( ))
310 LOCAL I
320 FOR I=1 TO SIZE (Z$,I) - 1
330 PRINT ZS(I),
340 NEXT I
350 PROCEND
```

MULTIPLE-LINE FUNCTIONS

LASER BASIC FOR THE COMMODORE 64

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

```
10 LABEL FACT(N)
20 LOCAL I, J
30 I=1
40 CIF N ,> 1
50 FOR J=2 TO N
60 I=I*J
70 NEXT J
80 CEND
90 =I
```

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

?CFNfact(4) will print '24'.

TOOLKIT COMMANDS AND MISCELLANEOUS KEYWORDS

HEXADECIMAL NUMBERS

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

```
? $9800/2
```

will give a result of 19456.

It is also possible to convert a numeric result into hexadecimal using the string function HEX\$, e.g

```
? HEX$ (19456)
```

will give \$4C00.

The string is always five characters long - the '\$' plus four digits. The number being converted must be in the range 0 to 65535.

DOUBLE-BYTE PEEK AND POKE

(DEEK, DOKE)

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

DEEK returns a value between 0 and 65533; for example

```
? DEEK (65532)
```

will give the same result as

```
? PEEK (65532) + 256 * PEEK (65532)
```

DOKE \$COOO, \$55FF is equivalent to POKE \$COOO, \$FF: POKE \$C001 \$55

LASER BASIC FOR THE COMMODORE 64

TURNING OFF THE STOP KEY

(DISABLE)

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

LOADING AND SAVING PROGRAMS WITH DISK

(DLOAD, DSAVE)

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

Example: **DLOAD "DEMO"**

POPPING A RETURN ADDRESS

(PULL)

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

PULL : RETURN

PI

PI is a constant which will return the value 3.14159265

PRINTING A DISK DIRECTORY (DIR)

If you have a disk drive, DIR can be used to print out the disk directory. As with LOAD or SAVE, the filename and device number can be specified;

DIR"\$0:0*",9

would print out all filenames on device number 9 which start with "0". For further details, refer to your disk drive manual.

REMOVING A NEWED PROGRAM (OLD)

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

AUTOMATIC LINE NUMBERING (AUTO)

This command is very useful when typing in programs, because it puts the line numbers onto the screen automatically. It should be followed by the 'step size' between the li ne numbers.

As an example, type in **AUTO 10** and then the following line:

10 for j = 1 to 100

The line number for line 20 will appear automatically. If you had typed **AUTO 11** instead, line 21 would have appeared.

LASER BASIC FOR THE COMMODORE 64

To exit from this mode, hit return when the line number appears without typing anything on the line. The auto-line-numbering feature is still enabled and will be re-activated the next time you type in a line of program. To disable the facility, type **AUTO 0** or just **AUTO** on its own.

RENUMBERING A PROGRAM (RENUM)

This command will renumber all the lines in a program, changing all GOTOs, GOSUBs and RESTOREs. If it is used without any parameters, the first line number and the step size are both 10. To specify a particular starting line number, but still using steps of 10, use:

```
RENUM < first line>
```

e.g. **RENUM 100**

To specify both the first line and the step size, use:

```
RENUM < first line>,<step size>
```

e.g. **RENUM 1,1**

Before any renumbering is actually carried out, the whole program is checked to see that all line numbers used in GOTOs, GOSUBs and RESTOREs actually exist. Any references to lines that are undefined are printed out like this:

A is the line in which the offending GOTO, GOSUB or RESTORE was found and B is the line number which is not defined in the program. If any errors of this type occur, no renumbering takes place.

As an example, type NEW followed by these lines:

```
1 GOSUB 25  
2 GOTO 19  
3 RESTORE 2
```

Now type RENUM. The following is printed:

```
1?25  
2?19
```

Note that the RESTORE in line 3 did not generate an error message because line 2 is defined in the program.

If you now type LIST, you will see that renumbering has not taken place because of the errors.

N.B. LIST, OLD, AUTO, and RENUM can only be used from immediate mode (i.e. they cannot be put in a program) and they cannot be put on the same line as other commands.

GRAPHICS COMMANDS

GRAPHICS MODES

The Commodore 64 has five different "display modes". In each display mode, the number of colours allowed, display resolution and speed of operation differs. This section describes each of these modes in detail. If you are already familiar with the 64's hardware, you can skip this part and go on to [sprites](#).

LASER BASIC FOR THE COMMODORE 64

BITS AND BYTES

The most fundamental unit of information is called a 'bit'. It can be represented by a binary digit which is either 0 or 1. The 64's memory has 65536 separate memory locations. Each of these holds a 'byte' of information. A byte contains eight bits, each of which may be either 0 or 1.

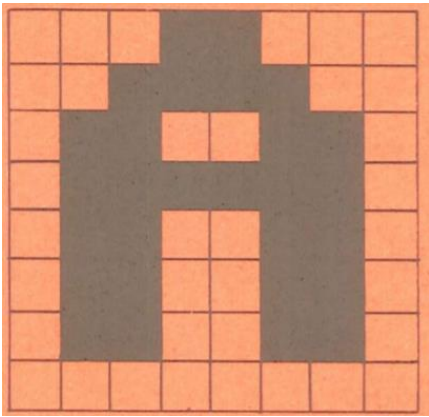
For additional information, refer to pages 76 to 78 of the Commodore 64 User's Guide.

In the following section on graphics modes, you will notice that characters, screen colours and so on tend to be numbered starting from zero rather than one.

TWO-COLOUR TEXT MODE (LORES MODE)

This is the screen mode that the computer is in when you switch it on. The screen is 40 characters wide and 25 characters high. Each character takes up one byte in screen memory, so 256 different characters are allowed, and the screen uses up $25 \times 40 = 1000$ bytes.

A character is formed by an 8x8 matrix of dots, like this:



The information governing what characters look like, called character definitions, is stored sequentially in a separate area of memory called character memory. Each character definition has $8 \times 8 = 64$ separate dots or 'pixels' which are either on or off. If each pixel is represented by one bit which is either 1 or 0, then 64 bits are needed to hold one-character definition. This is 8 bytes, since each byte holds 8 bits. There are 256 characters, so the character memory takes up $256 \times 8 = 2048$ bytes.

Also associated with each of the 1000 character positions on the screen is a one-byte attribute value. This determines what the colour of the corresponding character is going to be. Only 4 bits of each attribute memory location are used, giving 16 possible character colours. Note that although this is called two-colour mode, up to 16 colours may appear on the screen.

In this mode, the background colour is the same for the whole screen. The exact way that the bytes of screen, character and attribute memory are arranged need not concern you since this is all taken care of by Laser BASIC

EXTENDED BACKGROUND COLOUR MODE

This mode is similar to two-colour text mode. The difference is that it allows each character to have one of four background colours, rather than the background being the same for the whole screen. However, only the first 64-character definitions (0 to 63) can be displayed. The character to be displayed is

LASER BASIC FOR THE COMMODORE 64

determined by taking the remainder upon dividing the value in screen memory by 64, whilst the quotient (0, 1, 2 0 3) gives the background to be used. Each of the four backgrounds can be any one of 16 colours.

MULTI-COLOUR TEXT MODE

In this mode, it is possible to display four different colours in one-character square. If the attribute colour value for a character is less than 8, the character is displayed normally, as in two-colour text mode. However, if the colour value lies from 8 to 15, the following happens.

The number of pixels in the character is reduced from 64 to 32. The pixels are twice as wide as in two-colour mode, and the character is 4 pixels wide. Thus each pixel contains two bits which can be set up in four combinations: 00, 01, 10 or 11, i.e. 0, 1, 2 or 3. Colours 0, 1 and 2 are the same for the whole screen, while colour 3 is given by the attribute value for each character minus 8. So the letter 'A' shown earlier would look like this:

0	1	2	0
0	3	3	0
1	2	1	2
1	3	3	2
1	2	1	2
1	2	1	2
1	2	1	2
0	0	0	0

0=00

1=01

2=10

3=11

TWO-COLOUR HIRES (BIT-MAPPED) MODE

In the hires graphics modes, each pixel on the screen has a separate memory bit to control it, and is therefore accessed directly. highly detailed pictures can be drawn on the screen in this mode.

Unfortunately, hires graphics do have a couple of drawbacks:

- I. The pixels are the same size as in text mode. This gives us a resolution of 320 by 200 which is 64000 pixel bits, or 8000 bytes. So hires mode uses up 8 times as much memory as text mode.
- II. The larger memory usage of hires graphics means that moving large blocks of the screen around is eight times slower than in text mode.

As in text mode, each character-sized block on the screen has a corresponding attribute byte. In this case, all eight bits are used. The byte is split in two - one set of four bits is used to specify one of 16 foreground colours, while the other four bits specify the background colour for the character block.

MULTI-COLOUR HIRES MODE

In this mode, the horizontal resolution is reduced from 320 to 160 pixels. As with multi colour text mode, each pixel now uses two bits, giving four possible colours.

LASER BASIC FOR THE COMMODORE 64

Each character-sized block has two sets of attributes - primary attributes and secondary attributes. The primary attributes are split in a similar way to the attributes in two colour bit-map mode in order to specify colours 1 and 2. Colour 3 is given by the secondary attributes - only four bits are used. Colour 0 is the same for the whole screen.

Unlike multi-colour text mode, two-colour and four-colour character blocks cannot be mixed.

Note that the secondary attributes in this mode share the same memory as the attributes in the text mode. This will become important when we deal with split screen windows.

SPRITES

In this context, the word 'sprite' refers to one of Laser BASIC's software sprites which must not be confused with the 64's own very powerful hardware sprites. The use of hardware sprites is also allowed from Laser BASIC and this is dealt with in a later section.

WHAT IS A SPRITE?

A sprite is an area of memory of user-defined size in which graphics can be stored and manipulated before being displayed on the screen. When displayed, a sprite appears as a rectangular graphics character. The sprites are held in a special area of memory which is initially set to 7k in size, although this can be changed. The size of a sprite is measured in character blocks. A sprite can be up to 255-character blocks in height or width.

HIRES SPRITES

There are two types of sprites: hires sprites and character sprites. Hires sprites hold data to be displayed in one of the two hires modes. The sprite contains three separate parts, or 'fields'.

1. The data field. This holds the pixel data, and therefore uses up eight bytes for every character block.
2. The primary attribute field. This holds the primary attributes, using up one byte for each character block.
3. The secondary attribute field. This is only used if the sprite is to be displayed in multi-colour mode, and it holds the secondary attributes, again using one byte for each character block.

If the sprite is to be used in two-colour hires mode, the secondary attribute field still exists but is not used.

CHARACTER SPRITES

The other type of sprite is called a character sprite and is used to hold data to be displayed in one of the text modes. It has only two fields:

1. The attribute field. This holds the characters to be displayed, using up one byte per character.
2. The secondary attribute field. This holds the attributes for each character, again using up one byte per character.

So, the data in a character sprite uses up a fifth as much space as it does in a hires sprite of the same size (2 bytes per character block instead of 10). The nomenclature used (attribute field containing characters) may seem to be confusing, but as you will see later, characters in text mode are manipulated using the same commands as attributes in hires mode.

LASER BASIC FOR THE COMMODORE 64

SPRITE NUMBERS

Each sprite is given a number from 1 to 254. A character sprite and a hires sprite cannot both use the same number. The hires screen is treated as a hires sprite, number 0, of width 40 character blocks and height 25. The text screen is a 40x25 character sprite, number 255. The screens do not use up any space in the 7k sprite memory. The hires and text screens share the same secondary attribute memory.

Since the screens are treated as sprites in this way, any command that can be used on a sprite will also work with the screen, but its operation will be immediately apparent.

SPRITE UTILITIES

Now that we have covered the principles involved, we will look at some commands which are used to create and delete sprites.

DELETING ALL SPRITES (RESET)

RESET removes all sprites that are present, and it does not have any parameters

Example:

```
200 INPUT "COLD OR WARM START (C/W) ";A$
210 IF LEFTS(A$,1) ="C" THEN RESET
```

CREATING A HIRES SPRITE (SPRITE)

SPRITE is used to create a hires sprite. It is followed by three parameters, separated by commas. The parameters are: the sprite number, width in character blocks and height in character blocks.

Example: Create sprite No. 3 of width 16 characters and height 4 characters.

```
SPRITE 3,16,4
```

You cannot redefine a sprite that already exists. This will give a "?REDEF'D SPRITE ERROR". Also, if there is not enough memory left, "?OUT OF MEMORY ERROR" is printed.

Each hires sprite created uses up $7+10 \times \text{width} \times \text{height}$ bytes. To demonstrate this, type RESET to make sure that no sprites are in the memory.

Now type the following: **PRINT SFRE**

This prints 7167 which is the number of bytes left for sprite storage.

Now type: **SPRITE 5,20,5**

This sprite should use up $7+10 \times 20 \times 5 = 1007$ bytes, leaving $7167 - 1007 = 6160$ bytes free.

Type: **PRINT SFRE** again to check this.

CREATING CHARACTER SPRITES (CSPRITE)

CSPRITE is similar to SPRITE, and it takes the same parameters. However, it creates a character sprite rather than a hires sprite. Each sprite only uses up $7+2 \times \text{width} \times \text{height}$ bytes, so that creating a sprite using CSPRITE 5,20,5 would use up $7+2 \times 20 \times 5 = 207$ bytes.

LASER BASIC FOR THE COMMODORE 64

DELETING A SPRITE (WIPE)

WIPE removes a sprite from sprite memory. It can be used to delete both hires and character sprites. The command should be followed by the number of the sprite to be deleted.

Example: delete sprite number 6

```
WIPE 6
```

If you try to delete a sprite that doesn't exist, a "?NO SUCH SPRITE ERROR" will be printed. Also, attempts to delete either the hires or text screens (sprites 0 or 255), will give a "?WIPE ERROR".

FINDING A SPRITE'S ADDRESS

```
(DFA,AFA,AFA2)
```

DFA, AFA and AFA2 are functions which return the addresses of the data field, attribute field and secondary attribute field respectively. They should be followed by the sprite number in brackets.

Example: Print out the data field address of sprite zero, the hires screen.

```
PRINT DFA(0)
```

This should give a result of 57344. Now type:

```
PRINT WID; HGT
```

'40 25' is printed; after using DFA, AFA or AFA2, the width and height of the sprite are held in WID and HGT. WID and HGT are actually 'sprite variables' which will be explained shortly.

As mentioned earlier, character sprites have no data field, and if you use DFA in an attempt to find the data field address of a character sprite, you will get the same result as the attribute field address. Also, if you try to find the DFA, AFA or AFA2 of a non-existent sprite, all three will give a result of -1 Therefore this can be used to check if a sprite exists or not.

Example: Write a procedure to delete a sprite without giving a error message if it doesn't exist.

```
100 LABEL DELETE(SPTNUM)  
110 IF DFA(SPTNUM) <> -1 THEN WIPE SPTNUM  
120 PROCEND  
130 `
```

Example: Write a procedure which takes two formal parameters s1 and s2. A new sprite s2 should be created of the same size and type (hires or character) as sprite s1.

```
350 `  
360 LABEL SPTCOPV(S1,S2)  
370 IF DFA(S1) = -1 THEN STOP  
380 IF DFA(S1) = AFA(S1) THEN CSPRITE S2, WID, HGT ELSE SPRITE  
S2, WID, HGT  
390 PROCEND  
400 `
```

The actual addresses returned by DFA, AFA and AFA2 are unlikely to be needed except In advanced applications.

Example: This program uses DFA, AFA and AFA2 to printout details of all the sprites in memory.

LASER BASIC FOR THE COMMODORE 64

```
10 FOR N=1 TO 254
15 PRINT "CHECKING FOR SPRITE NO: "; N
20 CIF DFA(N) > 0
30 PRINT "SPRITE NUMBER: "; N
40 CIF DFA(N) <> AFA(N)
50 PRINT "HIRES SPRITE."
60 PRINT "          PIXEL DATA ADDRESS ="; DFA(N)
70 ELSE
80 PRINT "CHARACTER SPRITE."
90 CEND
100 PRINT "PRIMARY   ATTRIBUTE ADDRESS ="; AFA(N)
110 PRINT "SECONDARY ATTRIBUTE ADDRESS ="; AFA2(N)
120 PRINT "WIDTH ="; WID; "CHARACTERS."
130 PRINT "HEIGHT ="; HGT; "CHARACTERS."
140 PRINT :PRINT
150 REPEAT:GETA$:UNTIL A$=""
160 CEND
170 NEXT N
```

Available on disk as "ex check sprite"

SPRITE VARIABLES

WHAT ARE SPRITE VARIABLES?

In the previous section, the sprite variables WID and HGT were mentioned. There are actually 13 such sprite variables:

Name	Usage
SPN	Sprite number
COL	Column within sprite
ROW	Row within sprite
WID	Width of a sprite or sprite window
HGT	Height of a sprite or sprite window
SPN2	Sprite number 2
COL2	Column within sprite 2
ROW2	Row within sprite 2
NUM	Number of pixels to be scrolled, number of sides in a polygon, etc.
INC	Inclination of a polygon in degrees
ATR	Current attribute value
CCOL	Column within sprite where collision was detected
CROW	Row within sprite where collision was detected.

These sprite variables can be treated as normal variables. They can have their values read:

```
cr=ROW+1
```

Or they can have values assigned to them:

```
ROW=cr-ROW2
```

LASER BASIC FOR THE COMMODORE 64

Note that you cannot use the LET command when assigning a value to a sprite variable. For example, the following will give a syntax error:

```
LET ROW=CR-ROW2
```

WHY ARE SPRITE VARIABLES USED?

Sprite variables are used by Laser BASIC when passing values to the graphics commands. For example, the command SPRITE takes three parameters: SPN, WID and HGT. When you type SPRITE 10,2,4 Laser BASIC sets SPN to 10, WID to 2 and HGT to 4. It then executes a machine code sub-routine within Laser BASIC to carry out SPRITE.

This subroutine uses the values that have been stored in SPN, WID and HGT. If, after executing the above command, you were to type

```
PRINT SPN;WID;HGT
```

you would get

```
10 2 4
```

which is what you used as the parameters. Unless stated otherwise, none of the graphics commands change any of the sprite variables.

USING COMMANDS WITHOUT PARAMETERS

Most of the graphics commands can be used without any parameters. In this case the current values of the sprite variables are used. So, instead of using

```
SPRITE 10,2,4
```

you could use

```
SPN= 10:COL=2:ROW=4:SPRITE
```

although there is no advantage in doing it this way in this case. In some cases, however, this can make parts of your Laser BASIC program faster and more compact.

A few of the graphics commands do not pass parameters using the sprite variables and therefore cannot be used without parameters. Here is a list of them:

MODE,	SCRX,	WINDOW,	TBORDER,	HBORDER,	TPAPER,
HPAPER,	INK,	HON,	HOFF,	HSET,	H4COL,
H2COL,	H1COL,	H3COL,	HEXX,	HSHX,	HEXY,
HSHY,	HX,	HY,	HCOL,	OVER,	UNDER,
SCRY,	RASTER,	BG0,	BG1,	BG2,	BG3,
MCOL1,	MCOL2,	MCOL3,	FGND,	BGND,	RSYNC.

Also, none of the functions can be used without parameters.

From now on, any new commands, except for the ones listed above will be printed with a list of its parameters after it like this:

```
SPRITE SPN, WID, HGT
```

Here is a list of the commands encountered so far, with parameters.

```
SPRITE SPN,WID,HGT
```

LASER BASIC FOR THE COMMODORE 64

```
CSPRITE SPN,WID,HGT
WIPE SPN
DFA (SPN)
AFA (SPN)
AFA2 (SPN)
```

Remember that because DFA, AFA, and AFA2 are functions, their parameters cannot be missed out.

SAVING and LOADING Sprites

Because sprites are not saved and loaded automatically with your BASIC program, separate commands are used to save and load them.

SAVING SPRITES

(STORE, DSTORE)

STORE is used to save to tape, and DSTORE is used with disk. Like the ordinary SAVE command, STORE or DSTORE should be followed by the filename, if any.

Examples:

- Save sprites to tape with no filename: STORE
- Save sprites to tape under filename "SPT1" STORE "SPT1"
- Save sprites to disk under filename "EX3" DSTORE "EXT"

Like BASIC's existing SAVE command, it is also possible to specify a device number. So, if you have a disk drive configured as device 9, you can save the sprites to it using either:

```
STORE "SPT",9
```

or

```
DSTORE "SPT",9
```

The only difference between STORE and DSTORE is in the device number which is used if you do not specify it.

LOADING SPRITES (RECALL, DRECALL, MERGE and DMERGE)

RECALL or DRECALL are used to load sprites into the computer from tape or disk respectively. Any sprites which were present previously are lost. Like STORE and DSTORE, the filename and device number, if any, should be placed after this command.

RECALL or DRECALL can be included in a program as well as being typed in from direct mode.

Examples:

Command	Action
RECALL	Load first set of sprites found from tape:
RECALL " DEMO2"	Load sprites under filename "DEMO2" from tape:
DRECALL "EX3"	Load sprites under filename "EX3" from disk:
DRECALL "EX3",9	Load sprites under filename "EX3" from disk drive configured as device number 9:

There may be occasions on which you wish to load some new sprites into memory while keeping the sprites that already exist. This is done by using MERGE or DMERGE in place of RECALL or DRECALL.

LASER BASIC FOR THE COMMODORE 64

Please remember that there is nothing to stop you from loading in more sprites than there is room for in memory. If this happens, an error message is printed:

?sprite error

This means that the sprites are corrupted. If an error like this occurs, it is normally best to type in "RESET" immediately to prevent any further corruption from taking place. Also, any excess sprites may have overflowed into character memory, resulting in some strange shapes on the screen.

It is normally best to keep a note of the approximate length in bytes of any sprites you have saved, and check that there is enough space left before loading them in. The amount of free space left can be printed using "PRINT SFRE".

RENUMBERING SPRITES - (RESEQ)

RESEQ renumbers all the sprites in memory. The first sprite that was defined is number 1, the second is number 2, and so on

RESERVING MORE SPACE FOR SPRITES (RESERVE)

In some applications, 7168 bytes is not enough room for the sprites. RESERVE can be used to allocate more memory. Unfortunately, there will be less room for your BASIC program. The RESERVE command should be followed by the number of bytes needed for sprites. For example. To reserve 10,000 bytes, use:

```
RESERVE 10000
```

The minimum space that can be reserved without giving an error message is 7168. The maximum is 24829 bytes, or more if the multitasking or turbo tape are not present. This will leave you with no space for a BASIC program or variables.

Although RESERVE keeps your BASIC program and sprites intact clears out all BASIC variables in the same way as the CLR command

SETTING UP THE DISPLAY

Laser BASIC can be used with any of the display modes described at the beginning of this chapter. You can also have the top half of the screen in hires mode, and the lower half in text mode, using the WINDOWV command.

DISPLAY INITIALISATION (INIT)

The INIT command sets the display to two-colour text mode and initialises the entire graphics system. It should always be placed at the start of any program which uses graphics.

DISPLAYING HIRES GRAPHICS (HIRES)

The HIRES command puts the computer into hires mode. If you type HIRES at the keyboard, the screen will flash as the computer goes into hires mode and back into text mode. It must go back into text mode if you are to see the next line that you type in. To make it stay in hires mode. type

```
HIRES:REPEAT:UNTIL FALSE
```

This puts the computer into an infinite loop after going into hires mode, allowing the hires screen to be displayed. At the moment the screen is probably full of garbage because you haven't cleared it yet. Press RUN / STOP to break out of the loop.

LASER BASIC FOR THE COMMODORE 64

GOING BACK INTO TEXT MODE (LORES)

The opposite command to HIRES is LORES which puts the computer back into text mode.

As an example, here is a program which switches between hires and lores modes:

```
10 HIRES
20 FOR I=0 TO 100 : NEXT I
30 LORES
40 FOR I=0 TO 100 : NEXT I
50 GO TO 10
```

If you run this program, you will notice that no flicker occurs when changing between modes. This is because Laser BASIC automatically synchronises all mode changes to the display scan. Press RUN/STOP to break out of the program.

DISPLAYING EXTENDED BACKGROUND COLOUR MODE (EBACK)

The EBACK command, when used after LORES, will put the display into extended background colour mode.

Type EBACK and then LIST the program that you typed in above. The background colour of each character depends on the display code:

Display Code	Background Colour
0 to 63	light grey
64 to 127	light blue
128 to 191	yellow
192 to 255	white

Remember that the display code is not the same as the ASCII character code (the ASCII code for any character can be obtained by using ASC, e.g .

```
PRINT ASC ("a" ) .
```

Most of the characters on the screen have display codes from 0 to 63 and hence have the normal light grey background. The display codes for the capital letters are between 64 and 127, so they are displayed with a light blue background. The cursor is yellow because it works by adding 128 to the character code of the character underneath it. The character codes from 128 onwards are normally used for reversed characters. If you move the cursor over one of the keywords on the listing, you will see that it is now white because the cursor adds 128 to a display code from 64 to 127, giving a display code from 192 to 255.

Type " LORES". The computer now goes back to normal text mode

COLOUR CONSTANTS

Each of the sixteen colours that the Commodore 64 can display is a keyword in Laser BASIC. The keywords are:

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

LASER BASIC FOR THE COMMODORE 64

GRAY2 .GREEN .BLUE GRAY3

Each one of these is in fact a predefined constant with a value from 0 to 15 which is the corresponding attribute value. For example. < you type

```
PRINT GREEN
```

you will get 5,

```
PRINT BROWN
```

gives 9, and so on.

Three of the colours have a full stop in front of them. This is to be taken as meaning 'light'. For example, .BLUE is light blue. There are three shades of grey - GRAY1 , GRAY2 and GRAY3. GRAY3 is the lightest, and GRAY1 the darkest.

HOW TO CHANGE THE BORDER COLOUR (TBORDER , HBORDER)

TBORDER is used to change the border colour used In text mode The command is simply followed by the colour.

Example: Change the text border colour to black.

```
TBORDER BLACK
```

HBORDER is analogous to the TBORDER command, but it is used for the hires screen's border colour. The text border colour and hires border colour are independent of one another.

Example: Change the hires border colour to white and go into hires mode.

```
10 HBORDER WHITE
20 HIRES
30 GOTO 30
```

If you run this program, you must hit the RUN / STOP key to exit.

HOW TO CHANGE THE BACKGROUND COLOUR (TPAPER, HPAPER)

In a similar way to TBORDER, the TPAPER command changes the background colour in text mode.

Example: Change the text background colour to light blue.

```
TPAPER .BLUE
```

HPAPER s used in the same way to change the background colour in the hires mode. This background colour only applies to multi-colour hires mode which will be dealt with fully later.

Because the computer sees the colours as numbers, TPAPER, TBORDER, HPAPER ard HBORDER can use any numeric expression. For example, you might want to do something like this:

```
50 tp=BLUE
60 up=RED
```

later in the program

```
900 IF md THEN TBORDER tp ELSE TBORDER up
```

LASER BASIC FOR THE COMMODORE 64

Example: Here is a program which changes the border colour through all fifteen colours until terminated by the RUN/STOP key.

```
10 REPEAT
20 FOR I = 0 TO 15
30 TBORDER I
40 FOR J = 0 TO 100 : NEXT j
50 NEXT I
60 UNTIL FALSE
```

HOW TO CHANGE THE EXTENDED BACKGROUND COLOURS

In extended background colour mode, there are four background colours. These are set up using the commands BG0, BG1, BG2 and BG3.

CHARACTER DISPLAY CODES	BACKGROUND SET BY
0 to 63	BG0 colour
64 to 127	BG1 colour
128 to 191	BG2 colour
192 to 255	BG3 colour

Each of these commands is followed by the colour to be used. BG0 is in fact a synonym for TPAPER. Example: This program fills the screen with letter "a"s.

For each letter printed, one of the four background colours is selected at random. Each background colour is then repeatedly set to random values until RUN / STOP is pressed.

```
10 INIT:EBACK
20 N$(0) = CHR$(65)
30 N$(1) = CHR$(193)
40 N$(2) = CHR$(18)+CHR$(65) +CHR$(146)
50 N$(3) = CHR$(18)+CHR$(193)+CHR$(146)
60 PRINT CHR$(147)
70 FOR I = 1 TO 999
80 PRINT N$(RND(1) * 4) ;
90 NEXT I
100 REPEAT
110 BG0 RND(1) * 16
120 BG1 RND(1) * 16
130 BG2 RND(1) * 16
140 BG3 RND(1) * 16
150 UNTIL FALSE
```

Lines 20 to 50 set up four strings in the array n\$ which print display codes 1, 65, 129 and 193 respectively. Remember that the display codes are different from the ASCII codes used in the program. CHR\$(18) (RVS ON) and CHR\$(146) (RVS OFF) are used to get display codes greater than 128.

LASER BASIC FOR THE COMMODORE 64

Line 60 clears the screen. In lines 70 to 90. the screen is filled with 999 characters taken randomly from n\$. The rest of the program is a loop in which the background colours are assigned random values.

When you press the RUN/STOP key to exit from this program. you may find that the screen colours make any text on the screen unreadable. The simplest way to put everything. including the screen colours. back to the original settings is to keep the RUN/STOP key pressed and then hit RESTORE, or type INIT.

SETTING UP THE PRINTING COLOUR (INK)

Normally. Laser BASIC prints all characters on the screen in blue. Each time a character is printed. the corresponding position in attribute memory is loaded with the blue attribute. To change the colour used, you can use the INK command, which is followed by the colour.

Example: Change the text printing colour to purple.

```
INK PURPLE
```

Example: This program will print the numbers 0 to 15, each with a different INK colour.

```
10 TPAPER GRAY3  
20 FOR I = 0 TO 15  
30 INK I  
40 PRINT I;  
50 NEXT I  
60 INK BLUE
```

In the last line. the printing colour is set back to its normal value, blue. If you run this program, you will notice that the last number, 15, cannot be seen. This is because it is printed in GRAY3, the same colour as the background Type TPAPER WHITE, and 15 appears, while 1, which was printed in white. vanishes.

GOING INTO MULTI-COLOUR MODE (MULTI , MONO)

So far, we have only shown how to set up the screen in one of the two-colour modes. To go into multi-colour mode, use the command MULTI in conjunction with either LORES or HIRES. Thus, to go into multi-colour text mode, you would use:

```
LORES :MULTI
```

The LORES command need not be included if the computer is already in text mode. In that case, you could just use MULTI on its own. To go into multi-colour hires mode, use:

```
HIRES :MULTI
```

The opposite command to MULTI is MONO. MONO simply takes the computer back into ordinary two-colour mode.

To see multi-colour text mode working, try this:

First type INIT to make sure that the screen colours etc. are all back to the initial settings. Now type MULTI. Although the computer is now in multi-colour text mode, you will see no difference in the display this is because the current INK colour is blue, which has a value of 6. Remember that characters are only displayed in four-colour mode if they have an attribute value greater than seven. The eight colours which have attribute values less than or equal to seven are:

LASER BASIC FOR THE COMMODORE 64

BLACK **PURPLE** **WHITE** **GREEN**
RED **BLUE** **CYAN** **YELLOW**

To display characters in 4 colours, we have to add 8 to the ink colour Now type

INK BLUE + 8

The computer will print all characters in four-colour mode because the attribute value is greater than seven. Each pixel within a character is twice as wide as it was before, and can be one of four colours:

COLOUR NUMBER	SET BY
0 (00)	BGO colour (or TPAPER colour)
1 (01)	BG1 colour
2 (10)	BG2 colour
3 (11)	INK colour+8 (i.e. attribute value)

Colours 0,1 and 2 are the same for the whole screen, and are set up by the commands BGO, BG1 and BG2. These are the same commands that are used when setting up the background colour in extended background colour mode. Colour 3 is different for each character because it depends on the attributes A more detailed discussion on the uses of multi-colour mode graphics will be given later when more commands have been covered. Since multi-colour and extended background colour mode cannot be combined, the MULTI command will cancel EBACK and vice versa.

MIXING DISPLAY MODES (WINDOW)

The WINDOW command lets you mix hires and text modes on the screen. Between 1 and 23 lines at the top of the screen are hires graphics, while the lower part is text.

For example, to set the top 16 lines of the screen to hires and the lower 9 to text. you would use:

WINDOW 16

If you use a number greater than 23, the command puts no hires window on the screen

The hires and text paper colours still remain independent. However, MULTI and MONO apply to the whole screen, so it is not possible to mix multi-colour text and two-colour hires, for example.

The hires border colour is used for the whole screen when a window is setup.

If you try to use a disk drive or serial- bus printer while a window is set up, the screen will flicker. If you are going to use tape, it is best to remove the window from the screen first, using:

LORES

or

WINDOW 0

WINDOW 0 does not cancel extended background colour mode.

SCREEN MODES SUMMARY

Here is a list of the commands necessary to go into any of the screen modes in Laser BASIC:

Two-colour text mode:	LORES:MONO
Multi-colour text mode:	LORES:MULTI
Extended background colour mode:	LORES:EBACK
Two-colour hires mode:	HIRES:MONO

LASER BASIC FOR THE COMMODORE 64

Multi-colour hires mode:	HIRES:MULTI
Top n lines of the screen two-colour hires, rest two-colour text:	WINDOWn:MONO
Top n lines of the screen multi-colour hires, rest multi-colour text:	WINDOWn:MULTI
Top n lines of the screen two-colour hires, rest extended background colour mode text:	WINDOWn:EBACK

SETTING UP SOFTWARE MODES

So far, we have shown how to make the computer display graphics in all the screen modes. However, when Laser BASIC plots a point inside a sprite, for example, it needs to know what screen mode it will be displayed in so that it can set up a point correctly. To give you greater power and flexibility, Laser BASIC does not assume that the point being plotted will be displayed in the same mode that the screen is currently in. Instead, there are separate commands which set up the screen mode that Laser BASIC prepares the data for. This way, you can display the screen in one mode while you set up sprites to be displayed in another mode.

CONTROLLING ATTRIBUTES (ATTOFF, ATTON, ATT2ON)

Depending on the graphics that are being displayed, you may want to use no attributes, primary attributes only, or both sets of attributes. These three possibilities are covered by the commands ATTOFF, ATTON and ATT2ON.

COMMAND	Effect
ATTOFF	No attributes are used.
ATTON	Only primary attributes are used.
ATT2ON	Both sets of attributes are used.

The precise way in which each graphics command is affected by ATTOFF, ATTON and ATT2ON is slightly more complex than this. This will be dealt with as we come to each new command.

However, for most purposes you should use ATTON in two-colour hires mode and ATT2ON in all other modes.

When Laser BASIC is first loaded, it is set up to use primary attributes only.

SOFTWARE TWO-COLOUR AND FOUR-COLOUR MODES

The commands S2COL and S4COL set up the screen mode that Laser BASIC uses when plotting points, moving sprites around and so on.

After S2COL is executed, any graphics commands which use pixel data assume that it is to be displayed in two-colour mode. Similarly, S4COL makes Laser BASIC assume that the pixel data will be displayed in four-colour mode.

Because these two commands only affect the way that pixel data is treated, they do not apply to character sprites. Laser BASIC is initially set up to work with two-colour mode pixel data.

LASER BASIC FOR THE COMMODORE 64

SCREEN MODES SUMMARY WITH SOFTWARE MODES

Here is a new version of the SCREEN MODES SUMMARY presented earlier, with commands added to put the Laser BASIC software into the same mode as the computer's hardware.

Two-colour text mode:	LORES:MONO:ATT20N
Multi-colour text mode:	LORES:MULTI:ATT20N
Extended background colour text mode:	LORES:EBACK:ATT20N
Two-colour hires mode:	HIRES:MONO:S2COL:ATT0N
Multi-colour hires mode:	HIRES:MULTI:S4COL:ATT20N
Top n lines of the screen two-colour hires, rest two colour text:	WINDOWn:MONO:S2COL:ATT20N
Top n lines of the screen multi-colour hires, rest multi-colour text:	WINDOWn:MULTI:S4COL:ATT20N
Top n lines of the screen two-colour hires, rest extended background colour mode text:	WINDOWn:EBACK:S2COL:ATT20N

CLEARING SPRITES (SETA, WCIR, SCIR)

The three commands SETA, WCIR and SCIR are used to clear sprites. When the data field of a sprite is cleared, it is set to the background colour. The attribute fields are set to the value in the sprite variable ATR.

SETTING UP THE ATTRIBUTE VALUE

(FGND, BGND, MCOL0, MCOL1, MCOL2)

The sprite variable ATR holds the attribute which is used when clearing a whole sprite or part of it. ATR contains the attributes for both the primary and secondary attribute fields. The exact way in which ATR holds the various parts of the attribute need not concern you at the moment, since it is important only for advanced applications using laser BASIC.

If you are using two-colour hires mode, ATR can be set up using the commands FGND and BGND:

COLOUR	SET BY
FOREGROUND	FGND colour
BACKGROUND	BGND colour

Example: Set up ATR for a blue foreground and a white background:

FGND BLUE:BGND WHITE

In multi-colour hires mode, ATR is set up using MCOL1, MCOL2 and MCOL3.

COLOUR NUMBER	SET BY
(0)	(HPAPER colour - same for whole screen)
1	MCOL1 colour
2	MCOL2 colour
3	MCOL3 colour

LASER BASIC FOR THE COMMODORE 64

Example: Set up ATR so that colour 1 is black, 2 is cyan and 3 is red.

```
MCOL1 BLACK:MCOL2 CYAN:MCOL3 RED
```

In the text modes the character display codes are held in the attribute field and the colour of each character goes in the secondary attribute field.

The simplest way to set up ATR for one of the text modes is:

1. Set ATR equal to the character code you want to use.
2. Use "MCOL3 colour" to set up the colour.

Example: Set up ATR with character code 102 displayed in green in two-colour text mode:

```
ATR=102:MCOL3 GREEN
```

Example: Set up ATR with character code 211 displayed in four-colour mode. Colour 3 should be RED.

```
ATR=211 :MCOL3 RED+8
```

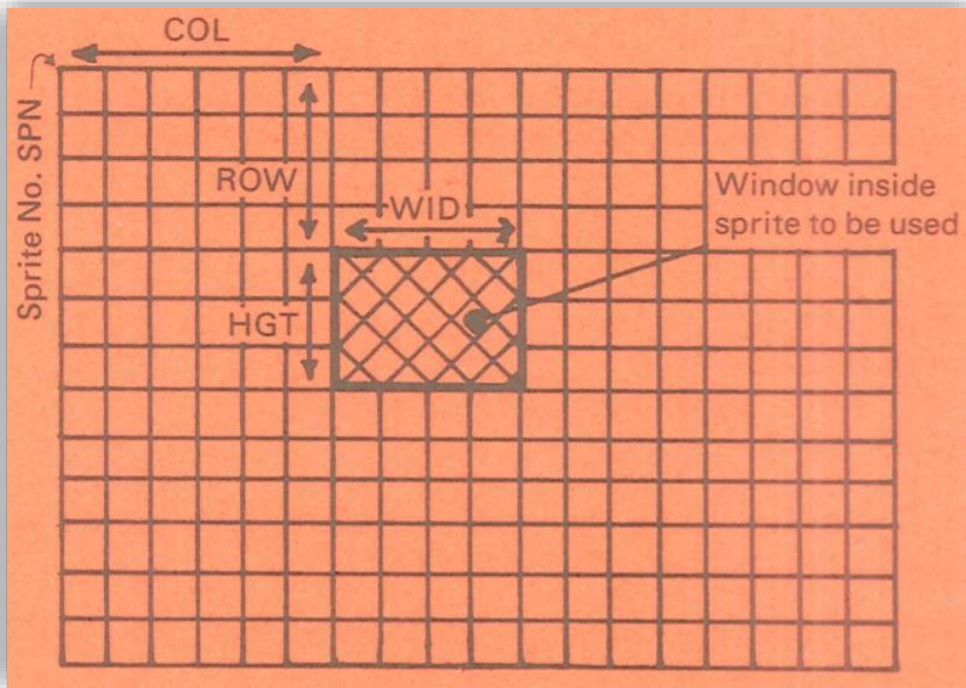
NOTE: BASIC Lightning used a command called SETATR which is now superseded by FGND, BGND, MCOL1, MCOL2 and MCOL3. Although SETATR can still be used, it can be converted to the new commands as follows:

OLD SETATR COMMANDS	NEW COMMANDS
SETATR 0,a,b	FGNDa:BGNDb
SETATR a,b,c	MCOL3a:MCOL1b:MCOL2c

THE CONCEPT OF SPRITE WINDOWS

Many of the graphics commands in laser BASIC operate on a rectangular block, or window, inside a sprite:

LASER BASIC FOR THE COMMODORE 64



This window is specified by the sprite variables COL, ROW, WID and HGT which are used as parameters to the graphics commands.

In the diagram above, the sprite itself is 16 character blocks wide, and 13 character blocks high. The width of the window is 4 characters, and the height is 3 characters, so WID=4 and HGT=3. COL and ROW are set to the position of the top left hand corner of the window. In this case, COL=6 and ROW=4.

SETTING OR CHANGING ATTRIBUTES (SETA)

The SETA command is used to set up the primary and secondary attributes in a rectangular sprite window. It has SIX sprite variables as its parameters:

SETA SPN, COL, ROW, WID, HGT, A TR

SPN is the sprite to be used. COL and ROW specify the top left hand corner of the window, while WID and HGT are its dimensions. ATR is the attribute value to be used.

If the ATT20N command has been executed, both sets of attributes are set up. Otherwise, only the primary attributes are set.

Examples:

This program sets the attributes in sprite 2 to blue foreground and cyan background. Sprite 2 is a hires sprite to be displayed in two-colour hires mode.

```
10 ATTON
20 I = DFA(2)
30 FGND BLUE
40 BGND CYAN
```

LASER BASIC FOR THE COMMODORE 64

```
50 COL=0 : ROW=0 : SETA
60 INK BLUE
```

Notice that DFA is used to obtain the width and height of sprite 2, so that the whole sprite is covered by the window.

Example: Set a 4x4 block in the centre of the two-colour mode text screen to display code 102 with a red background.

```
10 ATT20N
20 ATR=102
30 MCOL3 RED
40 SETA 255,18,10,4,4,ATR
```

SWITCHING ATTRIBUTE FIELDS (SWITCH, NORM)

After using the SWITCH command, Laser BASIC will treat the primary attribute field of each sprite as the secondary attribute field and vice versa. The NORM command puts the attributes back to normal operation.

SWITCH is useful if you wish to set only the secondary attribute field of a sprite.

For example, the following program will set all characters on the two colour mode text screen to yellow:

```
10 ATTON
20 ATR=YELLOW
30 SWITCH
40 SETA 255,0,0,40,25,ATR
50 NORM
```

Note that in this case, ATR is simply equal to the colour to be displayed. When Laser BASIC is executing the SETA command in line 40, it thinks that it is setting up the primary attribute field. It is in fact setting up the secondary attribute field because the SWITCH command has been used. If you delete line 30, and try running the program, the screen is filled with letter "g"s. This is because the primary attribute field, which holds the character codes, is now being set up.

CLEARING PIXEL DATA (WCLR, SCLR)

The WCLR command is similar to SETA. It has the same parameters:

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

WCLR can only be used with hires sprites. All the pixel data inside the window is cleared to the background colour. Then, providing that the ATTOFF command has not been used, SETA is executed.

Attribute Mode	Fields Cleared
ATTOFF	Pixel data only.
ATTON	Pixel data and primary attributes
ATT20N	Pixel data, primary and secondary attributes.

Example: Clear a 4x4 character block at the top left hand corner of sprite 3. This sprite is to be displayed in multi-colour mode. Colour 1 should be purple, colour 2 yellow and colour 3 green.

```
10 ATT20N
```

LASER BASIC FOR THE COMMODORE 64

```
20 MCOL1 PURPLE
30 MCOL2 YELLOW
40 MCOL3 GREEN
50 WCLR 3,0,0,4,4,ATR
```

The SCLR command operates in the same way as WCLR does. However, it is used to clear a sprite rather than a window within one. It has two parameters:

```
SCLR SPN,ATR
```

SPN is the sprite to be cleared and ATR is the attribute to be used.

Example: Clear the pixel data only on the hires screen:

```
ATTOFF: SCLRO,A TR
```

The value of ATR is irrelevant here. It is not used because the attribute fields are not being cleared.

When you create a hires sprite using the SPRITE command, it is automatically cleared in the same way as SCLR. Similarly, a character sprite created with CSPRITE is cleared as with SETA.

HIRES GRAPHICS COMMANDS

(MODE, PLOT, BOX, DRAW, POLY, FILL, POINT)

The seven so-called 'hires graphics' commands allow you to set individual pixels within a hires sprite, draw shapes and fill in large areas. They are not designed to be used for high-speed animation; the PUT and GET family of commands are used for this purpose, and they will be dealt with in the next section. None of these commands use the attribute field: only the pixel data part of a sprite is altered.

HOW TO SET UP THE PLOTTING COLOUR (MODE)

When one of the hires graphics commands sets a pixel inside a sprite, there are several different ways in which it can be done. If two-colour mode plotting is taking place (i.e. the S2COL command has been used), there are three possibilities:

- I. Set the pixel to the background colour.
- II. Set the pixel to the foreground colour.
- III. Invert the pixel.

In four-colour mode, there are five ways of setting the pixel:

- I. Set the pixel to colour 0.
- II. Set the pixel to colour 1.
- III. Set the pixel to colour 2.
- IV. Set the pixel to colour 3.
- V. Invert the pixel.

Inverting a pixel in two-colour mode is quite simple. If the point is already set to the foreground colour, it is changed to the background colour, and vice versa. In four-colour mode, it is more complex:

COLOUR BEFORE INVERSION	COLOUR AFTER INVERSION
0	3
1	2

LASER BASIC FOR THE COMMODORE 64

2	1
3	0
4	4

The plotting mode is set up by the MODE command. It should be followed by number from 0 to 4 which specifies how pixels are to be set:

COMMAND	IN TWO-COLOUR MODE, PIXELS SET TO:	IN FOUR-COLOUR MODE, PIXELS SET TO:
MODE0	background	colour 0
MODE1	background	colour 1
MODE2	foreground	colour 2
MODE3	foreground	colour 3
MODE4	inverted	inverted

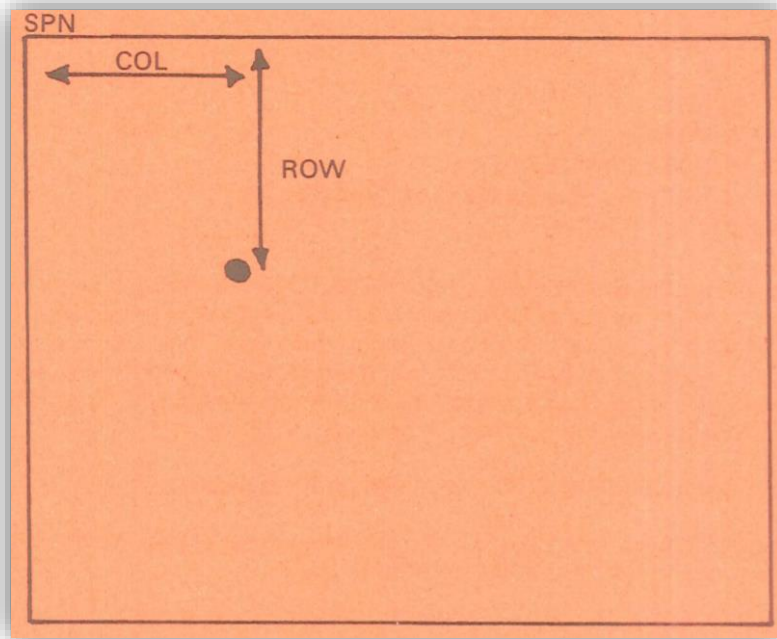
This command influences the behaviour of PLOT, DRAW, BOX, POLY and FILL. The plotting mode remains the same until it is changed again using the MODE command.

SETTING INDIVIDUAL PIXELS

(PLOT)

The PLOT command is used to set an individual pixel within a sprite. It has three parameters:

PLOT SPN, COL, ROW



SPN is the sprite to be used. COL and ROW specify the column and row respectively.

What colour the pixel is set to, or whether the pixel is inverted, depends on the plotting mode specified by the MODE command.

LASER BASIC FOR THE COMMODORE 64

The column and row are measured in pixels, not character blocks, so COL and ROW will be 8 times larger than they would be with most other commands.

In four-colour mode the pixels on the screen are twice as wide. Each character block is four pixels across instead of eight. In this case, COL still has the same scaling as it does in two-colour mode, so two different values of COL refer to the same pixel. For example, ROW=0,COL=0 and ROW=0,COL=1 both refer to the same pixel at the top left of a sprite.

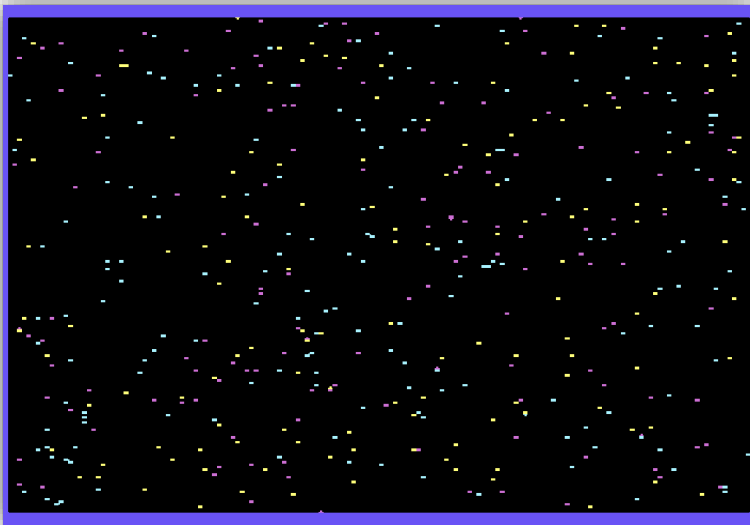
Example 1: This program sets pixels at random on the hires screen in multi-colour mode.

```
10 INIT : MULTI
20 S4COL
30 HBORDER BLUE
40 HPAPER BLACK
50 MCOL1 PURPLE
60 MCOL2 YELLOW
70 MCOL3 CYAN
80 ATT20N
90 SCLR 0,ATR
100 HIRES
110 REPEAT
120 MODE 3 * RND(1) + 1
130 PLOT 0,320*RND(1),200*RND(1)
140 UNTIL FALSE
```

Available on disk as "ex hires 01"

Lines 10 and 20 set up multi-colour mode. The hires border and paper colours are set up in lines 30 and 40. Next, the attribute value is initialised to give plotting col()urs of purple, yellow and cyan, and the screen is cleared. The program goes into an infinite repeat-until loop which sets up a random plotting colour and plots a point on the screen at random. Remember that the hires screen is sprite 0.

You must use the RUN/STOP key to exit from this program.



LASER BASIC FOR THE COMMODORE 64

Example 2: This program plots a three-dimensional shape on the hires screen. It takes just under half an hour to run.

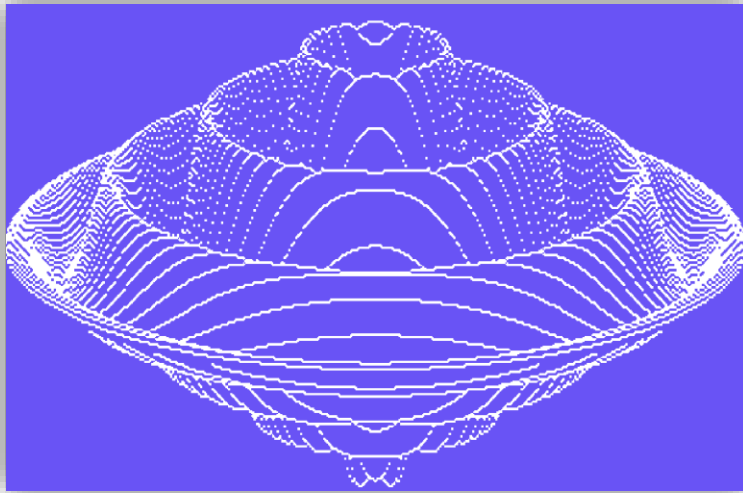
```
10 INIT
20 `
30 FGND    WHITE
40 BGND    BLUE
50 HBORDER BLUE
60 ATTON
70 SCLR 0,ATR
80 HIRES : MODE 3
90 A=160 : B=A*A : C=100 : D=100
100 FOR X=0 TO A
110  S=X*X
120  P=SQR(B-S)
130  I = -P
140  REPEAT
150   R=SQR(S+I*I)/A
160   Q=(R-1) * SIN(24*R)
170   Y=I/3 + Q * D
180   IF I=-P THEN N=Y : M=Y
190   CIF Y<N OR Y>M
200   IF Y<N THEN N=Y ELSE M=Y
210   ROW=C-Y
220   COL=A-X : PLOT
230   COL=A+X : PLOT
240  CEND
250  I=I+4
260  UNTIL I >= P
270 NEXT X
280 REPEAT
290 UNTIL FALSE
```

Available on disk as "ex hires 02"

The first eight lines of the program set up two-colour mode clear the hires screen and go into hires mode. The PLOT commands in lines 220 and 230 are used without parameters. SPN does not need to be set up here because it has already been set to zero in line 70. Hidden line removal is performed by keeping a record of the highest point plotted in m, and the lowest point plotted in n. Only points greater than m and less than 1 are plotted. Line 170 determines the perspective at which the curve is viewed.

LASER BASIC FOR THE COMMODORE 64

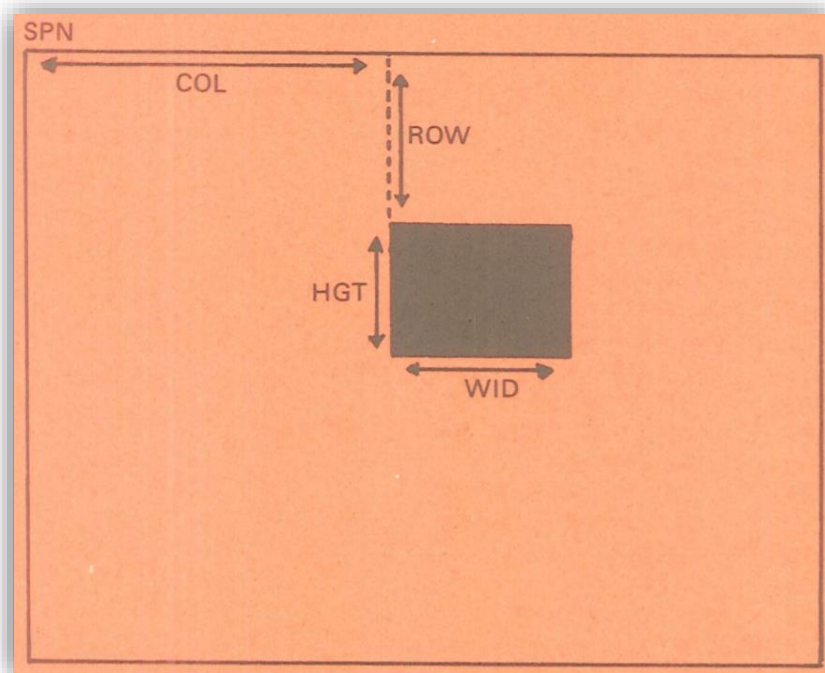
The function for evaluation is given in line 160, as the height q in terms of radius r , and this can be changed to give plots of other functions.



DRAWING RECTANGLES (BOX)

The BOX command draws a rectangular block inside a sprite. It takes five parameters:

BOX SPN, COL, ROW, WID, HGT



SPN is the sprite to be used. COL and ROW specify the top left hand corner of the block. HGT is the height and WID is the width of the block. As when using PLOT, everything is measured in pixels, and the scaling remains the same in both two-colour and four-colour modes. The colour used depends on the value set by the MODE command.

LASER BASIC FOR THE COMMODORE 64

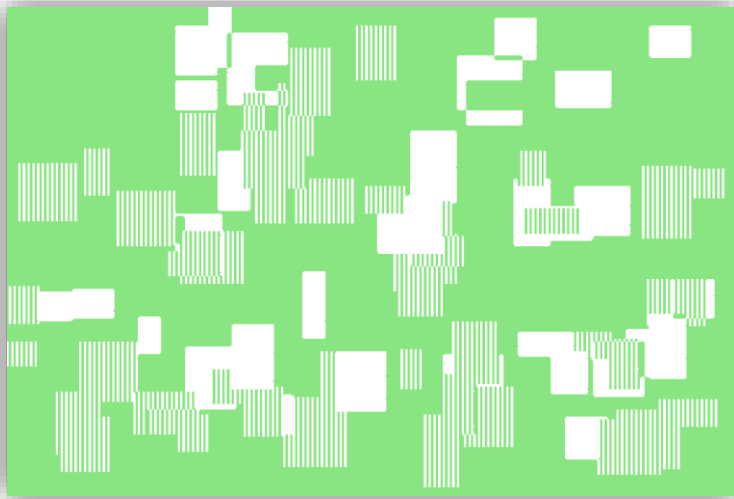
Example 3: This program draws random blocks on the multi-colour hires screen.

```
10 MULTI INIT
20 S4COL
30 HBORDER BLACK
40 HPAPER BLACK
50 MCOL1 WHITE
60 MCOL2 GREEN
70 MCOL3 RED
80 ATT20N
90 SCLR 0,ATR
100 HIRES
110 REPEAT
120 MODE 4*RND(1)+1
130 BOX 0,290*RND(1),170*RND(1),20*RND(1)+10,20 * RND(1) + 10
140 UNTIL FALSE
```

Available on disk as "ex hires 03"

The first ten lines of the program set up the hires screen, and its attributes. The remaining four lines are an infinite loop in which boxes are drawn at random. Line 120 selects a random plotting colour from 1 to 4. Remember that a plotting colour of 4 will result in all pixels inside the box being inverted. In line 130, the box is drawn with random position and size.

The range of numbers allowed do not let COL+WID exceed 319 or ROW+HGT exceed 199. If this were to happen, an error message would be printed since the box would not fit on the screen.

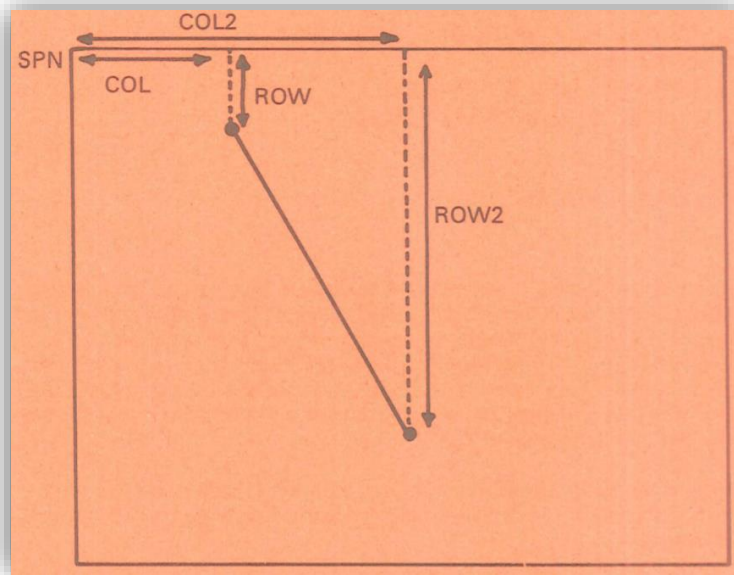


DRAWING LINES (DRAW)

The DRAW command draws a line inside a sprite. It has five parameters.

```
DRAW SPN, COL, ROW, COL2, ROW2
```


LASER BASIC FOR THE COMMODORE 64



SPN is the sprite in which the line is drawn. COL and ROW specify one end of the line whilst COL2 and ROW2 specify the other end. Again, all co-ordinates are in pixels, and the scaling is the same for both hires modes. The colour used when drawing depends on the value set by the MODE command.

Example 4: This program draws moire fringes on the two-colour hires screen. These fringes are created when two lines are drawn close to one another on the screen, due to the alienation or 'jaggies' produced when drawing a line.

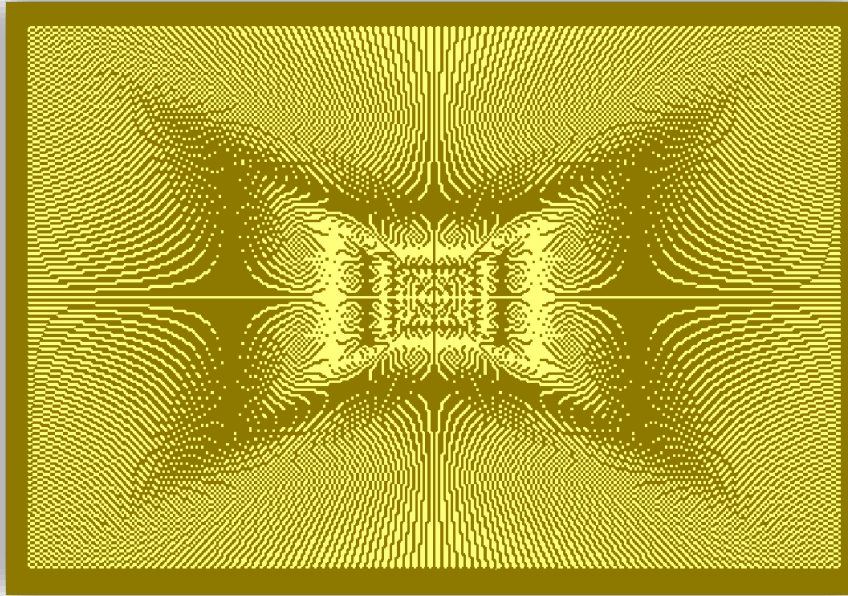
```
10 INIT
20 FGND BROWN
30 BGND YELLOW
40 HBORDER BROWN
50 SPN=0 SCLR
60 HIRES
70 MODE3
80 DRAW 0, 0,199,319,199
90 DRAW 0,319, 0,319,199
100 MODE4
110 FOR I=0 TO 317 STEP 3
120 DRAW 0,I,0,318-I,198
130 NEXT I
140 FOR I=0 TO 197 STEP 2
150 DRAW 0,0,198-I,318,I
160 NEXT I
170 REPEAT
180 UNTIL FALSE
```

Available on disk as "ex hires 04"

Lines 10 to 70 set up the screen, with brown foreground and yellow background. In lines 80 and 90, lines are drawn along the right and bottom edges of the screen. This effectively reduces the size of the

LASER BASIC FOR THE COMMODORE 64

screen, so that the lines drawn in the next part of the program fit without any gaps. In lines 120 to 170 the screen is filled with the diagonal lines which create the fringes. The last two lines of the program are an infinite loop which can only be left by hitting the RUN/STOP key.



Example: This program draws a logo, made up of cubes, on the multicolour hires screen. The logo is contained in the data statements from line 460 onwards.

```
10 DIM M$(19)
20 FOR J=1 TO 19
30 READ M$(J)
40 NEXT J
50 '
60 INIT : S4COL
70 MULTI
80 ATT20N
90 HPAPER BLACK
100 HBORDER BLACK
110 MCOL1 WHITE
120 MCOL2 BLUE
130 MCOL3 BLUE
140 SCLR 0,ATR
150 HIRES
160 '
170 FOR J=19 TO 1 STEP-1
180 FOR I=1 TO 31
190 IF MIDS(M$(J),I,1) <> "." PROC CUBE(I*10-10,J*10 )
200 NEXT I
210 NEXT J
220 '

```


LASER BASIC FOR THE COMMODORE 64

position on the hires screen, using PROCcube. The loop starts at the bottom of the screen, so that the cubes appear in the correct perspective. Once the whole display has been covered, the computer goes into an infinite loop in lines 230 and 240. You should press the RUN/STOP key to leave to program.

The procedure "cube" at line 260 draws a 10x10x10 cube on the screen. It has two parameters. These are the x and y co-ordinates of the cube's front top left hand corner. c1 and c2 are simply constants which are used in the procedure. Declaring them as variables, rather than using the number each time, speeds up execution. In lines 320 to 350, the top of the cube is drawn in blue. It is formed by drawing five lines next to one another. These merge to look like a flat surface. The light blue right side of the cube is drawn in lines 370 to 400 in a similar way. The BOX command in line 430 draws the front surface in white.



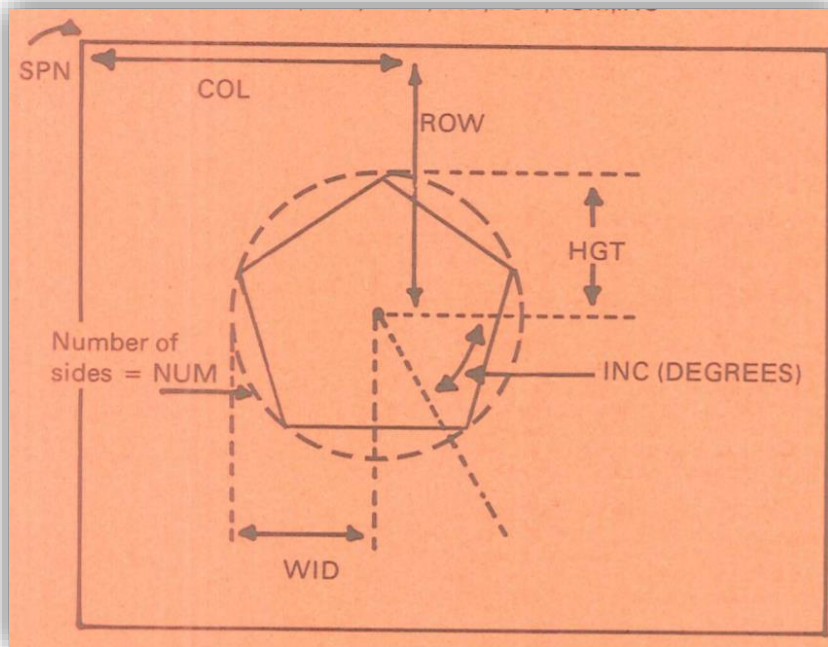
DRAWING CIRCLES AND POLYGONS

(POLY)

The POLY command draws a circle or polygon inside a sprite. It has seven parameters:

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

LASER BASIC FOR THE COMMODORE 64



The sprite used is sprite number SPN. COL and ROW are the column and row of the polygon's centre. WID is the horizontal radius of the polygon, i.e. WID is half the polygon's total width. Similarly, HGT is its vertical radius and is half the total height. INC is the inclination of the polygon in degrees.

NUM is the number of sides. INC and NUM must both be less than 256. If NUM is large, or less than three, a circle is drawn (or an ellipse if WID and HGT are not equal). As with the other hires graphics commands, all distances are measured in pixels. The scaling is the same for both hires modes. The colour that the polygon is drawn in depends on the colour value set up by the MODE command.

If the polygon does not fit into the sprite, an error message is not printed. Instead, any parts of the polygon which do not lie within the sprite are omitted.

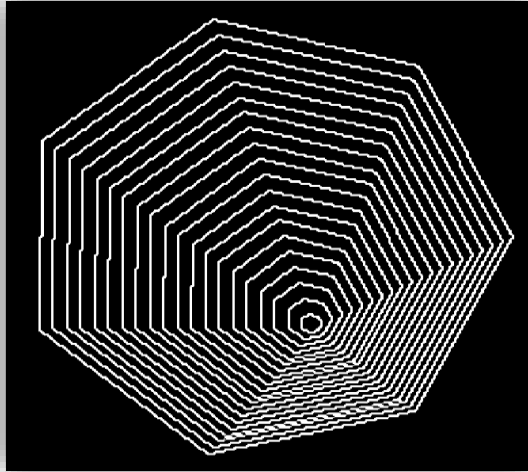
Example: The following program draws a series of concentric heptagons on the hires screen.

```
10 INIT
20 '
30 FGND WHITE
40 BGND BLACK
50 HBORDER BLACK
60 ATTON
70 SCLR 0,ATR
80 MODE 3
90 HIRES
100 FOR I=1 TO 20
110 POLY 0, 180-I, 140-2*I, I*5, I*5, 7, 0
120 NEXT I
130 REPEAT
140 UNTIL FALSE
```

LASER BASIC FOR THE COMMODORE 64

Available on disk as "ex drawing 01"

Lines 10 to 90 set up and clear the hires screen. The background is brown and the foreground is white. Lines 100 to 120 form the loop that places the heptagons on the screen. COL and ROW do not remain constant. As the size of the heptagon increases, the centre moves up towards the top left of the screen. The RUN/STOP key must be pressed to exit from the program.



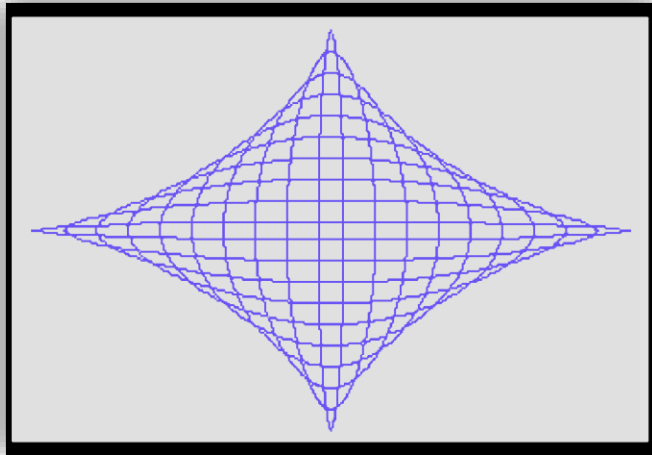
Example: This program draws a pattern on the screen made up of ellipses

```
10 INIT
20 `
30 FGND BLUE
40 BGND GRAY3
50 HBORDER GRAY3
60 ATTON
70 MODE 3
80 SCLR 0,ATR
90 HIRES
100 FOR I=5 TO 95 STEP 10
110 POLY 0,160,100,I * 1.6,100 - I,0,0
120 NEXT I
130 REPEAT
140 UNTIL FALSE
```

Available on disk as "ex drawing 02"

As in the previous example, lines 10 to 90 set up the hires screen. This time the foreground is blue and the background is light grey. The loop in lines 100 to 120 draws the ellipses on the screen. They all have the same centre, at (160,100). As the width is increased, the height is decreased. The width is multiplied by 1.6 so that the figure fills the whole screen.

LASER BASIC FOR THE COMMODORE 64



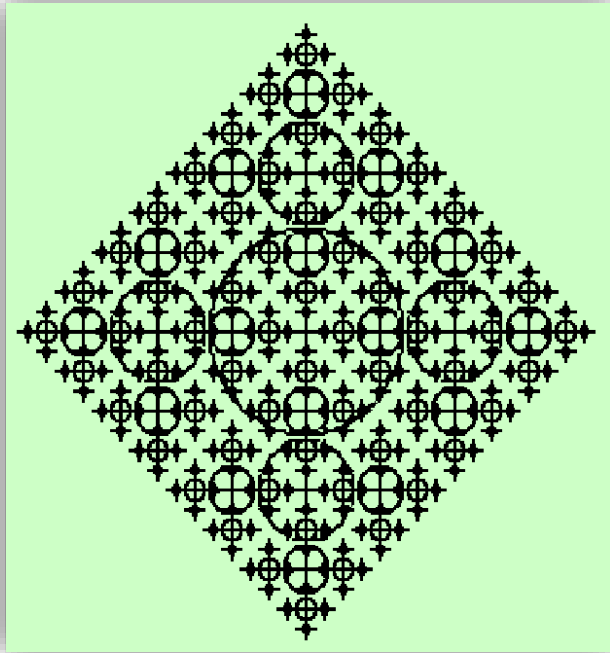
Example: This program draws a pattern on the screen which is made up of lines and circles. If you RUN it, you will see that each circle making up the pattern is itself surrounded by four smaller circles which are each surrounded by four even smaller circles which are etc.

```
10 INIT
20 FGND BLACK
30 BGND .GREEN
40 SCLR 0,ATR
50 MONO
60 HBORDER .GREEN
70 HIRES
80 MODE3
90 '
100 PROC QUAD(100,160,48)
110 '
120 REPEAT
130 UNTIL THE COWS COME HOME
140 '
150 LABEL QUAD(QRW,QCL,QSZ)
160 CIF QSZ > 2
170 POLY 0,QCL,QRW,QSZ/1.5,QSZ/1.5,0,0
180 PROC QUAD(QRW+QSZ,QCL,QSZ/2)
190 PROC QUAD(QRW,QCL+QSZ,QSZ/2)
200 PROC QUAD(QRW-QSZ,QCL,QSZ/2)
210 PROC QUAD(QRW,QCL-QSZ,QSZ/2)
220 DRAW 0,QCL,QRW-QSZ,QCL,QRW+QSZ
230 DRAW 0,QCL-QSZ,QRW,QCL+QSZ,QRW
240 CEND
250 PROCEND
```

Available on disk as "ex drawing 03"

LASER BASIC FOR THE COMMODORE 64

This program is a good example of the use of recursion. The procedure "quad" starting at line 150 is recursive because it calls itself. Quad draws the pattern at position (qcl, qrw). Qsz is the size of the pattern. The procedure first draws a circle using the POLY command then draws four smaller copies of the pattern around the circumference of the circle by calling itself four times. Lastly, two lines are drawn at right-angles, using DRAW, to complete the pattern. The CIF-CEND in lines 160 and 240 are very important; the pattern is only drawn if it is greater than two pixels in size. If this were not the case, the recursion would continue indefinitely. The pattern would never be completed, and the computer would run out of memory (or the cows stop giving milk!).



FILLING IN BLANK AREAS (FILL)

The FILL command sets all the pixels around a point. It stops setting pixels when a line or the edge of the sprite is reached. The command has four parameters:

FILL SPN, COL, ROW, NUM

As usual, SPN is the sprite which is used. COL and ROW specify the point around which FILLing begins. The scaling of COL is the same for two colour and four-colour modes.

USING FILL IN TWO-COLOUR MODE

In two-colour mode, NUM is not used and therefore its value does not matter. The colour used when filling is either foreground or background, and this depends on the colour value established by the MODE command. FILL will only work correctly if this value is 0, 1, 2 or 3.

The area being filled can be as complex as you like. Here is an example, using FILL in two-colour mode. The program draws a shape on the screen and then fills it in.

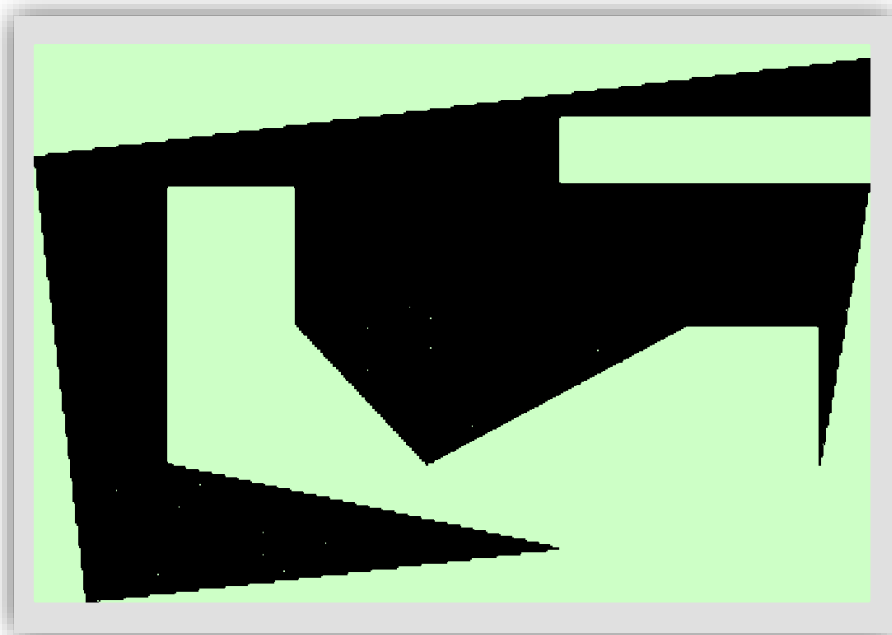
```
50 HBORDER GRAY3  
60 ATTON
```


LASER BASIC FOR THE COMMODORE 64

```
70 MODE 3
80 SCLR 0,ATR
90 HIRES
100 READ X1,Y1
110 REPEAT
120 READ X
130 IF X=-1 THEN EXIT
140 READ Y
150 DRAW 0,X1,Y1,X,Y
160 X1=X: Y1=Y
170 UNTIL FALSE
180 FILL 0,120,100,0
190 REPEAT
2@@ UNTIL FALSE
210 DATA 100,100,150,150,250,100
220 DATA 300,100,300,150,319, 50
230 DATA 200, 50,200, 25,319, 25
240 DATA 319, 5, 0, 40, 20,199
250 DATA 200,180, 50,150, 50, 50
260 DATA 100, 50,100,100, -1, -1
```

Available on disk as "ex fill 01"

Lines 10 to 90 set up the hires screen in the normal way. The background is light grey and the foreground is blue. In lines 100 to 170, the shape is drawn using DRAW. The list of points used is contained in the data statements at line 210 onwards. The FILL command is in line 180. Here, filling is done with the foreground colour. It is also possible to use the background colour. In this case, filling stops when a border of background, rather than foreground, colour is reached.



The next example uses both foreground and background filling:

LASER BASIC FOR THE COMMODORE 64

```
10 INIT : HBORDER BLACK
20 BGND BLACK
30 FOR X=0 TO 39 STEP 5
40   FOR Y=0 TO 24 STEP 5
50     FGND 1+RND(1)*15
60     WCLR 0,X,Y,5,5,ATR
70     NEXT Y
80   NEXT X
90   `
100  MODE 4 : HIRES
110  FOR I=0 TO 240 STEP 80
120   IF I <> 240 BOX 0,0,1,320,40
130   BOX 0,1,0,40,208
140  NEXT I
150  `
160  FOR I=0 TO 7
170   FOR J=0 TO 4
180    IF I+J AND 1 THEN MODE 1 ELSE MODE 3
190    POLY 0, 20+40*1, 20+40*J, 18, 18,J+2, 1*360 / (J +2) / 7
200    FILL
210   NEXT J
220  NEXT I
230  `
240  REPEAT
250  UNTIL FALSE
```

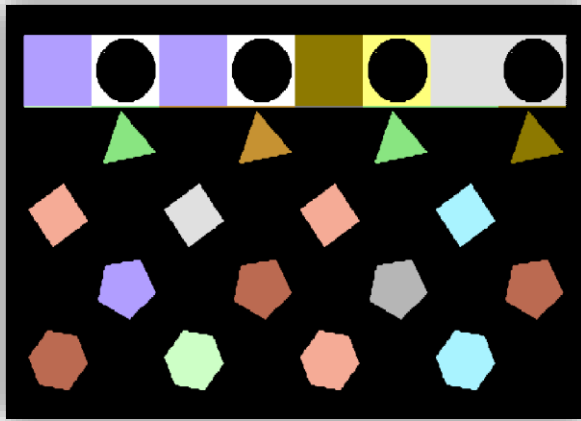
Available on disk as "ex fill 02"

Lines 10 to 80 set up the screen. The foreground in each 5 x 5 character block is set to a different colour in lines 50 and 60. The background is always black. The loop from lines 110 to 140 draws a pattern on the screen, like part of a large chessboard. Each square is 40x40 pixels. The double loop from 160 to 220 draws polygons on the screen with varying inclinations and numbers of sides. One is drawn in each square. Line 180 checks to see if the current square is foreground or background. If i is foreground, the polygon will be drawn in the background colour (MODE 1) and vice versa. The FILL command in line 200 does not need any parameters because SPN, COL and ROW have already been set up by the POLY command in the previous line.

N.B.

After using the FILL command, the sprite variables CCOL and CROW will be set to random values. This should not cause any problems, because CCOL and CROW are only used for sprite collision detection which is covered later in [COLLISION DETECTION \(DTCTON, DTCTOFF\)](#).

LASER BASIC FOR THE COMMODORE 64



When a FILL is taking place, BASIC's free memory space is used to store temporary data. You can print out the number of free bytes using:

```
print fre(i)
```

If your program uses strings a lot, it would be a good idea to force a garbage collection before using FILL. This will ensure that the maximum amount of space is available, reducing the probability of an "out of memory" error. A garbage collection can be forced like this:

```
i=fre(i)
```

USING FILL IN FOUR-COLOUR MODE

Because of the larger number of colours, FILL is more difficult to use in four-colour mode. As in two-colour mode, the colour used when filling is selected by using the MODE command. The colour value set must be 0,1, 2 or 3.

However, there are fourteen different combinations of colours which can act as the border for the FILL. These depend on the value of NUM:

NUM	Border Colours
1	0
2	1
3	0,1
4	2
5	0,2
6	1,2
7	0,1,2
8	3
9	0,3
10	1,3
11	0,1,3
12	2,3
13	0,2,3
14	1,2,3

LASER BASIC FOR THE COMMODORE 64

The actual plotting colour selected by MODE is always a border colour, irrespective of the value of NUM.

Here are a few examples to illustrate this. In all cases, filling is taking place at (100,100) in sprite 3.

Fill with colour 2. Colours 1, 2 and 3 are borders for the fill:

```
MODE 2:FILL 3,100,100,14
```

Fill with colour 1. Only colour 1 is a border for the fill:

```
MODE 1:FILL 3,100,100,1
```

Fill with colour 0. Colours 0 and 2 are borders for the fill:

```
MODE 0:FILL 3,100,100,5
```

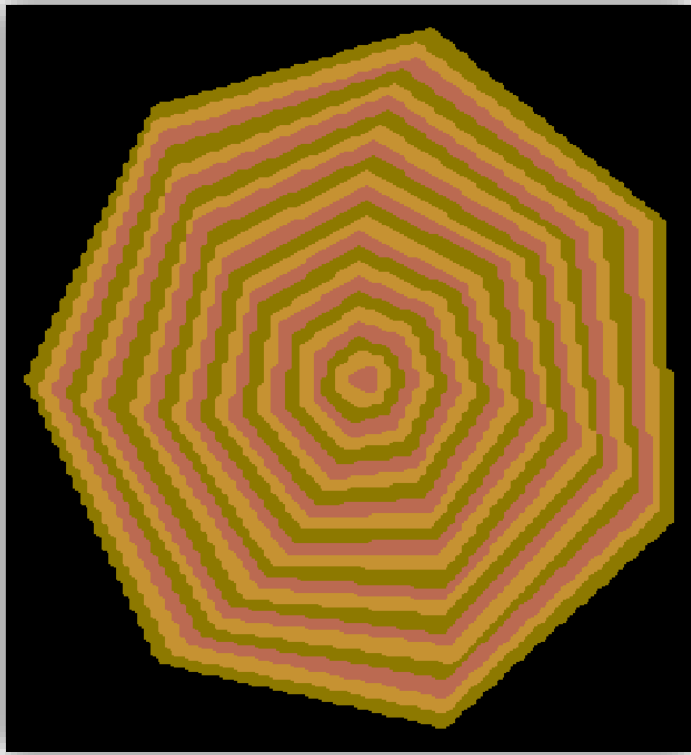
Example: This program uses POLY and FILL to generate a series of concentric heptagons. By using the SETA command to change the colours quickly, the program gives a "tunnel" effect:

```
10 INIT:MULTI:S4COL:HIRES:ATT20N
20 MCOL1 BROWN: MCOL2 RED
30 MCOL3 ORANGE:SCLR 0,ATR
40 HBORDER BLACK:HPAPER BLACK
50 FOR R=24 TO 1 STEP-1
60 MODE R-3 * INT(R/3) + 1
70 POLY 0,160,100,R * 4,R * 4,7,R
80 FILL 0,160,100,0
90 NEXT R
100 A=BROWN : B=RED : C=ORANGE
110 REPEAT
120 MCOL1 A : MCOL2 B : MCOL3 C
130 SETA 0,0,0,40,25,ATR
140 T=C: C=B : B=A : A=T
150 UNTIL FALSE
```

Available on disk as "ex fill 03"

The first four lines of the program set up the screen in four-colour mode. Lines 50 to 90 draw and fill the heptagons. Line 60 re-sets the drawing colour so that each heptagon is a different colour. In lines 110 to 150, the variables a, b and c are used to hold the three colours, which are swapped over in line 140 after each SETA.

LASER BASIC FOR THE COMMODORE 64



EXAMINING INDIVIDUAL PIXELS

(POINT)

POINT is a function which you can use to examine individual pixels within a sprite. It takes parameters which must be put within brackets:

POINT (SPN, COL, ROW)

As usual, SPN is the sprite to be examined, and COL and ROW are the co-ordinates of the point, measured in pixels. POINT does not alter the pixel.

If Laser BASIC is in two-colour mode (i.e. the S2COL command has been executed), POINT returns a value of 0 or 1, corresponding to the point being cleared or set respectively. In four-colour mode, it returns 0, 1, 2 or 3 corresponding to one of the four colours. Remember that the background is treated as colour 0.

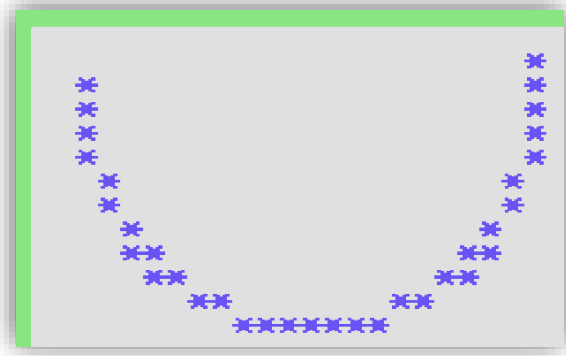
Example: This program draws a circle inside a sprite, and then reads it out of the sprite and prints it on the text screen.

```
10 INIT
20 RESET
30 SPRITE 1,3,3
40 POLY 1,1,12,11,11,0,0
50 `
60 PRINT
70 FOR i=0 TO 23
```

LASER BASIC FOR THE COMMODORE 64

```
80 FOR J=0 TO 23
90 IF POINT(1,I,J) = 1 PRINT "*"; ELSE PRINT " ";
100 NEXT J
110 PRINT
120 NEXT I
```

Available on disk as "ex pixel 01"



SPRITE 'BLOCK MOVE' COMMANDS

So far, you have learned how to set up sprites of any dimension, set up the display and draw shapes within any sprite. However, a sprite is no use unless there is some way of copying it onto the screen, and the 'block move' commands have been provided to do just that. (Remember that the screen is treated as sprites 0 and 255 in bit-mapped and text modes respectively)

Most of the examples in this section use the arcade sprite library. Therefore, before proceeding, load the sprites in from tape or disk, using

```
RECALL "SPRITESA"
```

or `DRECALL "SPRITESA"` for disk

.

DISPLAYING A SPRITE ON THE SCREEN

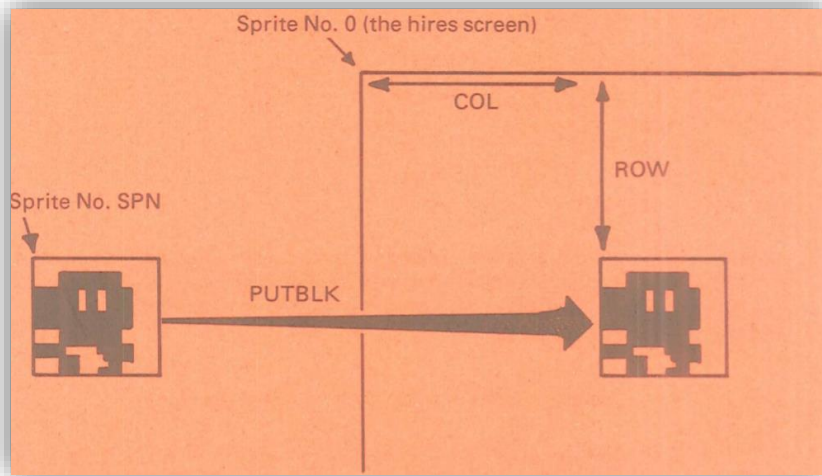
(PUTBLK)

To display an entire sprite on the hires screen, you can use the PUTBLK command. Its parameters are:

```
PUTBLK SPN, COL, ROW
```

SPN is the sprite which is to be displayed, and COL and ROW are the co-ordinates, measured in character blocks, of the top left hand corner of the sprite's position on the screen:

LASER BASIC FOR THE COMMODORE 64



Obviously, the sprite overwrites what was on the screen at COL, ROW previously. Also, since PUTBLK copies the sprite, any subsequent changes to the sprite, such as inverting it or even deleting it, do not affect the copy on the screen. PUTBLK can only place sprites on the screen to a resolution of one character; to place sprites on the screen at pixel resolution, you have to use the scrolling commands which will be introduced later.

Example 1: Place a sprite no. 38, PAC-MAN, on the hires screen at (6, 2)

```
10 INIT
20 FGND BLACK : BGND WHITE : SCLR 0,ATR
30 HIRES
40 PUTBLK 38,6,2
50 GOTO 50
```

Available on disk as "ex putblk 01"



Remember that the arcade sprite library. "SPRITESA" must be loaded to run this program.

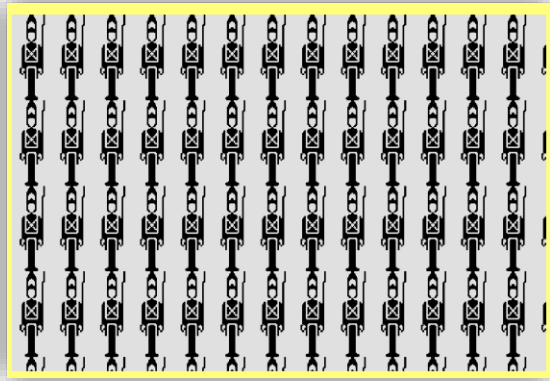
Example 2: Completely fill the screen with sprite no. 45:

```
10 INIT : SCLR0,0
20 HIRES
30 FOR i=0 TO 39 STEP 3
40   FOR J=0 TO 24 STEP 6
50     PUTBLK 45,I,J
60   NEXT J
70 NEXT I
80 GOTO 80
```

LASER BASIC FOR THE COMMODORE 64

Available on disk as "ex putblk 01"

If you run this program, you will notice that the sprites along the bottom and righthand side of the screen have been 'clipped' by Laser BASIC, so that they fit onto the screen. This 'clipping' is carried out automatically in all the block-move commands.



CONTROLLING ATTRIBUTES

(ATTOFF, ATTON, ATT20N)

Normally, Laser BASIC copies the primary attributes as well as the pixel data when a PUTBLK or any other block-move command is executed. This is obvious if you run Example 1 above: if the attributes were not copied with the sprite, the sprite would have been displayed in black and white.

If, for some reason, you don't want the attributes to be copied, use the ATTOFF command. After ATTOFF, any subsequent block-move commands will not copy the attributes. To see this working, try typing in Example 1 above and add:

```
35 ATTOFF
```

Upon running the program, the sprite is displayed in black and white because of the attributes placed there by the SCLR in line 20.



ATTON is the opposite of ATTOFF - it turns the movement of attributes back on again.

If you are working in four-colour mode, you will probably want both sets of attributes to be copied. This is done by using the ATT20N command.

LOGICAL OPERATIONS

(PUTOR, PUTAND, PUTXOR)

LASER BASIC FOR THE COMMODORE 64

The PUTBLK command covered earlier simply places the sprite on the screen, overwriting what was there before. However, there are many situations in which it is desirable to combine the sprite pixel data with the screen in some way, and this is the purpose of PUTOR, PUTAND and PUTXOR

These three commands each have the same parameters as PUTBLK, i.e. SPN, COL and ROW. With PUTOR, each pixel on the screen is set if the pixel on the screen OR the corresponding pixel in the sprite is already set. Similarly, with PUTAND, a screen pixel is set if both the screen pixel AND the sprite pixel are set. PUTXOR carries out an 'exclusive-or' instead of the normal 'inclusive-or' in PUTOR - with PUTXOR, the screen pixel is set if either BUT NOT BOTH the sprite pixel and screen pixel are already set.

This is summarised below.

SPRITE PIXEL BEFORE	SCREEN PIXEL BEFORE	SCREEN PIXEL AFTER PUTOR	SCREEN PIXEL AFTER PUTAND	SCREEN PIXEL AFTER PUTXOR
cleared	cleared	cleared	cleared	Cleared
cleared	set	set	cleared	Set
set	cleared	set	cleared	Set
set	set	set	set	Cleared

PUTXOR has one special property. If you put a sprite on to the screen using PUTXOR, it can be removed again by using another PUTXOR command. The screen is left as it was before.

With all these commands, the attributes will always overwrite the screen's attributes, regardless of whether PUTOR, PUTAND or PUTXOR are being used.

To clarify this, here is a program which demonstrates the difference between PUTBLK, PUTOR, PUTAND and PUTXOR:

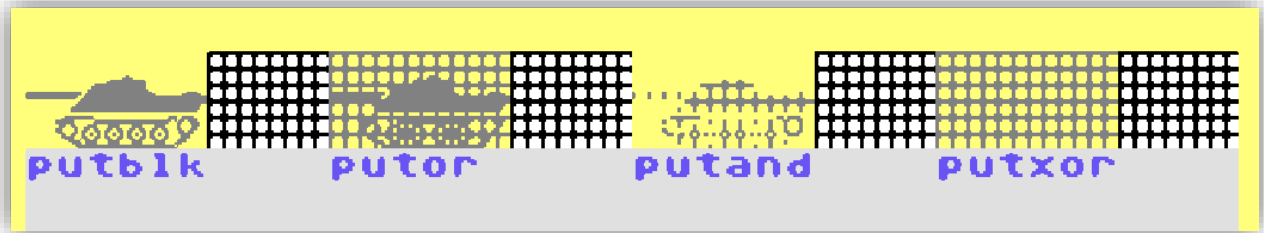
```
10 INIT
20 SCLR 0,1
30 WINDOW 3
40 FOR I=0 TO 23 STEP 4
50 DRAW 0,0,I,319,I
60 NEXT I
70 FOR I=0 TO 319 STEP 4
80 DRAW 0,I,0,I,23
90 NEXT I
100 PRINT CHR$(147); "PUTBLK", "PUTOR", "PUTAND", "PUTXOR"
110 PUTBLK 32,0,0
120 PUTOR 32,10,0
130 PUTAND 32,20,0
140 REPEAT
150 PUTXOR 32,30,0
160 REPEAT : GET A$ : UNTIL A$ <> " "
170 UNTIL FALSE
```

Available on disk as "ex putblk 03"

Assuming that you will have the arcade sprite library loaded into the computer, this program will draw a grid on the top three lines of the screen, and place sprite no. 32 on the screen four times, using PUTBLK,

LASER BASIC FOR THE COMMODORE 64

PUTOR and PUTXOR respectively. The loop in lines 140 to 170 puts the sprite on the screen using PUTXOR each time you press a key. Each time you press the space bar, you will see the tank alternately appearing and disappearing, as it is exclusive-ORed with what is already on the screen.



Re-read the previous paragraphs carefully if it is not clear how the various effects on the screen are generated.

If you are working in four-colour mode, the colours produced are rather more complex:

SPRITE COLOUR BEFORE	SCREEN COLOUR BEFORE	SCREEN COLOUR AFTER PUTOR	SCREEN COLOUR AFTER PUTAND	SCREEN COLOUR AFTER PUTXOR
0	0	0	0	0
0	1	1	0	1
0	2	2	0	2
0	3	3	0	3
1	0	1	1	0
1	1	1	1	0
1	2	3	0	3
1	3	3	1	2
2	0	2	0	2
2	1	3	0	3
2	2	2	2	0
2	3	3	2	1
3	0	3	0	1
3	1	3	1	2
3	2	3	2	0
3	3	3	3	1

Note that with PUTOR, PUTAND and PUTXOR, the attributes are always copied to the screen unaltered.

OTHER BLOCK-MOVE COMMANDS

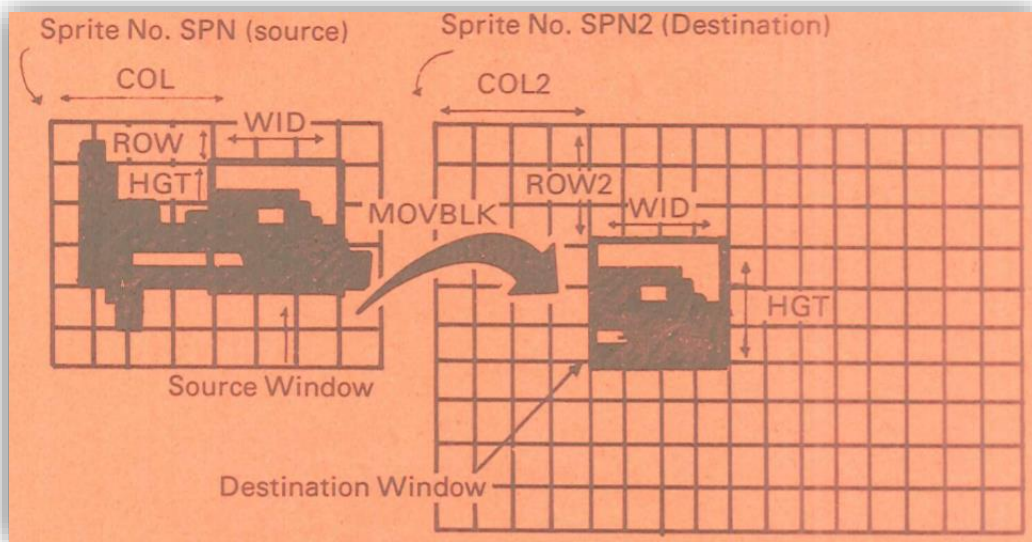
(MOVBLK, MOVOR, MOVAND, MOVXOR, GETBLK, GETOR, GETAND, GETXOR, CPYBLK, CPYOR, CPYAND, CPYXOR)

LASER BASIC FOR THE COMMODORE 64

For some applications, the PUT commands (PUTBLK, PUTOR, PUTAND and PUTXOR) are not flexible enough. You might want to display only part of a sprite, or perhaps copy part of one sprite into another. In this case, you could use one of the four MOV commands - MOVBLK, MOVOR, MOVAND and MOVXOR. All have eight parameters:

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

SPN is the source sprite, and COL and ROW are the co-ordinates of the top left hand corner of the source window, measured in character blocks. WID and HGT are the dimensions of the window to be moved. SPN2 is the destination sprite, and COL2 and ROW2 are the co-ordinates of the top left hand corner of the destination window.



Thus, PUTBLK is simply a special case of MOVBLK, with the source window equal to the whole source sprite, and the destination sprite the screen.

The other three MOV commands - MOVOR, MOVAND and MOVXOR all operate in an exactly analogous way to PUTOR, PUTAND and PUTXOR

There are two other sets of commands which are used less often Like PUT they are special cases of MOV:

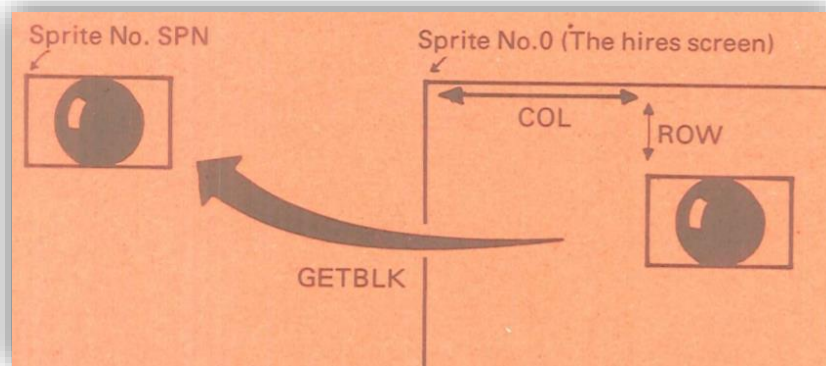
- I. GETBLK, GETOR, GET AND, GETXOR

These four commands all have three parameters.

GETBLK SPN, COL, ROW

GET can be thought of as the opposite of PUT - the sprite no SPN is copied back from position COL.ROW on the screen:

LASER BASIC FOR THE COMMODORE 64

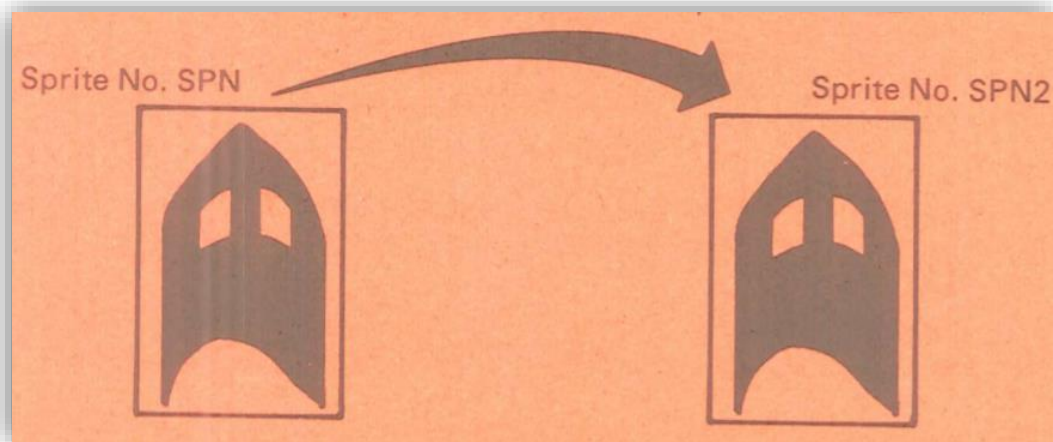


II. CPYBLK, CPYAND, CPYOR, CPYXOR

These commands have two parameters:

CPYBLK SPN, SPN2

CPY copies sprite no. SPN into sprite no.



SUMMARY

The block-move commands encountered so far make up the sixteen one-way move commands:

MOVBLK	MOVOR	MOVAND	MOVXOR
GETBLK	GETOR	GETAND	GETXOR
PUTBLK	PUTOR	PUTAND	PUTXOR
CPYBLK	CPYOR	CPYAND	CPYXOR

The prefixes and suffixes have the following meanings:

Prefix:

Prefix	Parameters	Description
--------	------------	-------------

LASER BASIC FOR THE COMMODORE 64

MOV	SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2	A window of width WID and height HGT in sprite SPN whose top left hand corner is at COL,ROW is MOVED into sprite SPN2 at COL2 and ROW2.
PUT	SPN, COL, ROW	Sprite no. SPN is PUT onto the screen with its top left hand corner at COL, ROW.
GET	SPN, COL, ROW	GETs sprite no. SPN from the screen at position COL, ROW.
CPY	SPN, SPN2	COPYs sprite number SPN into sprite no SPN2.

SUFFIXES:

Suffix	Description
BLK	the source pixels overwrite their destination.
OR	the source pixels are ORed with their destination
AND	the source pixels are ANDed with their destination
XOR	the source pixels are exclusive-ORed with their destination.

SOME SIMPLE PROGRAMMING EXAMPLES

Please remember that in order to run any of these examples, the arcade sprite library must be loaded into the computer. You can do this by using

RECALL "SPRITESA" for tape,
or **DRECALL "SPRITESA"** for disk.

Example 1: This program sets up the screen with a border of multicoloured teddy bears (sprite no. 43).

```

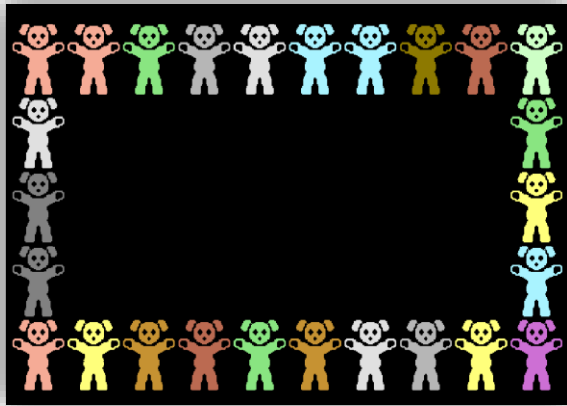
10 INIT : SCLR 0,0 : HBORDER BLACK
20 HIRES : ATTOFF
30 FOR I=0 TO 36 STEP 4
40  PROC PUT(I,0)
50  PROC PUT(I,20)
60 NEXT I
70 FOR I=5 TO 15 STEP 5
80  PROC PUT(0,I)
90  PROC PUT(36,I)
100 NEXT I
110 REPEAT
120 UNTIL FALSE
130 '
140 LABEL PUT(X,Y)
150 PUTBLK 43,X,Y
160 BGND BLACK
170 FGND RND(1) * 15 + 1
180 SETA 0,X,Y,4,5,ATR
190 PROCEND

```

Available on disk as "ex sprite 01"

LASER BASIC FOR THE COMMODORE 64

The first two lines of the program simply set up the hires screen. Note that there is an ATTOFF in line 20. The procedure PROC put in line 140 is used to place the sprite at any position on the screen. Since ATTOFF has been used, the attributes are not copied. Instead, the SETA command is used in line 180 to set the bears' colour to a random value.



Example 2: This program illustrates the use of MOVBLK to create a large sprite from a smaller one.

```
10 RESERVE 9000
20 IF DFA(51) < 0 THEN SPRITE 51,45,3
30 INIT : WINDOW 3
40 FOR I=0 TO 45 STEP 6
50  MOVBLK 34,0,0,6,3,51,I,0
60  NEXT I
70  REPEAT
80    FOR I=0 TO -5 STEP -1
90      PUTBLK 51,I,0
100     FOR J=0 TO 30 : NEXT J
110    NEXT I
120  UNTIL FALSE
```

Available on disk as "ex sprite 02"

Lines 10 to 20 set up sprite 51 to be 45 by 3 character 1 locks. A RESERVE must be used to give more space for sprites, because there is not sufficient room for a sprite of this size if the arcade sprites are loaded in.

Also, note that DFA is used in line 20 to check if sprite 51 already exists before defining it.

The loop in line 40 copies sprite no. 34, the crocodile, into sprite 51, eight times. Lines 70 to 120 scroll the sprite along the top of the screen. The column used to put the sprite on the screen in line 90 is actually negative -because of Laser BASIC's automatic clipping, only part of the sprite appears on the screen.

LASER BASIC FOR THE COMMODORE 64



```
break in 100
ready.
list

10 RESERVE 9000
20 IF DFA(51) < 0 THEN SPRITE 51,45,3

30 INIT : WINDOW 3
40 FOR i=0 TO 45 STEP 6
50 MOVBLK 34,0,0,6,3,51,i,0
60 NEXT i
70 REPEAT
80 FOR i=0 TO -5 STEP -1
90 PUTBLK 51,i,0
100 FOR j=0 TO 30 : NEXT j
110 NEXT i
120 UNTIL FALSE
130 END
ready.
■
```

Example 3: This program shows how to fill the hires screen (or any sprite) very quickly with a pattern, and also how to get more than 16 colours on the screen

```
10 INIT : SCLR 0,1 : HIRES
20 MODE 4
30 FOR I=1 TO 6 STEP 2
40 DRAW 0,I,0,I,7
50 DRAW 0,0,I,7,I
60 NEXT I
70 MOVBLK 0,0,0,39, 1,0,1,0
80 MOVBLK 0,0,0,40,24,0,0,1
90 FOR i=0 TO 38 STEP 2
100 FOR J=0 TO 22 STEP 2
110 SETA 0,I,J,2,2,RND(1) * 256
120 NEXT J,I
130 HBORDER BLACK: SETA 0,0,24,40,1,0
140 REPEAT
150 UNTIL FALSE
```

Available on disk as "ex sprite 03"

Lines 20 to 60 of this program draw a very fine check pattern in the top left hand corner of the hires screen. Lines 70 and 80 then copy the pattern over the entire screen. When the MOVBLK command copies a rectangular sprite window from one place to another, it copies the top left hand corner first, and proceeds in the same order as reading a book. In line 70 MOVBLK copies a window of width 39 characters and height 1 character to a position one character block to the right of its original position. The source and destination windows overlap. So, the first character copied is at the left of the source window. When Laser BASIC copies the second character, it has already been set to the check pattern because the second character of the source window is the first character of the destination window.

LASER BASIC FOR THE COMMODORE 64

Continuing in this way, the whole top line of the screen is filled with the check pattern. In a similar way, line 80 copies the pattern from the top line to the entire screen.

Lines 90 to 120 set the attributes in each 2x2 character block on the screen to a random value. Since the check pattern is so fine, the colours appear to merge into one another, giving the illusion of more than 16 colours on the screen at once.



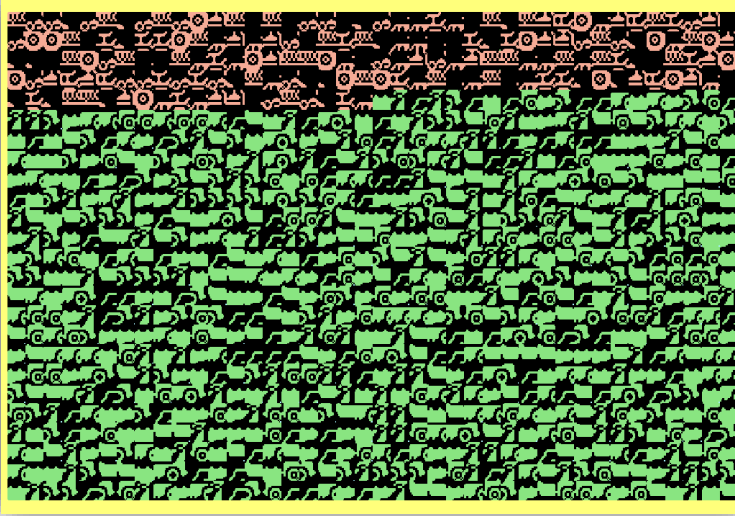
Example 4: The following program takes random 1 x 1 character windows out of sprites 3 to 7 and places them on the screen:

```
10 INIT
20 HIRES
30 FOR S=1 TO 3
40   FOR V=0 TO 24
50     FOR X=0 TO 39
60       MOVBLK S,RND(1)*4,RND(1)*2,1,1,0,X,V
70     NEXT X
80   NEXT V
90 NEXT S
100 END
```

Available on disk as "ex sprite 04"

S is the number of the source sprite; x and y are the column and row positions in sprite 0, the screen.

LASER BASIC FOR THE COMMODORE 64



Example 5: This program shows how PLOT can be used with sprites other than the screen. It 'fills up' a sprite (no. 200) with random PLOTs, putting the sprite on the screen after each PLOT.

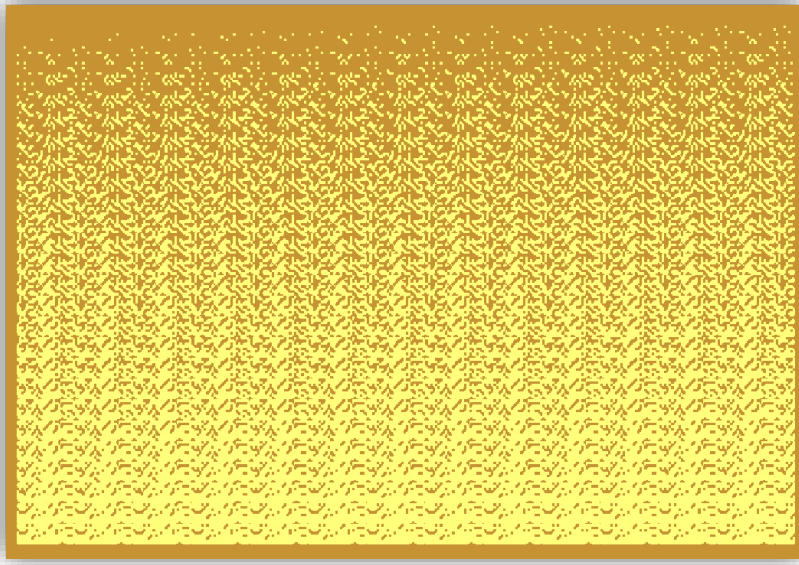
```
10 INIT: IF DFA(200) > 0 WIPE
20 SPRITE 200,3,1
30 BGND ORANGE : FGND YELLOW
40 SCLR 0,ATR : HIRES: HBORDER ORANGE
50 SCLR 200,ATR
60 FOR Y=0 TO 24
70   FOR X=0 TO 39 STEP 3
80     PLOT 200,RND(1)*24,RND(1)*8
90     PUTBLK 200,X,Y
100  NEXT X
110  NEXT Y
110  WIPE 200
```

Available on disk as "ex sprite 05"

Lines 10 and 20 initialise the system and create sprite no. 200. Lines 30, 40 and 50 set up the hires screen, and clear sprite 200. In lines 60 to 100, x and y are the current column and row positions on the screen. Line 80 plots the random point inside sprite 200, while line 90 puts it on the screen.

Sprite 200 is erased before exiting the program in line 110.

LASER BASIC FOR THE COMMODORE 64



TWO-WAY BLOCK MOVE COMMANDS

(XXX%XXX)

The block-move commands you have looked at so far are one-way move commands. That is, the sprite data is always moved from source to destination. Laser BASIC also has a set of very powerful two-way move commands which allow you to exchange data between the two sprite windows

There are four possible logical operations (BLK, OR, AND and XOR), each of which May be applied to either sprite window. This gives 4 x 4 = 16 commands

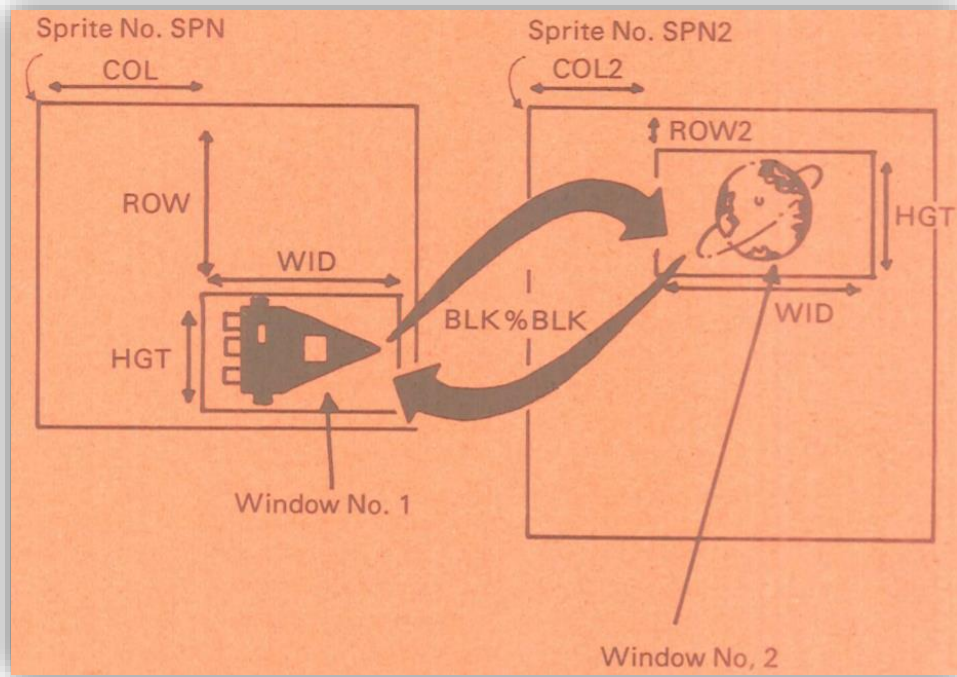
BLK%BLK	OR%BLK	XOR%BLK	AND%BLK
BLK%OR	OR%OR	XOR%OR	AND%OR
BLK%XO	OR%XOR	XOR%XOR	AND%XOR
BLK%AND	OR%AND	XOR%AND	AND%AND

All sixteen commands use the same parameters:

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

Sprite window no. 1 is held inside sprite no. SPN, at position COL, ROW, WID and HGT are the size of the window. Similarly, sprite no. 2 is inside sprite no. SPN2, at position COL2, ROW2. Obviously, window no. 2 is the same size as window no. 1. The pixel and attribute data is exchanged between the two windows. The command names themselves consist of two logical operations (i.e. BLK, OR, XOR and AND) separated by a '%' sign. This first of these is the logical operation used for all the data coming from window 2 and going into window 1. Similarly, the second logical operation is for the data coming from window 1 and going into window 2

LASER BASIC FOR THE COMMODORE 64



Laser BASIC exchanges the data between the two windows simultaneously. Also, ATTOFF, ATTON and ATT2ON can be used to control the use of attributes, as with the one-way block move commands.

To get a clearer idea of how this set of commands works, type in and run the following program, with the arcade sprites loaded into memory:

```

10 INIT : FGND GREEN : BGND WHITE : WINDOW 8
20 FOR N=0 TO 15
30 IF N : REPEAT: GET A$ : UNTIL A$ <> ""
40 PRINT CHR$(147)
50 SCLR 0,ATR : WINDOW 8
60 FOR I=0 TO 48 STEP 16
70   FOR J=0 TO 48 STEP 16
80     IF I+J AND 16 BOX 0,I,J,16,16
90   NEXT J,I
100 FOR I=0 TO 6 STEP 2
110   PUTBLK 40,8,I : NEXT I
120 FOR J=0 TO 400 : NEXT J
130 SPN=0 : COL=0 : ROW=0 : WID=8 : HGT=8
140 SPN2=0 : COL2=8 : ROW2=0
150 CASE N
160   OF 0 : BLK%BLK : PRINT "BLK%BLK"
170   OF 1 : BLK%OR  : PRINT "BLK%OR"
180   OF 2 : BLK%XOR : PRINT "BLK%XOR"
190   OF 3 : BLK%AND : PRINT "BLK%AND"
200   OF 4 : OR%BLK  : PRINT "OR%BLK"
210   OF 5 : OR%OR   : PRINT "OR%OR"

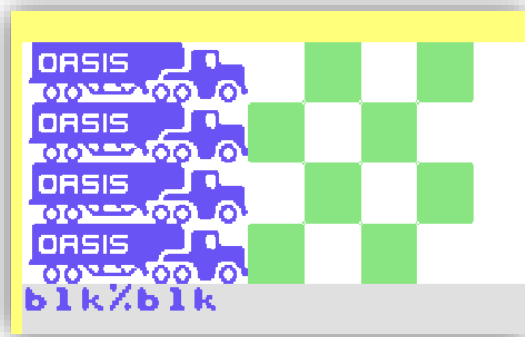
```

LASER BASIC FOR THE COMMODORE 64

```
220 OF 6 : OR%XOR : PRINT "OR%XOR"  
230 OF 7 : OR%AND : PRINT "OR%AND"  
240 OF 8 : XOR%BLK : PRINT "XOR%BLK"  
250 OF 9 : XOR%OR : PRINT "XOR%OR"  
260 OF 10 : XOR%XOR : PRINT "XOR%XOR"  
270 OF 11 : XOR%AND : PRINT "XOR%AND"  
280 OF 12 : AND%BLK : PRINT "AND%BLK"  
290 OF 13 : AND%OR : PRINT "AND%OR"  
300 OF 14 : AND%XOR : PRINT "AND%XOR"  
310 OF 15 : AND%AND : PRINT "AND%AND"  
320 CASEND  
330 NEXT N
```

Available on disk as "ex sprite 06"

Lines 50 to 120 draw a check pattern at the top left of the screen, and place four lorries sprite no. 40) next to it. Each time you press a key, the computer will carry out one of the 16-way moves, the check pattern being window no. 1, and the lorries are window no. 2.



COLLISION DETECTION

(DTCTON, DTCTOFF)

After putting a sprite on the screen, or moving pixel data between sprites, it is often necessary to know if there has been a collision between the source and destination pixel data. (A collision occurs if a pixel in the source sprite and the corresponding pixel in the destination sprite are both set).

Laser BASIC provides automatic collision detection as part of the oneway and two-way block move commands. There are two commands which are used to control collision detection:

```
DTCTON - turn collision detection on  
DTCTOFF - turn collision detection off
```

It is always best to switch off collision detection when it is not needed, because it slows down all the block move commands significantly.

After a block move command, the sprite variables CCOL and CROW will contain the column and row of any collision, measured in character blocks. If no collision is found, both of these are set to -1. If there are several collisions in the window being copied, Laser BASIC will only record the first one that it finds.

LASER BASIC FOR THE COMMODORE 64

Remember that sprite windows are copied starting at the top left hand corner, in the same order as one would read a book.

When collision detection is used with the one-way block move commands, CCOL and CROW contain the co-ordinates of the collision the destination sprite - with the PUT command, the destination sprite is on the screen, with MOV and CPY it is sprite no. SPN2, and with GET it is sprite no. SPN.

When the two-way block moves are used with collision detection, CCOL and CROW will be a co-ordinate inside sprite no. SPN2.

If you are working in multi-colour mode, Laser BASIC will only detect collisions between colours 2 and 3. Colour no. 1 is regarded as being transparent to collisions, like the background (colour 0).

Example: This program places sprites 10, 20, 25, 32, 28 and 33 on the screen at random positions. Collision detection is used to ensure that they do not overlap.

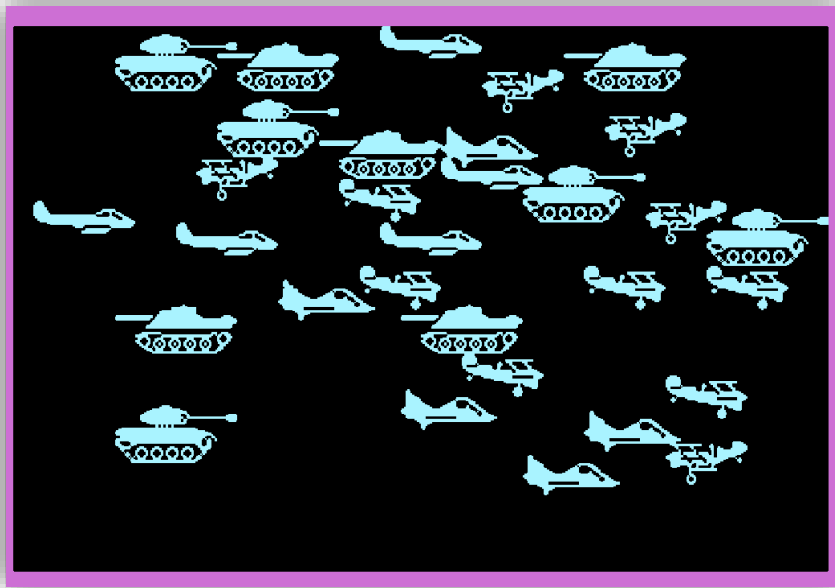
```
10 INIT : HIRES : BGND BLACK : FGND CYAN
20 SCLR 0,ATR : HORDER PURPLE: ATTOFF : DTCTON
30 FOR I=0 TO 30
40  READ J : IF J=-1 RESTORE : READ J
50  PUTXOR J,RND(1)* 35,RND(1) * 20
60  IF CCOL <> -1 THEN PUTXOR : GOTO 50
70 NEXT I
80 DATA 10,20,25,32,28,33,-1
90 GOTO 90
```

Available on disk as "ex sprite 07"

The first two lines of the program set up the screen - note that collision detection is turned on in line 20. Lines 30 to 70 form the loop that places the sprites on the screen at random positions. The sprite numbers are read from the DATA in line 80 into the variable j. Line 50 XORs the sprite with the screen. If no collision has occurred, the sprite will be displayed on the screen, and CCOL and CROW will both be set to -1. If there has been a collision, CCOL and CROW will both be set to some value other than -1.

Line 60 checks for this collision, and if there has been a collision, the sprite is XORed with the screen again to remove it.

LASER BASIC FOR THE COMMODORE 64



ATTRIBUTE BLOCK-MOVE COMMANDS

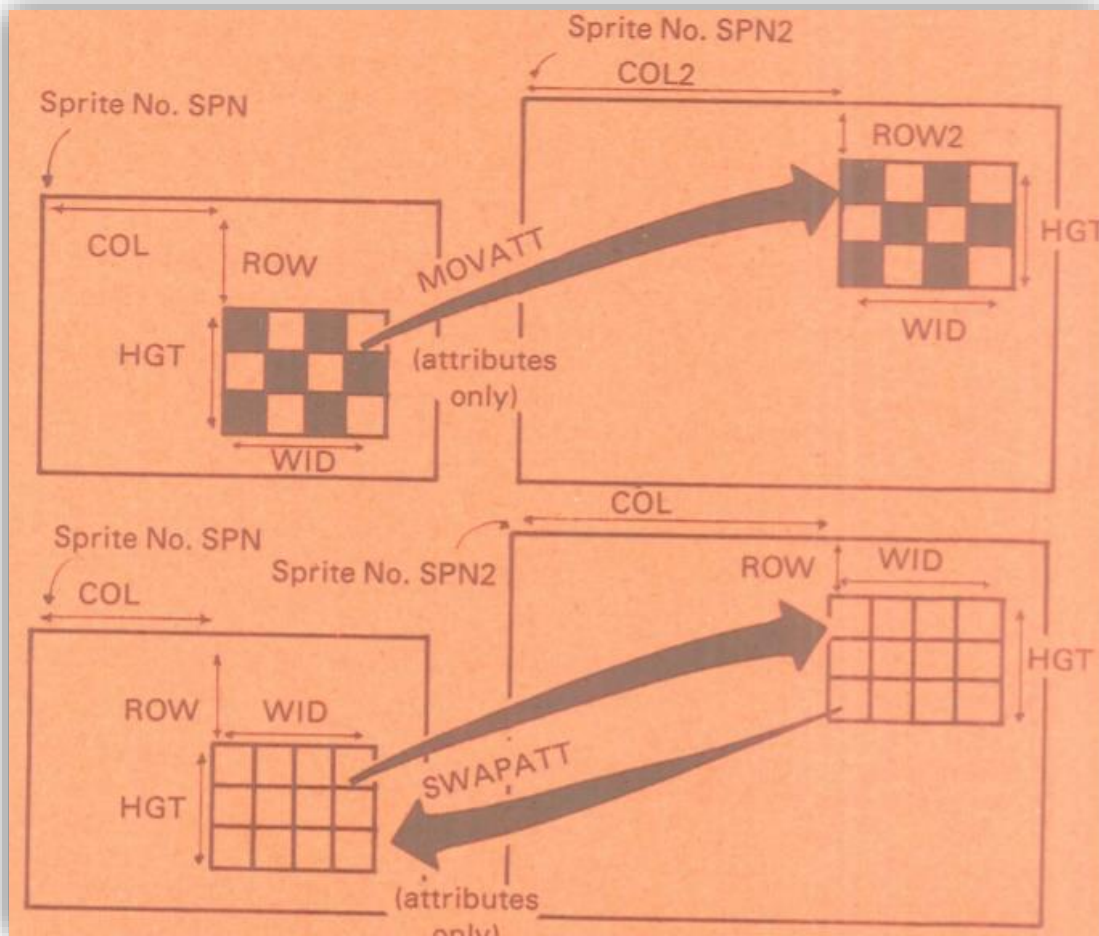
(MOVATT, SWAPATT, CPUT, CGET, CSWAP)

The block-move commands described so far move the pixel data, and perhaps the attributes as well, depending on whether ATTOFF, ATTON or ATT2ON have been used. There are also two commands, MOVATT and SWAPATT, which are block moves for attributes only. Both of them use the same set of parameters:

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

MOVATT is analogous to the MOVBLK command - the attributes are copied from a Window inside sprite no. SPN at position COL, ROW, width WID and height HGT to another window inside sprite no. SPN2 at position COL2,ROW2. SWAPATT is analogous to BLK%BLK - it swaps the attributes between the two windows:

LASER BASIC FOR THE COMMODORE 64



Example: This program draws 300 stars on the screen, and uses the SWAPATT command to make them change colour rapidly

```

10 FGND WHITE: BGND BLACK: SCLR 0,ATR
20 INIT : HBORDER PURPLE : HIRES
30 FOR I=0 TO 300
40 PLOT 0,RND(1)*319,RND(1)*200
50 COL=COL + 1 : PLOT
60 NEXT I
70 FOR X=0 TO 39
80 FOR Y=0 TO 24
90 FGND RND(1)* 16
100 SETA 0,X,Y,1,1,ATR
110 NEXT Y, X
120 REPEAT
130 SWAPATT 0,0,0,40,12,0,0,13
140 SWAPATT 0,0,0,20,25,0,20,0
150 UNTIL FALSE

```

Available on disk as "ex sprite 08"

LASER BASIC FOR THE COMMODORE 64

Lines 10 and 20 set up the hires screen. The FOR loop from lines 30 to 60 PLOTs the stars on the screen. Lines 70 to 110 set the attributes of each character block of the screen to a random value. The SWAPATT commands in lines 130 and 140 make the stars twinkle by swapping attributes between the left and right, and top and bottom halves of the screen alternately.



In general, these commands are very useful whenever you need to change the colour of something on the screen. They can also be used, in conjunction with the SETA command, to make objects appear and disappear from the screen very rapidly.

To make an object vanish, set the background and foreground to the same colour, and to make it appear again, put the attributes on the screen again using MOVATT. there are three other attribute commands - CPUT, CGET and CSWAP. Unless you are using text mode graphics, these are not important. Text mode graphics and smooth scrolling are described later in [CHARACTER SPRITES AND SMOOTH SCROLLING](#).

SPRITE TRANSFORMATIONS AND OTHER MISCELLANEOUS COMMANDS

The commands in this section carry out transformations on sprites, such as inversion and rotation. These are slow compared to the block move commands covered in the last section, so they should usually be used before the sprite is displayed on the screen. For example, you might have a spaceship which you want to display in four orientations on the screen - pointing up, down, left or right. You could use the rotation command to generate four sprites (one for each direction) from just one sprite produced using the sprite generator program.

HORIZONTAL REFLECTION

(FLIP, FLIPA)

LASER BASIC FOR THE COMMODORE 64

The FLIP command is used to reflect a sprite window around a horizontal line through its centre (i.e. the window is turned upside-down).

The command has five parameters which define the sprite window in the normal way (see [THE CONCEPT OF SPRITE WINDOWS](#)):

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

As usual, COL, ROW, WID and HGT are all measured in character blocks.

Depending on whether the ATTOFF, ATTON or ATT20N commands have been used, no attributes, primary attributes or all attributes will be reflected along with the pixel data.

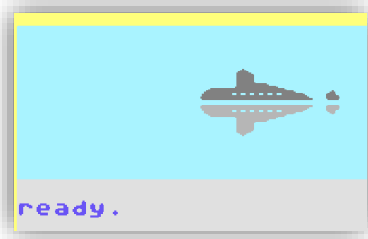
There is also a command to reflect the attributes only - FLiPA. This has the same parameters as FLIP. The secondary attributes are reflected only if the ATT20N command has been used.

Example: This program displays a submarine and its reflection in the water.

```
10 INIT:SCLR 0, CYAN  
20 WINDOW 8  
30 PUTBLK 31,10,2  
40 PUTBLK 31,10,4  
50 FLIP 0,10,4,8,2  
60 FGND GRAY2 : BGND CYAN  
70 SETA 0,10,4,8,2,ATR  
80 END
```

Available on disk as "ex sprite 08"

Lines 10 to 20 set up the hires screen, with a cyan background. In lines 30 and 40, two copies of sprite no. 31, the submarine, are put on the screen. The lower copy is turned upside-down in line 50, and in lines 60 and 70 the SETA command is used to make the reflection grey, so that it is lighter than the submarine itself.



VERTICAL REFLECTION

(MIR, MAR)

The MIR and MAR commands reflect a sprite window around a vertical line through its centre (i.e. the window is mirrored). In all respects other than the direction in which the reflection takes place, MIR and MAR are the same as FLIP and FLiPA respectively.

Example: This program uses MIR to create a reversed version of sprite no 1 the vintage car.

```
10 IF DFA(51) > 0 WIPE
```

LASER BASIC FOR THE COMMODORE 64

```
20 SPRITE 51,4,2
30 CPYBLK 1,51
40 MIR 51,0,0,4,2
```

Lines 10 and 20 create a new sprite (no. 51) to put the reversed car in. Line 10 deletes it if it already exists. The car is copied from sprite 1 to sprite 51 in line 30, and it is reversed by the MIR command in line 40.

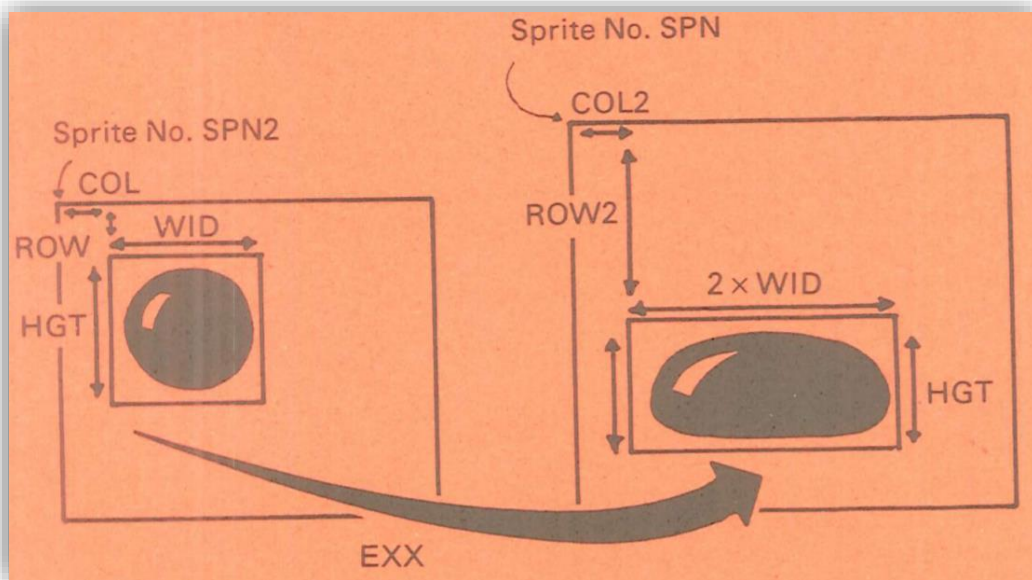
HORIZONTAL EXPANSION

(EXX)

The EXX command expands a sprite window by a factor of 2 in the X-direction. It has eight parameters:

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

The first five parameters specify the source window in the normal way. SPN2, COL2 and ROW2 are the sprite number, column and row of the destination window. However, the destination window is twice as wide as the source window - i.e. it is $WID \times 2$ character blocks wide and HGT character blocks high.



It is possible to expand a window into itself by having SPN, COL and ROW the same as SPN2, COL2 and ROW2. As usual, EXX can also expand the attributes, depending on whether ATTOFF, ATTON or ATT2ON have been used

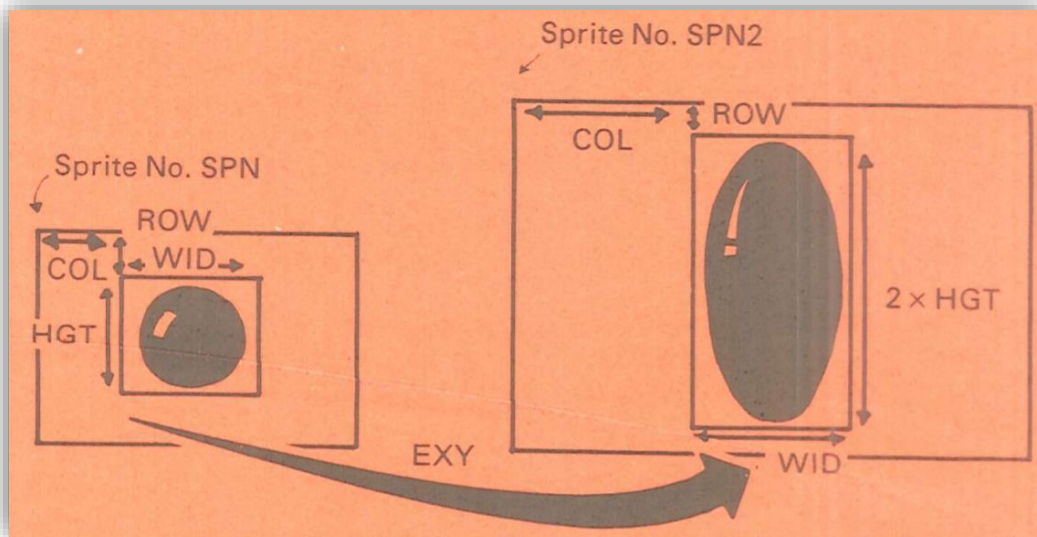
VERTICAL EXPANSION (EXY)

The EXY command is similar to EXX, the difference being that it expands in the Y-direction rather than the X-direction. It uses the same parameters as EXX

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

The destination window is twice as high as the source window - i.e. it is WID character blocks wide and $HGT \times 2$ character blocks high.

LASER BASIC FOR THE COMMODORE 64



Example: This program uses the EXX and EXY commands to convert sprite no. 2, the van, into sprite no. 51 twice the size:

```
10 PROC DOUBLE (2,51)
20 END
30 '
40 LABEL DOUBLE (S,D)
50 LOCAL I
60 IF DFA(D) > 0 WIPE
70 I = DFA(S)
80 SPRITE D,WID*2,HGT*2
90 CPYBLK S,D
100 I = DFA(S)
110 EXX D,0,0,WID, HGT,D,0,0
120 EXY D,0,0,WID*2,HGT,D,0,0
130 PROCEND
```

This program uses the procedure 'double' which starts in line 40 to do the expansion. In line 60, the destination sprite is WIPEd if it already exists. Using the dimensions of the source sprite obtained in line 70 with the OFA command, the new destination sprite is defined in line 80. The source sprite is copied to the top left hand part of the destination sprite in line 90 and the sprite is expanded in both directions in lines 110 and 120.

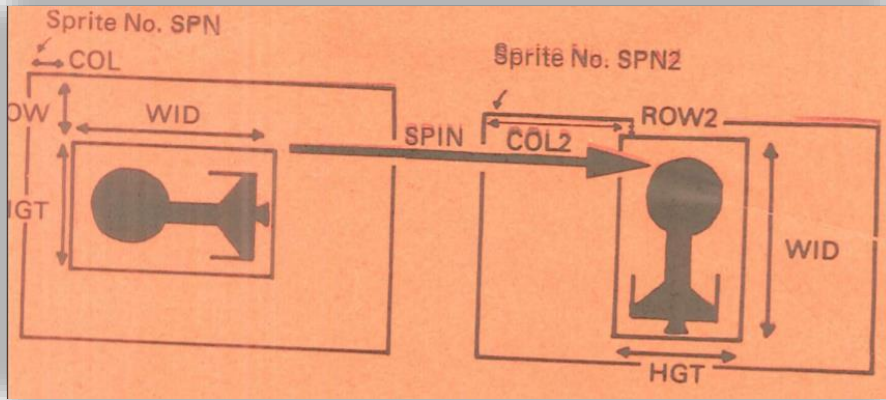
ROTATING A SPRITE WINDOW (SPIN)

The SPIN command allows you to rotate one sprite window into another. The rotation is 90 degrees clockwise, although it is possible to rotate a window by any multiple of this by using more than one SPIN. This is the only sprite transformation which cannot be used in four-colour mode. SPIN has eight parameters:

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

LASER BASIC FOR THE COMMODORE 64

The first five parameters define the source window in the normal way. The destination window is in sprite no. SPN2 at position COL2, ROW2. Since rotation is taking place, the destination window will be HGT pixels wide and WID pixels high.



Example' Using the SPIN command, this program defines sprites 51 , 52 and 53 to be copies of sprite no. 6 (a spaceship) rotated by 90, 180 and 270 degrees respectively.

```

10 FOR I=51 TO 54
20 IF DFA(I) > 0 WIPE
30 NEXT I
40 SPRITE 51, 2, 4
50 SPRITE 52, 4, 2
60 SPRITE 53, 2, 4
70 SPIN 6, 0, 0, 4, 2, 51, 0, 0
80 SPIN 51, 0, 0, 2, 4, 52, 0, 0
90 SPIN 52, 0, 0, 4, 2, 53, 0, 0
    
```

Lines 10 to 30 ensure that sprite 81 to \$4 do not exist. In lines 40 to 60 the sprites are dimensioned, The spin commands first rotate sprite 6 into 51, then 51 into 52 and 52 into 53,

INVERTING A WINDOW (INV)

The INV command can be used to invert all the pixels in a sprite window. That is, each pixel which was set to the background colour is set to the foreground and vice versa. In four-colour mode, the pixels are charged as follows:

Colour before INV	Colour after INV
0 (background)	3
1	2
2	1
3	0 (background)

This command has five parameters:

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

LASER BASIC FOR THE COMMODORE 64

These parameters specify the window that is to be inverted in the normal way (see section 4.6.15.2). Note that this command only alters the pixel data - it leaves the attributes untouched.

Example: This program allows you to guide a flashing cursor around the hires screen using the cursor keys. Every time you press the space bar, sprite no. 1 is put on the screen using the PUTXOR command. The INV command is used to provide the flashing cursor.

```
10 INIT : HBORDER BLUE
20 BGND BLUE : FGND WHITE
30 SCLR 0,ATR : ATTOFF : HIRES
40 UP$=CHR$(145) : DN$=CHR$(17)
50 LF$=CHR$(157) : RG$=CHR$(29)
60 X=0 : Y=0 : AF=FALSE
70 REPEAT
80 IF A$=UP$ AND Y>0 THEN Y=Y-1
90 IF A$=DN$ AND Y<23 THEN Y=Y+1
100 IF A$=LF$ AND X>0 THEN X=X-1
110 IF A$=RG$ AND X<36 THEN X=X+1
120 IF A$=" " PUTXOR 1,X,Y
130 GET A$
140 INV 0,X,Y,4,2 : AF = -1 - AF
150 IF AF AND AS <> " " : INV : AF = FALSE
160 UNTIL FALSE
```

Available on disk as "ex sprite 10"

The first three lines of the program set up the hires screen in the normal way. UP\$, DN\$, LF\$ and RG\$ are defined in lines 40 and 50 - these are the cursor control codes for up, down, left and right respectively. In line 60, x and y are the current co-ordinates of the cursor, and af is set to either TRUE or FALSE to indicate that the cursor is on or off respectively. In lines 80 to 110, the cursor co-ordinates x and y are altered if the current key pressed, a\$, is cursor up, down, left or right. In line 120, sprite 1 is put on the screen with PUTXOR if the space bar has been pressed. Line 140 inverts the variable af (i.e. if af was TRUE it is set to FALSE and vice-versa). In line 150, a check is made - if af is true (i.e. the cursor is on the screen) and a key has been pressed, the cursor is removed from the screen, so that nothing is left on the screen if the cursor is moved.



LASER BASIC FOR THE COMMODORE 64

SCANNING AN AREA OF THE SCREEN

(SCAN)

SCAN allows you to inspect a sprite window to see if any pixels are set in it. SCAN is a function, not a command, and it has five parameters:

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

These five parameters specify a sprite window in the normal way (see section 4.6.152). If all the pixels in the window are set to background (colour 0), the function will return the value of 0, or FALSE.

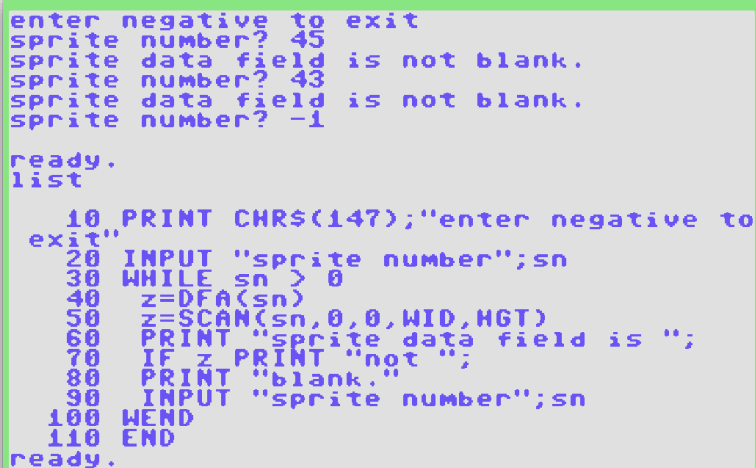
Otherwise, it returns a value of -1, or TRUE.

Example: This program will input a sprite number from the keyboard, and say whether the sprite data field is blank.

```
10 PRINT CHR$(147); "enter negative to exit"
20 INPUT "SPRITE NUMBER"; SN
30 WHILE SN > 0
40   Z = DFA(SN)
50   Z = SCAN( SN,0,0,WID,HGT)
60   PRINT "SPRITE DATA FIELD is ";
70   IF Z PRINT "NOT ";
80   PRINT "BLANK."
90   INPUT "SPRITE NUMBER"; SN
100 WEND
110 END
```

Available on disk as "ex sprite 11"

Line 10 obtains the sprite number, and in line 20, DFA is used to obtain the dimensions of the sprite. In line 30, the whole sprite is scanned, and the result is put in variable Z - TRUE if the sprite is not blank, and FALSE if it is blank. The remainder of the program prints out the result.



```
enter negative to exit
sprite number? 45
sprite data field is not blank.
sprite number? 43
sprite data field is not blank.
sprite number? -1

ready.
list
10 PRINT CHR$(147);"enter negative to
exit"
20 INPUT "sprite number";sn
30 WHILE sn > 0
40   z=DFA(sn)
50   z=SCAN(sn,0,0,WID,HGT)
60   PRINT "sprite data field is ";
70   IF z PRINT "not ";
80   PRINT "blank."
90   INPUT "sprite number";sn
100 WEND
110 END
ready.
```

LASER BASIC FOR THE COMMODORE 64

INSPECTING ATTRIBUTES

(ATTGET)

Using the ATTGET command, you can inspect the attributes at an y character block In a sprite. The command has three parameters:

ATTGET SPN, COL, ROW

COL and ROW are the position of the attribute to be interrogated (measured in character blocks), inside sprite no. SPN The attribute is placed in the sprite variable ATR. Only the primary attributes are read If ATTOFF or ATTON have been used, but both sets of attributes are read If ATT2ON has been used.

SCROLLING COMMANDS

Laser BASIC has 18 scrolling commands. These allow you to scroll attributes or pixel data up, down, left or right within a sprite window. They can be used in conjunction with the PUT commands to put sprites on the screen at pixel resolution, or to animate objects on the screen

SCROLLING PIXEL DATA HORIZONTALLY

There are twelve commands for scrolling pixel data:

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

All these commands have the same parameters, which specify the rectangular sprite window to be scrolled, in the normal way

SCR1 SPN, COL, ROW, WID, HGT

The commands differ in the direction of scrolling, number of pixels scrolled, and in whether the pixel data is wrapped round.

The first two letters of each command are either SC or WR. WR means that the pixel data is wrapped round, i.e. any pixel data which is scrolled of the edge of the window re-appears at the other side. SC means: that the pixel data is not wrapped round, i.e. any pixel data scrolled off the edge of the window is lost, and background is scrolled in at the other side.

The third letter is either R or L. denoting a scroll right or a scroll left respectively.

The number on the end of each command is the number of pixels by which the window is scrolled. This is either 1, 2 or 8. The 1 pixel scrolls can only be used in two-colour mode. because each pixel in four-colour mode is twice as wide. The 2-pixel scrolls are mainly for use in four colour mode there is little point in using a 2-pixel scroll in two-colour mode because It is no faster than two 1 -pixel scrolls. The 8-pixel scrolls scroll the window by an entire character block, and can be used in either two -colour or four-colour mode. Remember that this set of commands scroll the pixel data only - the attributes remain untouched.

Examples:

WRR1 scrolls the window right by 1 pixel with wrap.

SCL8 scrolls the window left by 8 pixels without wrap.

LASER BASIC FOR THE COMMODORE 64

SCR2 scrolls the window right by 2 pixels without wrap.

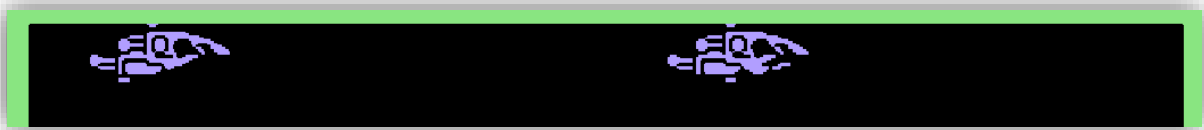
Example: This program uses the WRR8 command to move two spaceships (sprite no. 6) across the top of the screen:

```
10 INIT
20 BGND BLACK : FGND .BLUE
30 SCLR 0,ATR
40 HIRES : HBORDER BLACK
50 PUTBLK 6,0,0
60 PUTBLK 6,20,0
70 FOR 1=1 TO 320
80 WRR8 0,0,0,40,2
90 NEXT I
100 END
```

Available on disk as "ex sprite 12"

Remember that you must have the arcade sprite library loaded for this program.

Lines 10 to 40 set up the hires screen. The spaceships are put on the screen in lines 50 and 60, and they are scrolled along the top of the screen in lines 70 to 90. Note that the scrolling is with wrap - the spaceships are scrolled at the left-hand side of the screen as they disappear off the right-hand side.



If you run this program, you will see that the scrolling is not really satisfactory - the movement is not smooth and the graphics tend to flicker. Laser BASIC has commands which can circumvent these problems and are dealt with in the next section.

REPEATING A GRAPHICS COMMAND

(RPT)

Particularly when scrolling a window inside a sprite, a graphics command often has to be repeated several times. Of course, it is possible to use a FOR-NEXT loop for this purpose, as in the example above. The problem with this method is that the FOR-NEXT loop itself takes up too much time.

For this reason, Laser BASIC has a command called RPT which can be used to repeat a scrolling command several times. It is used as follows:

```
RPT n, scrolling command
```

n is the number of times that the command is to be executed.

Example: Here is an updated version of the example given in the last section. using the RPT command instead of a FOR-NEXT loop:

LASER BASIC FOR THE COMMODORE 64

```
10 INIT
20 BGND BLACK: FGND .BLUE
30 SCLR 0,ATR
40 HIRES : HBORDER GREEN
50 PUTBLK 6,0,0
60 PUTBLK 6,20,0
70 RSYNC 51
80 RPT 320,WRR8 0,0,0,40,2
Available on disk as "ex sprite 13"
```

If you run this program, you will see that the scrolling has speeded up compared with the previous version. However, the graphics still tend to flicker. This can be avoided by synchronising all the scrolls to the TV display and is dealt with in the next section.



AVOIDING FLICKER

(RASTER, RSYNC, UNSYNC)

A television or monitor displays a picture on the screen fifty times a second. The picture is built up by the electron beam scanning over the cathode ray tube one scan line at a time, starting at the top left of the screen. When part of the screen is updated, flicker can occur if it is displayed at the same time as it is updated. Therefore, to avoid this, it is necessary to ensure that when part of the screen is updated, it is not being displayed. The RASTER, RSYNC and UNSYNC commands exist for this purpose.

The RASTER command is followed by the line number:

RASTER line#

The next graphics command after RASTER will not be executed until just after that line number has been displayed. All commands after that will be executed as normal. The lines on the display are numbered starting from 51 at the top. Each pixel is counted as one scan line. So, since there are 25 lines of characters on the display, the scan lines are numbered from 51 to $51 + 25 \times 8 = 251$. There are also several scan lines which are not visible at the top of the screen, and these are numbered from 0 to 50.

When a window on the screen is being updated, it is usually best to synchronise to the top of the window. If the top of the window is at row no. ROW, this will be scan line no. $51 + \text{ROW} * 8$.

Example: This program uses the INV command to produce a flashing square at the top of left hand corner of the screen.

```
10 INIT : SCLR 0,1 : HIRES
20 REM
30 INV 0,0,0,8,8
40 FOR I=0 TO 10 : NEXT I
```

LASER BASIC FOR THE COMMODORE 64

```
50 GOTO 20
```

If you run this program, you will see an unpleasant "staircase" effect sometimes appearing when the square is inverted. This is because the square is being displayed at the same time that it is updated, and parts of it are displayed inverted while other parts are not inverted. To avoid this, line 20 should be changed to:

```
20 RASTER 51
```

The number 51 is used because the top of the window being updated, i.e. inverted, is at ROW no. 0, which is equivalent to scan line no. 51. If you run the program with this modification, no flicker occurs.

If a scrolling command is used in conjunction with RPT, RASTER cannot be used because synchronisation to the display would only take place the first time the scrolling command was executed. In this case, the RSYNC command would be used. Like RASTER, RSYNC should be followed by the scan line number:

```
RSYNC line#
```

However, all subsequent graphics commands will be synchronised to the display at the specified scan line. To turn the synchronisation off and return to normal. You should use the UNSYNC command. This command has no parameters (note that the INIT command automatically turns off all synchronisation).

Example: This program is an updated version of the example given in the previous section. It uses RSYNC to avoid any flicker.

```
10 INIT
20 BGND BLACK: FGND .BLUE
30 SCLR 0, ATR
40 HIRES : HBORDER BLACK
50 PUTBLK 6,0,0
60 PUTBLK 6,20,0
70 RSYNC 51
80 RPT 320, WRR8 0,0,0,40,2
```

If you run this program, the scrolling is now perfectly smooth. In line 70, the synchronisation is set to scan line 51 - this is equivalent to ROW no 0 which is at the top of the window being scrolled.

Example: This program demonstrates the use of collision detection, as well as scan line synchronisation.

```
10 INIT
20 BGND CYAN: FGND YELLOW SCLR 0,ATR
30 WINDOW 20
40 PUTBLK 21,12 + RND(1) * 24,10
50 DTCTON
60 PUTXOR 8,0,10
70 I=1
80 REPEAT
90 RASTER 131
100 PUTXOR 8,I-1,10
```

LASER BASIC FOR THE COMMODORE 64

```
110 COL = I
120 PUTXOR
130 I = I + 1
140 UNTIL CCOL <> -1
150 PRINT "COLLISION"
```

Available on disk as "ex sprite 14"

In line 40, an alien is placed on the screen, somewhere on ROW 10. In lines 70 to 120, sprite 8 is moved in from the left of the screen until a collision occurs. The collision is detected by checking the value in CCOL in line 120. In line 100, the previous copy of the spaceship is deleted using PUTXOR before the new copy is placed on the screen. Variable I is the column used.

The RASTER command in line 90 ensures that the animation is flicker free. Try removing line 90, to see the difference.



SCROLLING PIXEL DATA VERTICALLY

(SCROLL, WRAP)

As well as the twelve horizontal scrolling commands already described, Laser BASIC has two commands, WRAP and SCROLL, for scrolling a sprite window vertically with or without wrap. Like the horizontal scrolls, these commands only scroll the pixel data and leave the attributes untouched. Both commands have the same six parameters:

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

The first five parameters specify the sprite window which is to be scrolled, in the normal way (see [THE CONCEPT OF SPRITE WINDOWS](#)).

These two vertical scrolling commands are much more versatile than the horizontal scrolls, because scrolling can be by any number of pixels - the window is scrolled by NUM pixels. NUM can be any value between -127 and +127, a negative value meaning a scroll down and a positive value meaning a scroll up.

Obviously, WRAP is a scroll with wrap-around, and SCROLL has no wrap-around.

Example: This program uses the vertical scroll command to make sprite no 35, a frog hop across the screen each time the space bar is pressed

```
10 INIT
```

LASER BASIC FOR THE COMMODORE 64

```
20 BGND BLACK : FGND GREEN
30 SCLR 0,ATR
40 HBORDER CYAN : HIRES
60 PUTBLK 35,10,22
50 FOR Y=20 TO 0 STEP -2
70 REPEAT : GET A$ : UNTIL A$=" "
80 RSYNC 51 + Y * 8
90 RPT 8,SCROLL 0,18,Y,3,5,2
100 NEXT Y
```

Available on disk as "ex sprite 15"

The first four lines of this program Set up the hires screen. Note that in line 30, the attribute for the whole screen are set to a black background with a green foreground. This is necessary because otherwise the frog would vanish when it scrolled, since the SCROLL command does not scroll the attributes. In line 50 the frog is placed at the bottom of the screen. The remainder of the program forms the loop which scrolls the frog up by 2 character blocks every time the space bar is pressed.



Note the use of RSYNC in line 80 to make the scrolling smooth.

SCROLLING ATTRIBUTES

(ATTL, ATTR, ATTUP, ATTDN)

Laser BASIC has four commands which scroll the attributes within a sprite window. These are:

- ATTL - scroll attributes one character block left.
- ATTR - scroll attributes one character block right.
- ATTUP - scroll attributes one character block up.
- ATTDN - scroll attributes one character block down.

With these commands, the attributes which are scrolled off the edge of the window are always wrapped around to the other side. All four commands use the same five parameters, which specify the sprite window to be used:

ATTL SPN, COL, ROW, WID, HGT

If the ATTOFF or ATTON commands have been used, only the primary attributes are scrolled. If ATT20N has been used, both primary and secondary attributes are scrolled.

Example: This program uses the ATTR command to display multicoloured patterns on the screen.

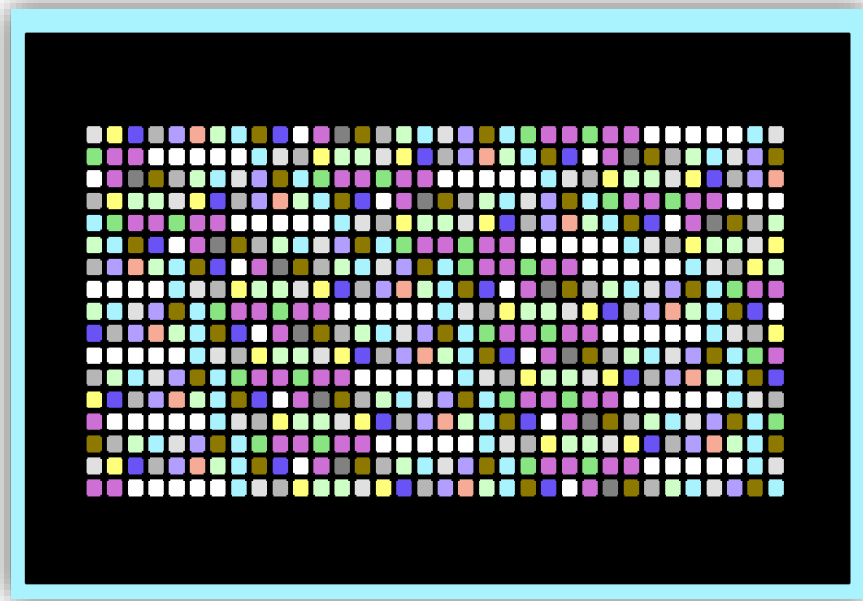
```
10 INIT
20 FBND WHITE : BGND BLACK
30 SCLR 0,ATR : HBORDER BLACK
```

LASER BASIC FOR THE COMMODORE 64

```
40 HIRES
50 BOX 0,24,34,6,6
60 MOVBLK 0,3,4,33, 1,0,4,4
70 MOVBLK 0,3,4,34,16,0,3,5
80 FOR X=3 TO 36
90  FGND RND(1)*15+1
100  SETA 0,X,4,1,18,ATR
110 NEXT X
120 FOR S=1 TO 38
130  FOR N=4 TO 21
140    RPT N,ATTR 0,1,N-1,38,1
150  NEXT N
160 NEXT S
170 REPEAT
180 UNTIL FALSE
```

Available on disk as "ex sprite 16"

The first four lines of the program set up the hires screen. In lines 50 to 70, an array of squares is placed on the screen. Note how MOVBLK is used to fill up a rectangular area of the screen rapidly. Lines 80 to 110 set up the colours of the squares, at random. The ATTR command is used, inside a double loop, to re-arrange the pattern of colours.



TWO PROGRAMMING EXAMPLES

Example 1: One use of the scrolling commands is to put sprites on the screen at pixel resolution. Of course, this is much slower than using an ordinary PUT command and is certainly not fast enough for high speed animation. Fortunately, the 64's hardware sprites, which are dealt with in a later section, can be used if speed is required.

Here is a procedure which PUTs sprites on the screen at pixel resolution.

LASER BASIC FOR THE COMMODORE 64

```
10 LABEL PIXPUT(S,C,R)
20 LOCAL I
30 IF DFA(254) < 0 SPRITE 254,8,8
40 ATTGET S,0,0 : SCLR 254, ATR
50 CPYBLK, S,254 : I = DFA(S)
60 RPT C AND 7, SCR1, 254,0,0,WID+1, HGT+1
70 NUM = -1: RPT R AND 7,SCROLL
80 MOVBLK 254,0,0,WID,HGT,0,C/8, R/8
90 PROCEND
```

The procedure has three parameters - the sprite number and the column and row on the screen, measured in pixels. Sprite no. 254 is used to store the sprite temporarily, before placing it on the screen. In line 30, sprite 254 is created if it does not already exist. In line 40, the ATTGET command is used to obtain the sprite's attribute, and this is used when clearing sprite no. 254. In line 50, the sprite to be displayed is copied into the top left hand corner of sprite 254, and the DFA command is used to set WID and HGT to the size of the sprite.

Lines 60 and 70 scroll the pixel data in sprite no. 254, so that it will be correctly positioned when PUT on the screen. Note that R AND 7 is simply the remainder upon dividing r by 8, and it could be written as $r-8 * \text{INT}(r / 8)$.

Finally, the shifted sprite is placed on the screen using a PUTBLK command in line 80.

Example 2: This program shows how the scrolling commands can be used to move a sprite around the screen under control of the cursor keys.

```
10 BGDN CYAN: FGND BLUE: SCLR 0,ATR
20 HIRES: X=0 : Y=0 : PUTBLK21,0,0
30 UP$=CHRS(145) : DNS=CHRS(17)
40 LF$=CHRS(157) : RG$=CHRS(29)
50 REPEAT
55 GET AS
60 CIF A$=RG$ AND X<36
70 REPEAT: RASTER 51+Y : SCR1 0,X,Y,5,2
80 X=X+1
85 REPEAT: GET AS : UNTIL AS<>" "
90 UNTIL AS<>RG$ OR X>35
95 CEND
100 CIF A$ = LFS AND x > 0
110 REPEAT: X=X-1 : RASTER 51+Y
120 SCLB 0,X,Y,5,2
125 REPEAT: GET AS : UNTIL AS <> " "
130 UNTIL AS <> LFS OR X=0
135 CEND
140 CIF AS=DNS AND Y<23
150 REPEAT: RASTER 51+Y
155 SCROLL 0,X,Y,4,3,-8
160 Y=Y+1
165 REPEAT: GET AS : UNTIL AS<>" "
```

LASER BASIC FOR THE COMMODORE 64

```
170 UNTIL AS <> DNS OR Y > 22
175 CEND
180 CIF AS=UP$ AND Y>0
190 REPEAT: Y=Y-1 : RASTER 51+Y
200 SCROLL 0,X,Y,4,3,8
205 REPEAT : GET AS : UNTIL AS<>" "
210 UNTIL AS <> UP$ OR Y=0
215 CEND
220 UNTIL FALSE
```

Available on disk as "ex sprite 17"

The hires screen is initialised in the first two lines of the program. Variables x and y are used throughout the program to keep track of the sprites position on the screen. Strings up\$, dn\$, lf\$ and rg\$ set up in lines 30 and 40, to be the four cursor control codes. Lines 50 onwards form an endless loop which is used to move the sprite around the screen. When a key depression is detected, the sprite is moved by scrolling the appropriate part of the screen.



LASER BASIC FOR THE COMMODORE 64

READING THE KEYBOARD, JOYSTICK AND LIGHTPEN

DETECTING KEY DEPRESSIONS (KB)

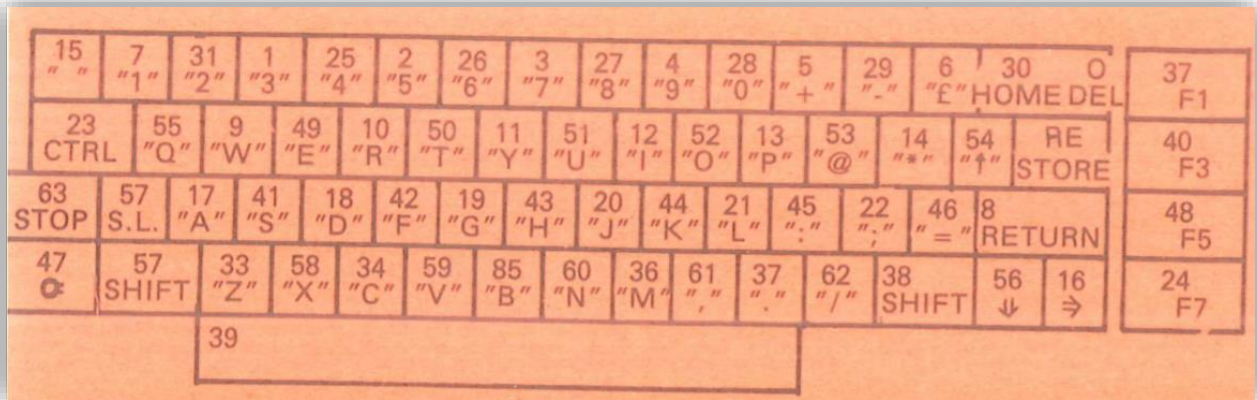
Commodore BASIC already has a GET command which can be used to detect key depressions. However, it suffers from the disadvantage that it cannot detect multiple key depressions, nor can it be used to test for the SHIFT or COMMODORE keys.

KB(<key>) is a function which you can use to test for Individual keys being depressed. Each key is assigned a number between 0 and 63 which is placed in brackets after the KB. It will return a value of TRUE (-1) if the key is depressed, and FALSE (0) otherwise.

The keys are numbered as follows:

Key	C64 Keyboard	US Keyboard	Key	C64 Keyboard	US Keyboard
0	Inst/Del	Del / Backspace	32	F1	F1
1	3		33	Z	
2	5		34	C	
3	7		35	B	
4	9		36	M	
5	+	+	37	. (decimal point)	
6	#	#	38	R.H. Shift	Right Shift
7	1 (one)		39	Space Bar	Space Bar
8	Return	Enter	40	F3	F3
9	W		41	S	
10	R		42	F	
11	Y		43	H	
12	I		44	K	
13	P		45	: (colon)	:
14	*	*	46	=	-
15	← Char	End key	47	Commodore Key	Windows or L Ctrl
16	Left, Right Arrow	Right Cursor	48	F5	F5
17	A		49	E	
18	D		50	T	
19	G		51	U	
20	J		52	O (oh)	
21	L		53	@	@
22	;	;	54	Up Arrow Char	^ shift 6 or ~
23	CTRL	CTRL or TAB	55	Q	
24	F7	F7	56	Up, Down Arrow	Down Cursor
25	4		57	Shift Lock, L.H. Shift	Left Shift
26	6		58	X	
27	8		59	V	
28	0 (zero)		60	B	
29	_ (underscore)	_	61	,	,
30	CLR/HOME	Home	62	/	/
31	2		63	RUN/STOP	Esc

LASER BASIC FOR THE COMMODORE 64



Example: This program uses KB to detect if either of the shift keys are pressed. It prints l or r at the top of the screen corresponding to the left and right shift keys respectively.

```

10 REPEAT
20 PRINT CHR$(147); CHR$(19);
30 IF KB(57) PRINT "LEFT SHIFT";
40 IF KB(38) PRINT "RIGHT SHIFT";
50 IF KB(47) PRINT "COMMODORE";
60 IF KB(34) PRINT "CTRL";
70 IF KB(8) PRINT "RETURN";
80 IF KB(30) PRINT "HOME";
90 IF KB(0) PRINT "DEL";
100 IF KB(56) PRINT "DOWN CURSOR";
110 IF KB(16) PRINT "RIGHT CURSOR";
120 IF KB(54) PRINT "↑ CHARACTER";
130 IF KB(15) PRINT "← CHARACTER";
140 UNTIL FALSE
150 END

```

Available on disk as "ex sprite 17"

CHR\$(147) clears the screen, and CHR\$(19) homes the cursor.

READING THE JOYSTICK

(FIRE1, FIRE2, JS1, JS2)

Laser BASIC has four functions for use with joysticks in port 1 or port 2.

FIRE1 and FIRE2 read the fire buttons on the joysticks in port 1 and port 2 respectively. If the button is pressed, FIRE1 or FIRE2 returns a value of TRUE (-1); otherwise it returns a value of FALSE (0).

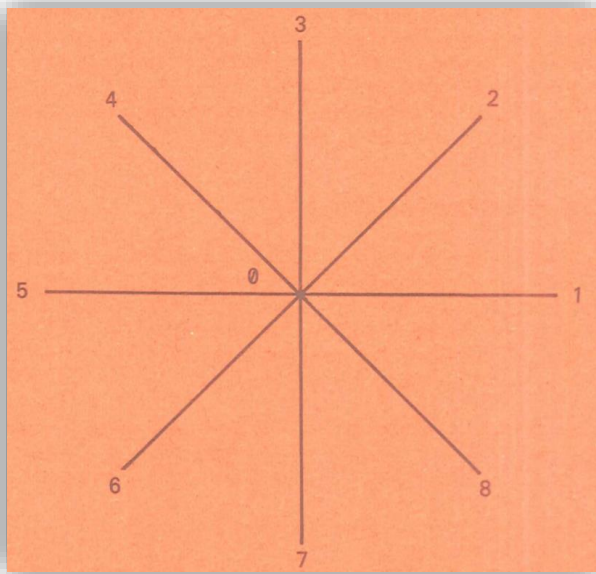
Example: This is a modified version of a program given earlier, in section [SCROLLING PIXEL DATA VERTICALLY \(SCROLL, WRAP\)](#). Line 70 has been changed, so that the fire button on joystick no. 1 is used to scroll the sprite over the screen.

LASER BASIC FOR THE COMMODORE 64

```
10 INIT
20 BGND BLACK
25 FGND GREEN
30 SCLR 0,ATR
40 HBORDER BLACK
45 HIRES
50 PUTBLK 35,18,22
60 FOR Y=20 TO 0 STEP- 2
70 REPEAT
75 UNTIL FIRE1
80 RSYNC 51 + Y * 8
90 RPT 8,SCROLL 0,18,Y,3,5,2
100 NEXT Y
```

Available on disk as "ex joystick 01"

JS1 and JS2 give the direction of the two joysticks. If the Joystick is in its central position, JS1 or JS2 return a value of 0. Otherwise they return a value of 1 to 8, representing the direction of the joystick.



Example: This program moves sprite no. 21 around the screen under control of joystick 1.

```
10 BGND CYAN : FGND BLUE : SCLR 0,ATR
20 HIRES : PUTBLK 21,0,0
30 REPEAT
40 REPEAT : D=JS1 : UNTIL D<>0
50 IF D=1 OR D=2 OR D=8 THEN X=X+1
60 IF D=4 OR D=5 OR D=6 THEN X=X-1
70 IF D=2 OR D=3 OR D=4 THEN Y=Y-1
80 IF D=6 OR D=7 OR D=8 THEN Y=Y+1
90 RSYNC 43 + Y + 8
100 PUTXDR 21,X,Y : PUTXOR 21,OX,OY
```

LASER BASIC FOR THE COMMODORE 64

```
110 DX=X: DY=Y
120 UNTIL FALSE
```

Available on disk as "ex joystick 02"

The variables x and y keep track of the sprite's current position on the screen. ox and oy are the previous position of the sprite. In line 40, the program goes around in a loop until the joystick is moved away from the central position. Lines 50 to 80 adjust x and y, the sprite's new position, depending on the direction of the joystick. In line 100 the sprite is first put on the screen at its new position, using PUTXOR, and then removed from the old position with another PUTXOR. Finally, ox and oy are updated before going around the loop again. Note that no checks are made to stop the sprite going off the screen in this program.

USING A LIGHTPEN

(LPX, LPV)

There are two functions in Laser BASIC, LPX and LPY, which allow you to use a lightpen in your programs. LPX and LPY return the x and y coordinates of the lightpen. With LPX, a value of 23 corresponds to the far left of the screen and 182 the far right; a change in LPX of 1 corresponds to two pixels. With LPY, the top of the screen is at 51, and the bottom is at 250; a change in LPY of 1 corresponds to 1 pixel.

If LPX and LPY are used directly, the lightpen reading tends to be rather unsteady, and it is best to use some kind of smoothing on lightpen values. The simplest way of doing this is to use the following formula:

$$\text{new reading} = n * \text{old reading} + (1 - n) * \text{LPX}$$

n is a constant between 0.5 and 0.9. Higher values of n give a greater degree of smoothing.

For example, to read the lightpen into the variables lx and ly, the following subroutine could be used:

```
100 LABEL READPEN
110 LX= 0.7 * LX + 0.6 * (LPX-23)
120 LY= 0.7 * LY + 0.3 * (LPV-51)
130 RETURN
```

In this case, n=0.7. Note that the value of LPX is multiplied by 0.6 and not 0.3, because LPX is at a resolution of 2 pixels. Also, LPX and LPY are adjusted in the subroutine so that the results of LX and LY are the coordinates within the hires screen.

Example: This program uses the above subroutine to draw on the screen under control of the lightpen.

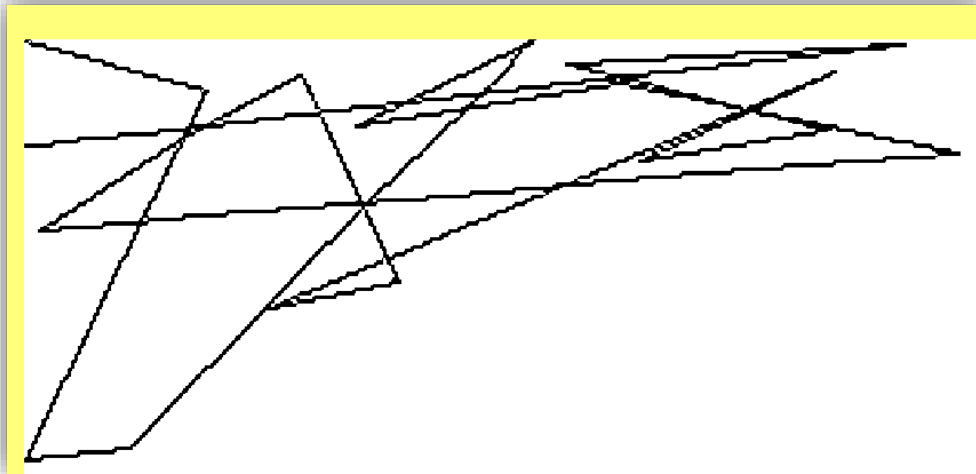
```
10 INIT : SCLR 0,1 : HIRES
20 REPEAT
30 GOSUB READPEN
40 C1F LX >=0 AND LX <320 AND LY >=0 AND LY <200
50 DRAW 0,X,V,LX,LV
60 X=LX
65 V=LV
```

LASER BASIC FOR THE COMMODORE 64

```
70 CEND
80 UNTIL FALSE
90 `
100 LABEL READPEN
110 LX = 0.7 * LX + 0.6 * (LPX-23)
120 LV = 0.7 * LV + 0.3 * (LPY-51)
130 RETURN
```

Available on disk as "ex lightpen 01"

Line 10 clears the hires screen. Line 40 checks that the point is on the screen. A line is drawn from the previous light pen position to the present position in line 50; the variables x and y store the previous light pen position.



LASER BASIC FOR THE COMMODORE 64

CHARACTER SET MANIPULATION

Laser BASIC has commands which allow you to change the character set which is used in the text mode. This feature is essential for the effective use of character sprites, which are discussed in [SPRITE VARIABLES](#). It is also possible to put text into a hires sprite. One use would be labelling a graph drawn on the hires screen.

LOWER CASE AND UPPER CASE

(LCASE, UCASE)

Normally, Laser BASIC displays all text in lower case. It is possible to switch between lower and upper case letters by using:

LCASE for lower case

UCASE for upper case

Both of these commands affect the whole screen.

ASCII AND DISPLAY CODES

Characters are stored inside a computer as numbers between 0 and 255, using the ASCII code. The function ASC can be used to give the ASCII code value of a character:

```
PRINT ASC ("a")
```

If typed in from the keyboard in immediate mode, the above will print out a value of 65 which is the ASCII code for "a". Conversely, a number can be converted into all ASCII character using CHR\$:

```
PRINT CHR$(66)
```

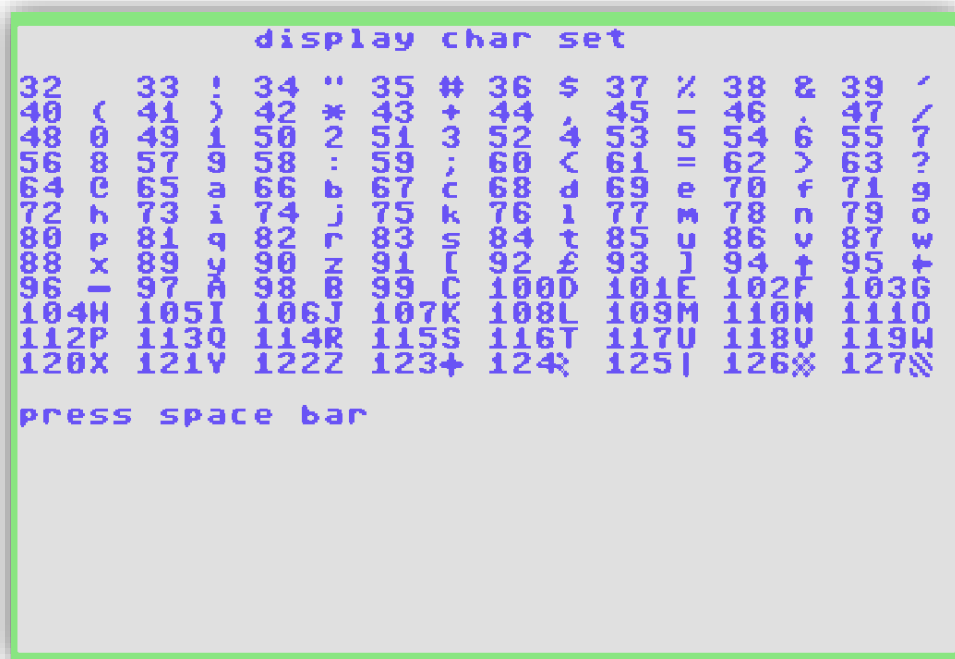
The above line would make the computer print "b", which has the ASCII code 66.

The following program prints all the printable characters, along with their ASCII code values. Some ASCII codes perform functions such as moving the cursor, and therefore cannot be printed. For example, the CHR\$(147) in the first line clears the screen.

```
10 PRINT CHR$(147) ;
20 FOR J=0 TO 128 STEP 128
30   FOR I=32 TO 127
40     PRINT MIDS(STR$(I+J) + "    ",2,3) ; CHR$(I+J) ; " ";
50   NEXT I
55 NEXT J
60 GOTO 60
```

However, when the computer prints a character by putting it in the screen memory, it converts the ASCII code into a 'display code'. For most characters, the ASCII and display codes are different.

LASER BASIC FOR THE COMMODORE 64



Example: This program will print out the whole character set, along with display codes

```

10 FOR J=0 TO 128 STEP 128
20 PRINT CHR$(147)
22 PRINT TAB(13); "DISPLAY CODES"
24 PRINT
30 FOR I=0 TO 127
40 PRINT MIDS(STR$(I+J)+ " ",2,5) ;
50 POKE SD87B + (I AND 7) * 5 + INT(I/8) * 40,I + J
60 POKE $D87B + (I AND 7) * 5 + INT(I/8) * 40,PEEK (646)
70 NEXT I
80 CIF J=0
90 PRINT
100 PRINT "PRESS SPACE BAR"
110 REPEAT
120 GET AS
130 UNTIL AS=" "
140 CEND
150 NEXT J

```

Available on disk as "ex charset 01"

If you run the program, you will see that the characters with display codes from 128 to 255 are reversed versions of the characters from 0 to 127. The computer uses these reversed characters to make the cursor flash in text mode.

LASER BASIC FOR THE COMMODORE 64

display codes

0		1	a	2	b	3	c	4	d	5	e	6	f	7
1	x	9	1	10	7	11	k	12	1	13	J	11	+	15
2	p	17	1	18	r	19	s	20	t	21	J	22	+	23
3		25	1	26	r	27	L	28	t	29	%	30	+	31
4		33	1	34	:	35	#	36	t	37	=	38	+	39
5		41	1	42	*	43	+	44	4	45	5	46	+	47
6		49	1	50	:	51	3	52	4	53	5	54	+	55
7		57	1	58	:	59	.	60	4	61	=	62	+	63
8		65	1	66	:	67	C	68	4	69	=	70	+	71
9		73	1	74	:	75	C	76	4	77	=	78	+	79
10	x	81	1	82	:	83	K	84	4	85	5	86	+	87
11	p	89	1	90	:	91	S	84	4	93	5	86	+	95
12		97	1	98	:	99	+	92	4	101	5	94	+	103
13		105	1	106	:	107	+	100	4	109	5	102	+	111
14		113	1	114	:	115	+	108	4	117	5	110	+	119
15		121	1	122	:	123	.	116	4	125	5	118	+	127
16								124	.			126	+	

press space bar

display codes

128		129		130		131		132		133		134		135
136		137		138		139		140		141		142		143
144		145		146		147		148		149		150		151
152		153		154		155		156		157		158		159
160		161		162		163		164		165		166		167
168		169		170		171		172		173		174		175
176		177		178		179		180		181		182		183
184		185		186		187		188		189		190		191
192		193		194		195		196		197		198		199
200		201		202		203		204		205		206		207
208		209		210		211		212		213		214		215
216		217		218		219		220		221		222		223
224		225		226		227		228		229		230		231
232		233		234		235		236		237		238		239
240		241		242		243		244		245		246		247
248		249		250		251		252		253		254		255

ready .

LASER BASIC FOR THE COMMODORE 64

PUTTING A CHARACTER INTO A SPRITE

(CHAR)

The CHAR command takes a single character and places it at a specified point inside a hires sprite. It has four parameters:

CHAR SPN, COL, ROW, NUM

The character is placed in sprite no. SPN, at position COL,ROW. COL and ROW are both measured in character blocks. NUM is the code of the character to be displayed, and can represent either ASCII, reversed ASCII or a display code. To display a character using ASCII, set NUM equal to the ASCII code. To display the character in reverse, add 256 to the ordinary ASCII code. For a display code, add 512.

CHAR can also put double width characters into the sprite. To do this, add 1024 to the value of NUM. This is all summarised below:

Start	End	Character Displayed
0	255	ASCII character no. NUM
256	511	Reversed ASCII character no. NUM-256
512	767	Display code no. NUM-512
1024	1279	Double width ASCII character no. NUM-1024
1280	1535	Reversed double width ASCII character no. NUM-1280
1536	1791	Double width display code no. NUM-1536

The CHAR command does not affect the attributes inside a sprite – only the pixel data is changed.

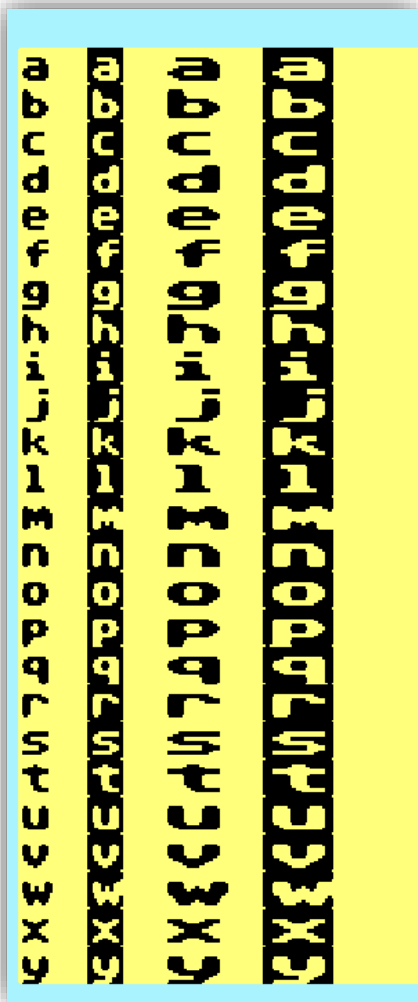
Example: This program uses the CHAR command to display the first 25 letters of the alphabet on the hires screen as normal, reversed, doublewidth and double-width reversed characters.

```
10 INIT
20 BGND YELLOW : FGND BLACK
30 SCLR 0,ATR
40 HIRES
50 FOR i=0 TO 24
60 CHAR 0,0,I,I + 65
70 CHAR 0,2,I,I + 65 + 256
80 CHAR 0,4,I,I + 65 + 1024
90 CHAR 0,7,I,I + 65 + 1280
100 NEXT 1
110 GOTO 110
```

Available on disk as "ex charset 02"

The FOR-NEXT loop from line 50 to line 100 displays all the first 25 (to match screen rows) characters of the alphabet on the screen. In lines 60 to 90, the ordinary characters are displayed in column 0, reversed characters in column 2, double-width characters in column 4, and reversed double-width characters in column 7. Remember that the ASCII code for "a" is 65, "b" is 66 and so on.

LASER BASIC FOR THE COMMODORE 64



Example: This procedure copies the entire character set into a 16x16 hires sprite. This is a standard format which is used by the sprite generator program.

```
10 LABEL UPLOAD(S)
20 IF DFA(S) <0 OR WID <>16 OR HGT <>16 STOP
30 FOR I=0 TO 15
40   FOR J=0 TO 15
50     CHAR S,I,J,512+I+J*16
60   NEXT J
70 NEXT I
80 PROCEND
```

In line 10, the parameter S is the sprite number to be used. A check is made in line 20 to see if the sprite exists and is the correct size. Line 50 places the characters in the sprite, according to the display code.

Example: This program shows how CHAR can be used in conjunction with EXX and EXY to display large characters. It displays the ASCII and reversed ASCII character sets on the screen.

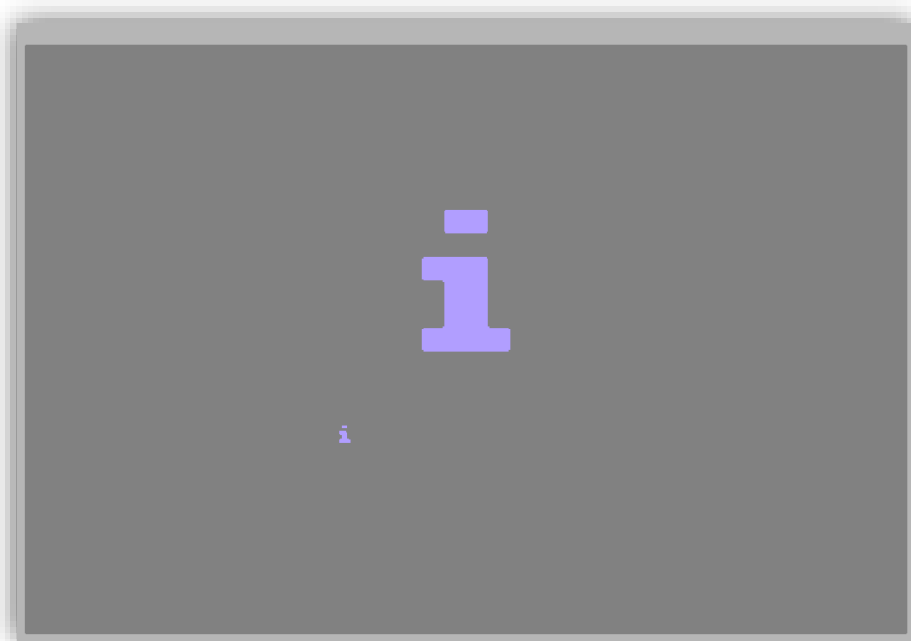
LASER BASIC FOR THE COMMODORE 64

```
10 INIT : BGND GRAY1 : FGND .BLUE
20 SCLR 0,ATR : HBORDER GRAY1 : WINDOW 23
30 IF DFA(200) > 0 THEN WIPE
40 SPRITE 200,8,8 : TPAPER GRAY1 : INK .BLUE
50 FOR M=0 TO 384 STEP 128
60   FOR N=M+32 TO M + 127
70     CHAR 200,0,0,N
80     RPT 3,EXX 200,0,0,4,1,200,0,0
90     RPT 3,EXY 200,0,0,8,4,200,0,0
100    CHAR 0,14,16,N
110    RASTER 179
120    PUTBLK 200,16,6
130    PRINT
140  NEXT N
150 NEXT M
160 WIPE 200
170 END
```

Available on disk as "ex charset 03"

The first four lines of the program initialise the hires screen window and set up sprite 200. This is a temporary sprite, 8x8 characters, which is used to hold the large character before placing it on the screen. Two nested FOR-NEXT loops are used from line 50 onwards - this is to avoid all non-printing (blank) ASCII characters. In line 70, the character is placed in the top left hand corner of sprite no. 200, and it is enlarged horizontally and vertically so that it fills the whole sprite. In lines 110, 120 and 130, the character is put on the screen as both normal size and 8 times normal size and the ASCII code is printed at the bottom of the screen as well.

Finally, sprite 200 is deleted before exiting the program.



LASER BASIC FOR THE COMMODORE 64

PUTTING A STRING INTO A SPRITE (TEXT)

The TEXT command is used to place a whole string inside a sprite, rather than just one character, as with CHAR. TEXT has five parameters:

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

The string is put into sprite no. SPN, as position COL,ROW <offset> is a number which is added to the ASCII value of each character before it is placed inside the sprite. By setting the offset to a multiple of 256, you can use TEXT to obtain reversed or double-width characters, in the same way as with CHAR.

The most useful offset values are given below:

Offset	Characters Displayed
0	Ordinary text
256	Reversed
1024	Double-width
1280	Reversed double-width
2048	Double-spaced
2304	Reversed double-spaced

Example: This program shows the various text styles available with the TEXT command:

```
10 INIT
20 BGND BLACK : FGND WHITE
30 SCLR 0,ATR
40 WINDOW 6
50 TEXT 0,0,0, "ORDINARY" , 0
60 TEXT 0,0,1, "REVERSE" , 256
70 TEXT 0,0,2, "DOUBLE WIDTH" , 1024
80 TEXT 0,0,3, "REVERSE DOUBLE WIDTH" , 1280
90 TEXT 0,0,4, "DOUBLE SPACE" , 2048
100 TEXT 0,0,5, "REVERSE DOUBLE SPACE" , 2304
```

Available on disk as "ex charset 04"



Example: This program draws a bar graph on the hires screen, and it uses the TEXT command to label the axes.

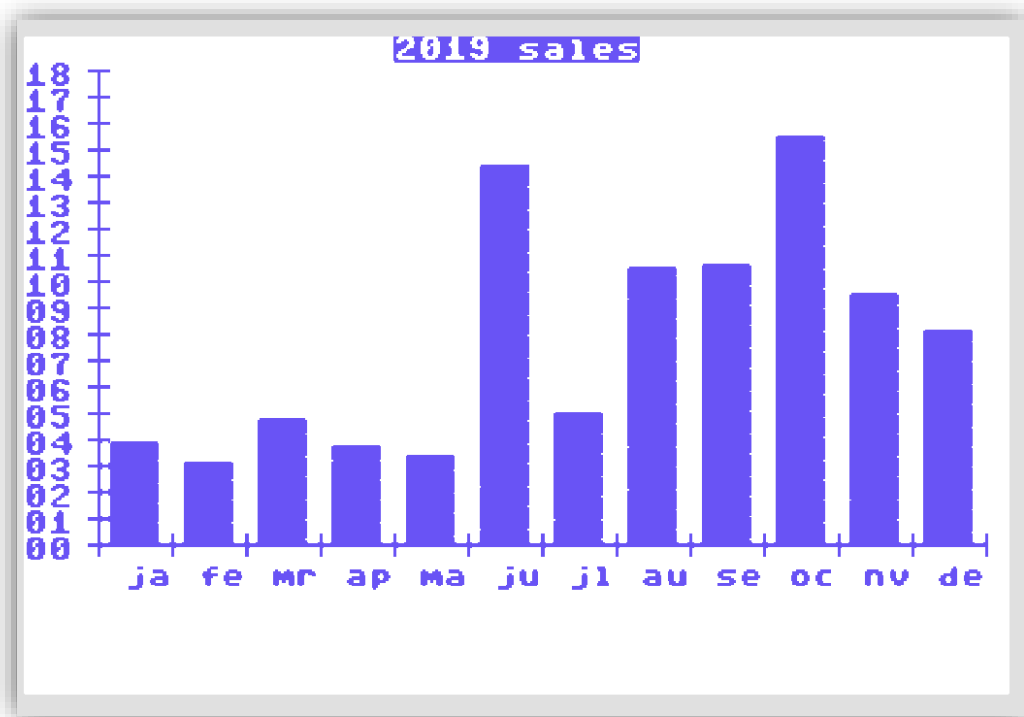
```
10 INIT : BGND WHITE : FGND BLUE
20 HORDER WHITE: SCLR 0,ATR : HIRES
```

LASER BASIC FOR THE COMMODORE 64

```
30 DRAW 0,24 154,312,154
40 FOR I=0 TO 12
50 DRAW 0,24+I*24,151,24+I*24,157
60 NEXT I
70 TEXT 0,3,20," JA FE MR AP MA JU JL AU SE OC NV DC",0
80 DRAW 0,24,10,24,154
90 FOR I=0 TO 18
100 DRAW 0,21,10+I*8,27,10+I*8
110 TEXT 0,0,1+I,RIGHTS(STRS (118-I),2),0
120 NEXT I
130 FOR J=0 TO 11
140 N = RND(1) * 18
150 BOX 0,28+J*24,155-8*N,16,N*8
160 NEXT J
170 TEXT 0,15,0,"2019 SALES",256
180 GOTO 180
```

Available on disk as "ex charset 05"

The horizontal axis is drawn in line 30, and the scale is added in lines 40 to 60. The TEXT command in line 70 labels the axis. The vertical axis is drawn in line 80 and the scale and labelling is added in lines 100 and 110. The loop from line 130 to 160 draws the bars that make up the graph. Variable N in line 140 is the figure for each month. Finally, the whole graph is given its title in reverse video line 170.



REDEFINING A CHARACTER

(PUTCHAR)

LASER BASIC FOR THE COMMODORE 64

The PUTCHR command can be thought of as the opposite of the CHAR command instead of copying a character from the character set into a sprite, it redefines a character by copying it from a sprite into the character set. The command has four parameters:

PUTCHR SPN, COL, ROW, NUM

The character to be redefined is copied from sprite no. SPN at position COL, ROW. COL and ROW are both measured in character blocks. NUM is the character to be redefined - this obeys the same rules as it did with CHAR, although double-width characters do not apply:

ASCII - NUM = ASCII code
 reversed ASCII - NUM = ASCII code + 256
 display code - NUM = display code + 512

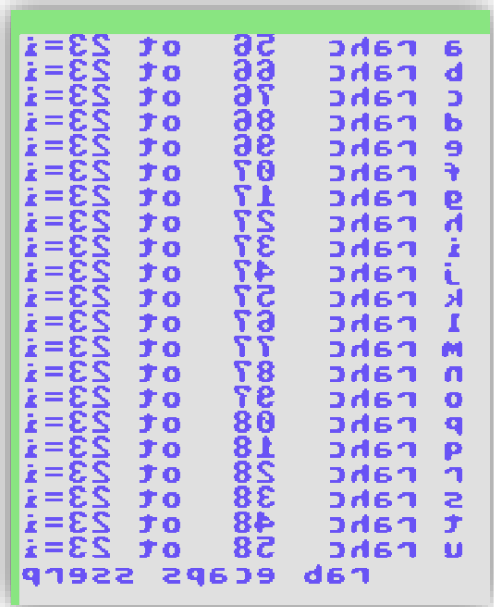
Example: This program reverses the entire character set:

```

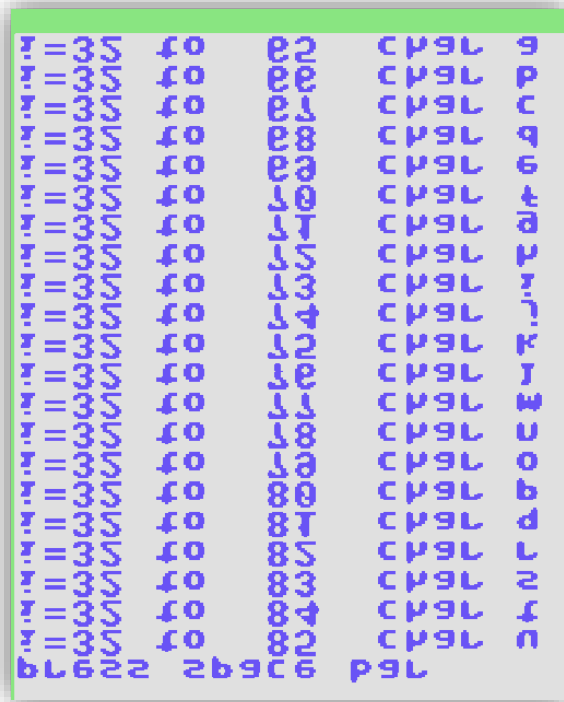
10 IF DFA(254) < 0 SPRITE 254,1,1
20 FOR I=512 TO 767
30 CHAR 254,0,0,I
40 MIR 254,0,0,1,1
50 PUTCHR 254,0,0,I
60 NEXT I
    
```

Available on disk as "ex charset 06"

Sprite no. 254 is used to hold each character while it is reversed Each character is read into the sprite using CHAR, reversed using MIR and then copied back into the character set again using the PUTCHR command. If the MIR command in line 40 is changed to FLIP, the program turns all the characters upside-down.



LASER BASIC FOR THE COMMODORE 64



Example: This procedure redefines the entire character set using the contents of a 16x16 sprite.

```
10 LABEL DOWNLOAD (S)
20 IF DFA(S) < 0 OR WID <> 16 OR HGT <> 16 STOP
30 FOR I=0 TO 15
40   FOR J=0 TO 15
50     PUTCHR S,I,J,512+I+J*16
60   NEXT J
70 NEXT I
70 PROCEND
```

This procedure is very similar to the example given in section [PUTTING A CHARACTER INTO A SPRITE \(CHAR\)](#) . In fact, the only difference is the use of PUTCHR in line 50 Instead of CHAR.

LASER BASIC FOR THE COMMODORE 64

HARDWARE SPRITES

The Commodore 64 has eight very powerful hardware sprites which can be used from Laser BASIC. Hardware sprites differ from software sprites in several respects:

- I. Each hardware sprite can only be displayed at one position on the screen, unlike software sprites which can be put on the screen as many times as you like.
- II. The hardware sprites are separate from the hires screen's pixel data and can be moved around the screen easily without affecting the pixel data.
- III. Each hardware sprite can be in either two-colour or four-colour mode, independent of the screen display mode
- IV. Hardware sprites are of fixed size - 24 pixels wide and 21 pixels high - although it is possible to display them double-size.
- V. Laser BASIC has commands which can move hardware sprites around the screen automatically under interrupt.

In order to display a hardware sprite on the screen, you must do the following:

- I. Create a hardware sprite definition, using part of a software sprite.
- II. "Tell" one of the eight hardware sprites to use the sprite definition.
- III. Give the sprite a colour.
- IV. Turn the sprite on.
- V. Position it on the screen.

These five stages are discussed in detail below. Remember that in order to run any of the example programs in this section, the arcade sprite library must be loaded.

CREATING A HARDWARE SPRITE DEFINITION

(CONV)

Before a hardware sprite can be displayed, you must create a hardware sprite definition for it from the software sprite by using the CONV command. Up to 32 hardware sprite definitions can exist at once and are numbered from 0 to 31. Due to memory constraints, the sprite definitions share memory with the character set. Each sprite definition takes up the same space as eight-character definitions:

SPRITE DEFINITION NUMBER	CHARACTER DEFINITION NUMBER (DISPLAY CODE)
0	0 to 7
1	8 to 15
2	16 to 23
3	24 to 31
4	32 to 39
N	$N*8$ to $N*8+7$
29	235 to 239
30	240 to 247
31	248 to 255

LASER BASIC FOR THE COMMODORE 64

If you set up a hardware sprite definition, the corresponding characters will appear on the screen as garbage. For this reason, it is best to use only sprite definitions 16 to 31. These correspond to the reverse character set which is only used to blink the cursor. The CONV command is used to actually create a sprite definition. It takes a rectangular block 24 pixels wide and 21 pixels high out of a hires sprite and copies it into one of the hardware sprite definitions. It has four parameters:

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

The sprite definition is taken from sprite no. SPN at position COL,ROW and copied into hardware sprite definition no. SPN2. COL and ROW are both measured in pixels. The rectangular block used to create the definition must be entirely within the software sprite, or garbage will be displayed when the hardware sprite appears on the screen.

ASSOCIATING A HARDWARE SPRITE WITH ITS DEFINITION

(HSET)

Once a hardware sprite definition has been created, one of the eight hardware sprites must be associated with the definition in order to display it. This is done by using the HSET command:

```
HSET <sprite number>.<definition number>
```

The sprite number is from 0 to 7, and the definition number is from 0 to 31. Using this command, it is possible to have several hardware sprites using the same definition, or to achieve animation effects by changing between several sprite definitions while a sprite is moving.

SETTING UP A HARDWARE SPRITE'S COLOUR

(HCOL)

Each hires sprite has its own colour which is independent of the screen attributes or the other hires sprites. It is set up with the HCOL command.

```
HCOL <sprite number>,<colour>
```

The sprite number lies between 0 and 7. Generally,<colour> is one of the colour constants described in section [DISPLAYING EXTENDED BACKGROUND COLOUR MODE \(EBACK\)](#).

SWITCHING A HARDWARE SPRITE ON AND OFF

(HON, HOFF)

Initially, all hardware sprites are switched off. To make a sprite appear, it must be turned on by using HON:

```
HON <sprite number>
```

There is a similar command, HOFF, to turn a sprite off again:

```
HOFF <sprite number>
```

As usual, the sprites are numbered from 0 to 7.

POSITIONING A HARDWARE SPRITE ON THE SCREEN

(HX, HY)

LASER BASIC FOR THE COMMODORE 64

Once a hardware sprite has been defined, given a colour and turned on, it will not be visible. This is because it is positioned off the top left hand corner of the screen and it must be moved into the visible screen area.

Horizontal Positioning

A hardware sprite is positioned horizontally, at pixel resolution, with the HX command:

```
HX <sprite number>,<position>
```

A sprite can be placed outside the screen area, so that only part of it can be seen:

HX	Visibility
HX = 0	Invisible
1 =< HX =< 23	Partly visible
24 =< HX =< 320	Visible
321 =< HX =< 343	Partly visible
344 =< HX =< 511	Invisible

Vertical Positioning

A sprite can also be positioned vertically, using the HY command:

```
HY <sprite number>.<position>
```

A sprite can be placed off the screen vertically also:

HY	Visibility
0 =< HY =< 29	Invisible
30 =< HY =< 49	Partly visible
50 =< HY =< 229	Visible
230 =< HY =< 249	Partly visible
250 =< HY =< 255	Invisible

Reading a Sprite's Position

HX and HY can also be used as functions within an expression, to find a hardware sprite's current position on the screen. The format is:

```
HX (<sprite number>>)
```

and

```
HY (<sprite number>>)
```

Example: This program shows how HX can be used to make a hardware sprite partly visible. A sprite is moved from the extreme left to the extreme right of the screen, with the X co-ordinate being displayed

```
10 INIT : SCLR 0,ATR : INV 0,0,0,40,25
20 CONV 0,0,0,31 : HSET 0,31
30 HCOL 0,RED : HY 0,100 : HON 0
```

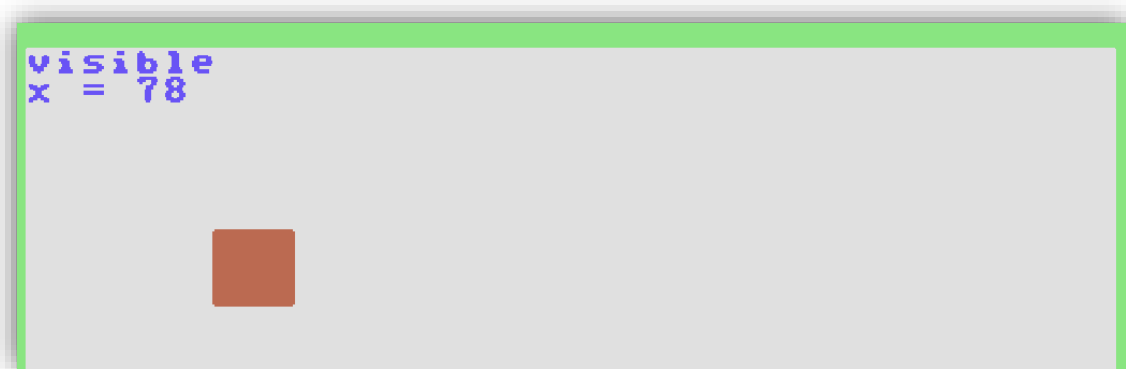
LASER BASIC FOR THE COMMODORE 64

```
40 PROC DISP("INVISIBLE",0,0)
50 PROC DISP("PARTLY VISIBLE",1,23)
60 PROC DISP("VISIBLE",24,320)
70 PROC DISP("PARTLY VISIBLE",321,343)
80 PROC DISP("VISIBLE",344,511)
90 HOFF 0 : END
100 LABEL DISP(N$,A,B)
110 LOCAL I, K$
120 PRINT CHR$(147);NS
130 FOR I=A TO B
140 HX 0,I
150 PRINT "X =" ; I
160 PRINT CHR$(145);
170 NEXT I : PRINT: PRINT "PRESS SPACE"
180 REPEAT : GET KS : UNTIL K$=" "
190 PROCEND
```

Available on disk as "ex hsprite 01"

The first three lines of the program set up hardware sprite no 0. The whole hires screen is set to foreground pixels, by using SCLR, followed by INV. The top left hand corner of the screen is then converted to hardware sprite definition no. 31, which is used for sprite no. 0. Therefore, when the sprite is displayed on the screen it appears as a solid block.

The procedure PROCdisp moves the sprite between two positions on the screen, printing the X coordinates at the top left hand corner of the screen. It would be fairly simple to change the program so that the sprite moves vertically instead of horizontally, and you might like to try doing this. Remember to change the numbers used in the calls to PROCdisp (lines 40 to 80).



Example: This program shows how to move a hardware sprite around the screen under control of a joystick plugged into control port 1.

```
10 FOR I=0 TO 8 : READ DX(I) : NEXT I
20 FOR I=0 TO 8 : READ DY(I) : NEXT I
30 INIT : SCLR 0,0 : PUTBLK 39,0,0
40 CONV 0,0,0,31 : HSET 0,31 : HON 0 : S = 1
50 X=100 : Y=100 : HCOL 0,BLACK
60 REPEAT
```

LASER BASIC FOR THE COMMODORE 64

```
70 IF FIRE1 AND S<10 THEN S=S+1 ELSE IF S > 1 S=S-1
80 X=X + DX(JS1) * S : Y=Y + DY(JS1) * S
90 IF X < 24 THEN X=24
100 IF Y < 50 THEN Y=50
110 IF X > 320 THEN X=320
120 IF Y > 229 THEN Y=229
130 HX 0,X : HY 0,Y
140 UNTIL FALSE
150 DATA 0,1, 1, 0,-1,-1,-1,0,1
160 DATA 0,0,-1,-1,-1, 0, 1,1,1
```

Available on disk as "ex hsprite 02"

For each position of the joystick, the arrays dx and dy hold the number of pixels that the sprite must be moved by, horizontally and vertically respectively. Data is read into these arrays in lines 10 and 20. Software sprite no. 39 is read into hardware sprite definition no. 31, but it is too small to be converted directly. The sprite is only 2 characters, or 16 pixels, high and CONV needs a rectangular block 21 pixels high. To get around this problem, the hires screen is cleared and sprite no. 39 is PUT onto the screen. CONV is now used to copy the sprite from the screen (sprite 0) into the hardware sprite definition.

The variable s is the speed at which the sprite moves: x and y are the horizontal and vertical positions of the sprite. Line 70 adjusts s: if the fire button is pressed, s is increased up to a maximum of 10.

Otherwise s decreases to 1. The new position of the sprite is calculated in line 80. Lines 80 to 120 ensure that the sprite will remain on the screen, and line 130 puts the sprite in its new position.



Example: This program shows how two hardware sprites can be used as one large sprite.

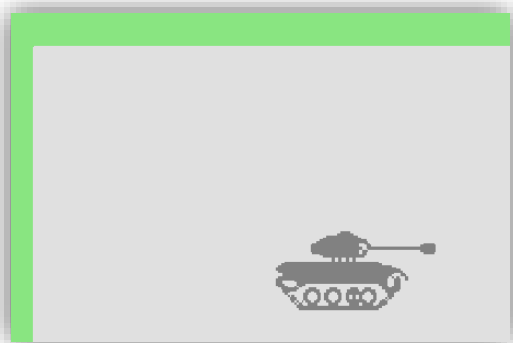
```
10 INIT
20 CONV 28, 0,3,30
30 CONV 28,24,3,31
40 HSET 0,30 : HSET 1,31
50 HCOL 0,GRAY1 : HCOL 1,GRAY1
60 HON 0 : HON 1
70 HY 0,100 : HY 1,100
80 FOR X=24 TO 296 STEP 2
90 HX 0,X : HX 1,X+24
100 NEXT X
```

LASER BASIC FOR THE COMMODORE 64

Available on disk as "ex hsprite 03"

The graphic to be displayed, a tank, is in software sprite no. 28. This sprite is 6 characters long and 3 characters high. In lines 20 and 30, the left half is put in hardware sprite definition no. 30, and the right half is put in definition no. 31.

Note that in line 90, the right half of the tank (hardware sprite no. 1) must be placed 24 pixels to the right of the left half (sprite no. 1).



EXPANDED HARDWARE SPRITES

(HEXX, HEXY, HSHX, HSHY)

Hardware sprites can be expanded to double size either horizontally or vertically. In an expanded hardware sprite, each pixel is twice the size - the resolution does not increase; the sprite simply gets bigger.

Laser BASIC has two commands for expanding hardware sprites:

HEXX <sprite no.> Display hardware sprite at double width

HEXY <sprite no.> Display hardware sprite at double height

There are two further commands to shrink a hardware sprite back to normal size:

HSHX <sprite no.> Display hardware sprite at normal width.

HSHY <sprite no.> Display hardware sprite at normal height.

An expanded hardware sprite is visible or partly visible on the screen for the following co-ordinate values:

HX	Visibility (expanded sprite)
481 =< HX =< 503	Invisible
0 =< HX =< 23	Partly visible
24 =< HX =< 296	Visible
297 =< HX =< 343	Partly visible
344 =< HX =< 480	Invisible

LASER BASIC FOR THE COMMODORE 64

HY	Visibility (expanded sprite)
0 =< HY =< 8	Invisible
9 =< HY =< 49	Partly visible
50 =< HY =< 208	Visible
209 =< HY =< 249	Partly visible
250 =< HY =< 255	Invisible

Placing an expanded sprite partly off the left hand side of the screen is not as straightforward as with normal sized sprites. To put an expanded sprite at the extreme left of the screen, with only the right hand edge of it showing, you must use an x co-ordinate of 481. Increasing this to 503 will move the sprite to the right. After 503, you must use an x co-ordinate of 0 to move the sprite right by one pixel.

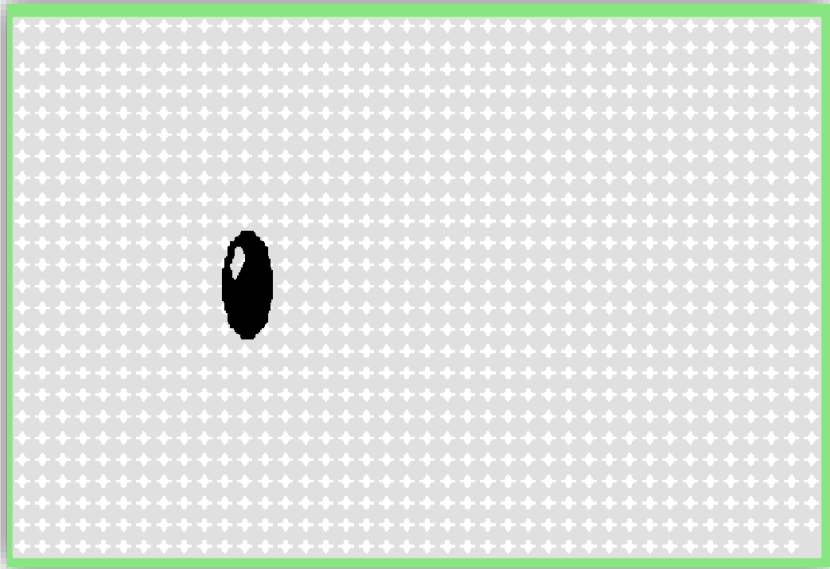
Example: This program displays a bouncing ball in front of a screenful of plus signs. The ball is displayed in both normal size and expanded versions.

```
10 INIT : INK WHITE
20 FOR I=1 TO 999 : PRINT "+"; : NEXT I
30 CONV 44,0,2,31
40 HSET 0,31
45 HCOL 0,BLACK
50 FOR EX=0 TO 1
60 IF EX=1 HEXX 0 ELSE HSHX 0
70 FOR EY=0 TO 1
80 IF EY=1 HEXY 0 : K=210 ELSE HSHY 0 : K=230
90 HON 0 : V=0 : Y=30
100 FOR X=1 TO 344
110 V=V + 0.33 : Y=Y+V
120 IF Y >= K THEN V = -V * 0.85 : Y=K
130 RASTER 51 : HX 0,X : HY 0,Y
140 NEXT X
150 HOFF 0
160 NEXT EY
170 NEXT EX
180 INK BLUE
```

Available on disk as "ex hsprite 04"

The + signs are placed on the screen in line 20, and hardware sprite no. 0 is set up in lines 30 and 40. When ex is set to 1, the sprite is expanded horizontally. Similarly, when ey=1, it is expanded vertically. Variable k is set to the sprite's lowest visible position on the screen - this depends on whether it is expanded or not. The sprite is moved around the screen in the FOR-NEXT loop from line 100 to 140. V is the downwards velocity of the ball, and it increases as the ball falls. In line 120, the new position, y, is checked to see if the ball has moved off the screen. If it has, the downward velocity is reversed to make the ball bounce back. The RASTER command is used in line 130 to ensure that the horizontal and vertical positions of the sprite change instantaneously. This is because the Laser BASIC software always updates a sprite's position when the display scan is at line 0.

LASER BASIC FOR THE COMMODORE 64



HARDWARE SPRITE DISPLAY PRIORITIES

(OVER, UNDER)

Hardware sprites can be used to create a 3-dimensional effect, since they can be made to pass in front of or behind each other.

Low-numbered sprites have priority over high-numbered sprites - i.e. sprite 0 has the highest priority and sprite 7 has the lowest priority. This means that if sprite 0 and sprite 1 occupied the same position on the screen, sprite 0 would appear to be in front of sprite 1 because it has higher priority

Because pixels in a sprite which are not set are transparent, a window effect is possible with the screen background or other sprites showing through "holes" in a sprite.

Example: This program shows how sprite display priorities work:

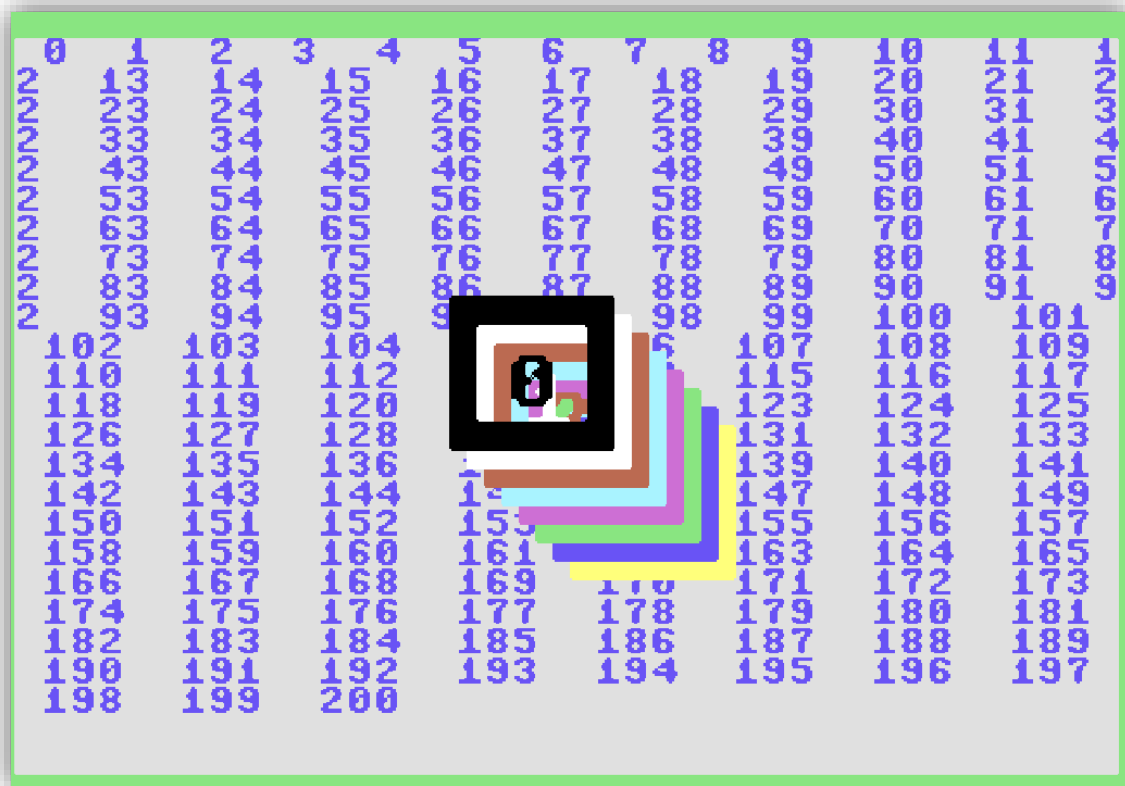
```
10 INIT
20 FOR I=7 TO 0 STEP -1
30 WCLR 0,0,0,3,3,1
35 INV
40 MODE 0
45 BOX 0,4,4,16,13
50 CHAR 0,1,1,I+48
60 CONV 0,0,0,24+I
70 HSET I,24+I
80 HCOL I,I
90 HX I,150-I * 10
100 HY I,150-I * 10
110 HEXX I
105 HEXY I
120 HON I
```

LASER BASIC FOR THE COMMODORE 64

```
130 NEXT I
140 FOR I = 0 TO 200
143 PRINT I;
146 NEXT I
150 FOR I = -9 TO 10
160 FOR K=0 TO 1000
165 NEXT K
170 FOR J=0 TO 7
180 HX J,150 + I * J
190 HY J,120 + I * J
200 NEXT J
210 NEXT I
220 END
```

Available on disk as "ex hsprite 05"

Lines 20 to 130 set up the eight sprites. Each sprite is a rectangle with its number inside it, and is displayed double-size. A different colour is given to each sprite. Line 140 places some numbers on the screen. Line 50 onwards moves the sprites over the screen. As they move, you should be able to see how the lowest numbered sprites appear to be above the highest numbered ones.



Sprite to Screen Display Priorities

Normally, hardware sprites pass over the hires or text screen background. However, the screen can be given priority over any sprite with the OVER command:

LASER BASIC FOR THE COMMODORE 64

OVER <sprite number>

To revert to the screen going under a sprite, the UNDER command is used:

UNDER <sprite number>

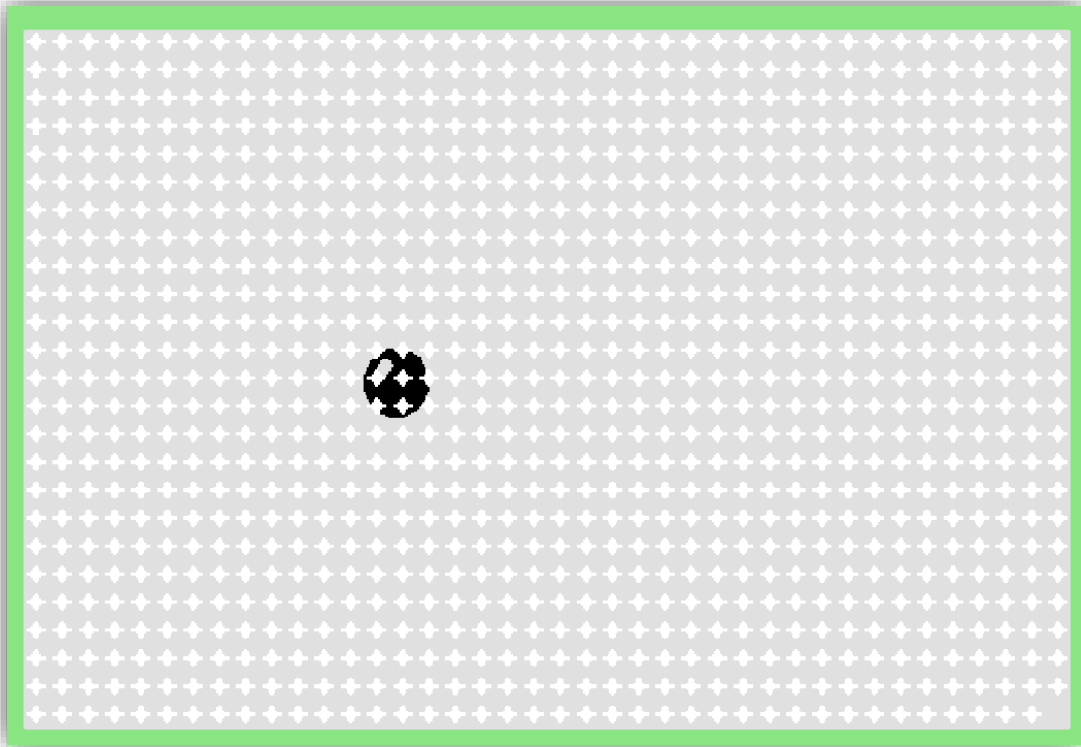
Example: This is a modified version of the 'bouncing ball' program given earlier. The ball is displayed twice - once in front of the screen and once behind it.

```
10 INIT : INK WHITE
20 FOR I=1 TO 444 : PRINT "+++"; : NEXT I
30 CONV 44,0,2,31
40 HSET 0,31 : HCOL 0,BLACK
50 FOR P=0 TO 1
60 IF P=0 UNDER 0 ELSE OVER 0
70 HON 0 : V=0 : Y=30
80 FOR X=1 TO 344
90 V=V + 0.33 : Y=Y+V
100 IF Y >= 230 THEN V= -V * 0.85 : Y=230
110 RASTER 51 : HX 0,X : HY 0,Y
120 NEXT X
130 HOFF 0
140 NEXT P
150 INK BLUE
```

Available on disk as "ex hsprite 06"

The important line is line 60 - the first time, when p=0, the UNDER command places the screen under sprite 0, and the second time, OVER places the screen over sprite 0.

LASER BASIC FOR THE COMMODORE 64



MOVING HARDWARE SPRITES AUTOMATICALLY

(MOVE)

Laser BASIC allows you to have hardware sprites moving around the screen automatically. Your program can send a sprite moving with a given direction and speed, and the sprite will travel over the screen while your program does something else. This is done with the MOVE command:

MOVE SPN, COL, ROW

This sets hardware sprite no. SPN moving. Every 50th of a second, the sprite will be moved right by COL pixels and down by ROW pixels. The sprite can be sent moving up or left by using negative numbers for COL and ROW. The sprite will keep moving until:

- a) You turn it off with the HOFF command.
- b) It goes off the screen. In this case, the sprite is automatically turned off by Laser BASIC.
- c) You stop it moving by using another MOVE command, with COL and ROW both zero (for example, MOVE 2,0,0).

Remember that HX(<sprite number>) and HY(<sprite number>) can be used to tell where a sprite is on the screen.

Example: This program uses the MOVE command to make a sprite travel over the screen in eight different directions:

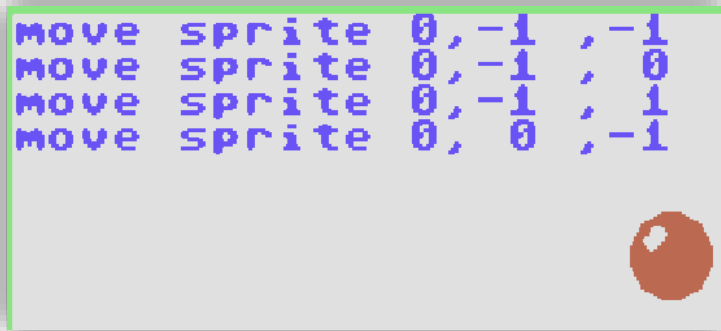
```
10 INIT
15 CONV 44,0,2,31
```

LASER BASIC FOR THE COMMODORE 64

```
20 HSET 0,31
25 HCOL 0, RED
30 FOR X=-1 TO 1
40   FOR Y=-1 TO 1 STEP 2 - ABS(X)
50     PRINT "MOVE 0, "; X; ". "; Y
60     HX 0,172
65     HY 0,139
70     HON 0
75     MOVE 0,X,Y
88     REPEAT
90     UNTIL HX(0)=0 OR HX(0) > 342 OR HY(0) < 29 OR HY(0) > 250
100    HOFF 0
100   NEXT Y
110  NEXT X
```

Available on disk as "ex hsprite 07"

The first two lines of the program define hardware sprite no. 0. The variables x and y are the amount by which the sprite will be Moved horizontally and vertically. The "STEP 2-ABS(x)" is included so that x and y cannot both be zero. Line 50 prints up the current direction on the screen. The sprite is placed at the centre of the screen in line 60, and MOVE is used to start it moving in line 70. The REPEAT -UNTIL loop r lines 80 and 90 waits until the sprite has gone off the screen.



Example: This program moves four copies of sprite no. 4 over the hires screen, both above and below the screen's pixel data. The sprites bounce off the edges of the screen area:

```
10 INIT : BGND GRAY2 : FGND BLACK
20 HBORDER GRAY2
30 SCLR 0,ATR : HIRES : MODE4
40 FOR I = 1 TO 50
50   BOX 0,8 + RND(1) * 254,8 + RND(1) * 134,20 + RND(1) * 30,20
+ RND(1) * 30
60 NEXT I
65 CONV 4,0,0,31
70 FOR I=0 TO 3
80   HSET I,31
85   HON I
```

LASER BASIC FOR THE COMMODORE 64

```
90  HX I,50 + RND(1) * 100
100 HY I,50 + RND(1) * 100
110 HCOL I,I + 5
113 HEXX I
116 HEXY I
120 MX(I) = 2 * (I AND 1) - 1
130 MY(I) =      (I AND 2) - 1
140 IF I > 1 OVER I
150 NEXT I
160 REPEAT
165  FOR I=0 TO 3
170   MOVE I, MX(I),MY(I)
180   IF HX(I) < 32 THEN MX(I) = 1 : MOVE I, 1,MY(I)
190   IF HX(I) > 288 THEN MX(I) = -1 : MOVE I, -1,MY(I)
208   IF HY(I) < 58 THEN MY(I) = 1 : MOVE I, MX(I), 1
210   IF HY(I) > 200 THEN MY(I) = -1 : MOVE I, MX(I), -1
220  NEXT I
225 UNTIL FALSE
230 END
```

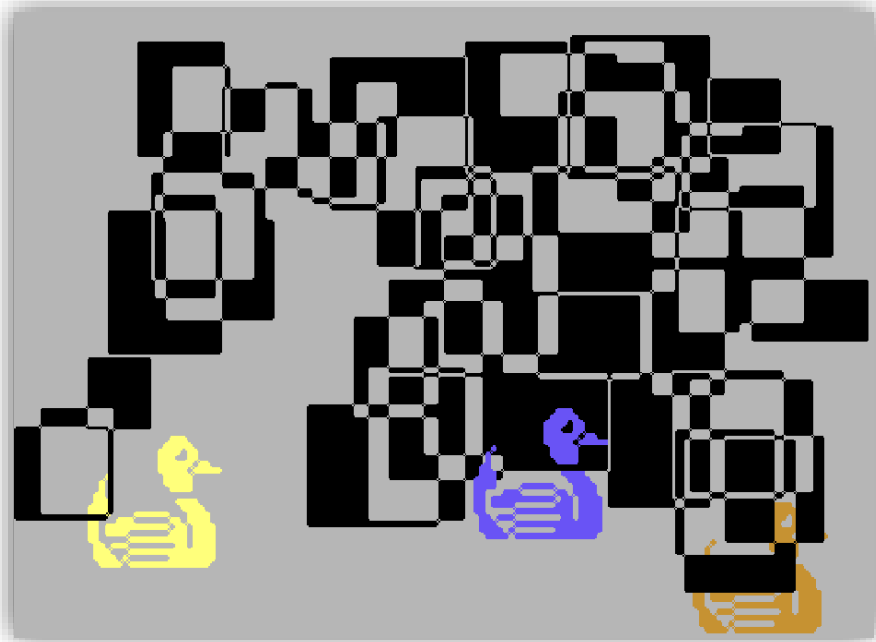
Available on disk as "ex hsprite 08"

The first six lines of this program place a pattern on the screen which consists of 50 randomly placed rectangles. The BOX command in line 50 draws the rectangles - note that MODE 4 is used, so that BOX inverts the screen pixels.

In line 60, software sprite no. 4 is converted into a hardware sprite definition. From line 50 to 140, hardware sprites 0 to 3 are set up. Note that they all use the same sprite definition. All the sprites are expanded. The arrays, mx and my, hold the horizontal and vertical speeds of each sprite which will be used by the MOVE command - these are always either 1 or -1. In line 140 sprites 2 and 3 are placed under the hires screen's pixel data.

The loop from line 160 onwards continuously checks the position of each sprite on the screen and sends it back in the opposite direction if it is near the edge of the screen. For example, line 180 checks to see if sprite no. 1 is near the left hand edge of the screen. If it is, its x-velocity is set to 1 so that it travels right instead of left and does not go off the screen.

LASER BASIC FOR THE COMMODORE 64



TRACKING SPRITES

(TRACK)

For some applications, the MOVE command is too limited. For example, you might want a sprite to appear at the top left of the screen, go backwards and forwards at the top of the screen twice, and then fall to the bottom of the screen. If this were done using MOVE, your program would constantly have to check the sprite's position and change its direction if necessary. Laser BASIC's TRACK command can do all of this automatically. It has two parameters:

TRACK SPN, SPN2

SPN2 is the hardware sprite which is to be moved around the screen. As usual, it is numbered from 0 to 7.

Software sprite no. SPN holds the path that the hardware sprite is to follow in its data field. The path is held as a list of numbers from -127 to +127, each one byte long. Every 50th of a second, a pair of numbers is read from the sprite by Laser BASIC. The first is added to the hardware sprite's horizontal position, and the second is added to its vertical position. The sprite continues to move around the screen in this way, until the end of the sprite's data field is reached.

To make the sprite stay stationary on the screen afterwards, the last pair of numbers in the software sprite should be 0,0. Remember that the hardware sprite is moved automatically, while your program continues to run.

The tracking sprite data is normally put into the software sprite with the POKE command. It is then stored on tape or disk along with any other sprites used in your program.

LASER BASIC FOR THE COMMODORE 64

As mentioned earlier, the numbers can range from -127 to 127. Positive numbers are simply POKEd into the sprite; negative numbers must have 256 added to them first. To determine the number of character blocks which are required in the sprite, divide the number of number pairs by four, and round up to the nearest integer.

Here is a short program which reads the data for the TRACK command from a set of data statements and places it in a software sprite of the correct size.

```
10 SN=200 'SPRITE NUMBER
20 IF DFA(SN) > 0 WIPE
30 REPEAT : N=N+1 : READ AS : UNTIL AS = "."
40 N=N-1 : IF N AND 1 STOP
50 K = INT(N+6) / 8 : I = INT(SQR(K))
60 REPEAT
70 IF K/I = INT(K/I) AND K/I < 256 EXIT
80 I=I-1
85 UNTIL I=0
90 IF I=0 : K=K+1 : GOTO 50
100 SPRITE SN,K/I,I : M=DFA(SN) : RESTORE
110 FOR C=1 TO N : READ J
120 IF J<0 POKE M,J+256 ELSE POKE M,J
130 M=M+1
140 NEXT C
150 SCLR 0,1 : INV 0,0,0,3,3
160 CONV 0,0,0,31 : HSET 0,31
170 HCOL 0,RED : HON 0 : HX 0,100 : HY 0,100
180 TRACK SN,0
190 `
200 DATA 3 , 3
210 DATA 3 , 3
220 DATA 3 , 3
230 DATA 3 , 3
240 DATA 3 , 3
250 DATA 3 , 3
260 DATA 3 , 3
270 DATA 3 , 3
280 DATA 3 , -3
290 DATA 3 , -3
300 DATA 3 , -3
310 DATA 3 , -3
320 DATA 3 , -3
330 DATA 3 , -3
340 DATA 3 , -3
350 DATA 3 , -3
360 DATA -3 ,-3
370 DATA -3 ,-3
380 DATA -3 ,-3
390 DATA -3 ,-3
```

LASER BASIC FOR THE COMMODORE 64

```
400 DATA -3 , -3
410 DATA -3 , -3
420 DATA -3 , -3
430 DATA -3 , -3
440 DATA -3 , 3
450 DATA -3 , 3
460 DATA -3 , 3
470 DATA -3 , 3
480 DATA -3 , 3
490 DATA -3 , 3
500 DATA -3 , 3
510 DATA -3 , 3
520 DATA 0 , 0
530 DATA .
```

Available on disk as "ex hsprite 09"

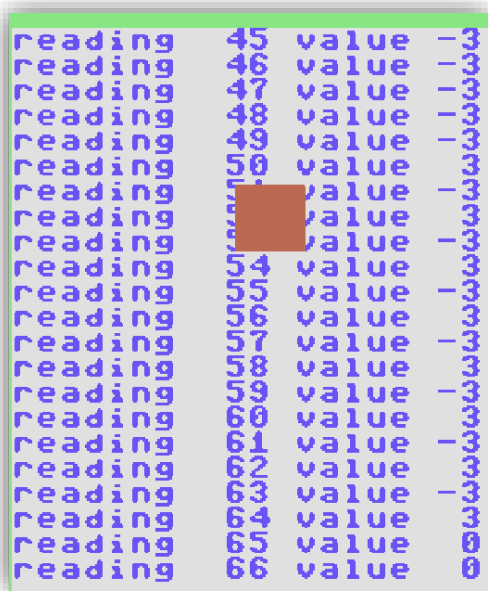
Each DATA statement from line 200 onwards contains the number of pixels that the hardware sprite is moved right and down. At the end of the list, there must be a DATA statement with a full stop after it. In its present form the program uses software sprite no. 200, but this can easily be changed in line 10.

After the software sprite has been created, the program sets up hardware sprite no. 0 as a solid block and uses TRACK to move it around the screen.

Its initial position can be changed in line 170.

IMPORTANT:

When a tracking sprite is moving under interrupt, you must not delete any software sprites with the WIPE command.



LASER BASIC FOR THE COMMODORE 64

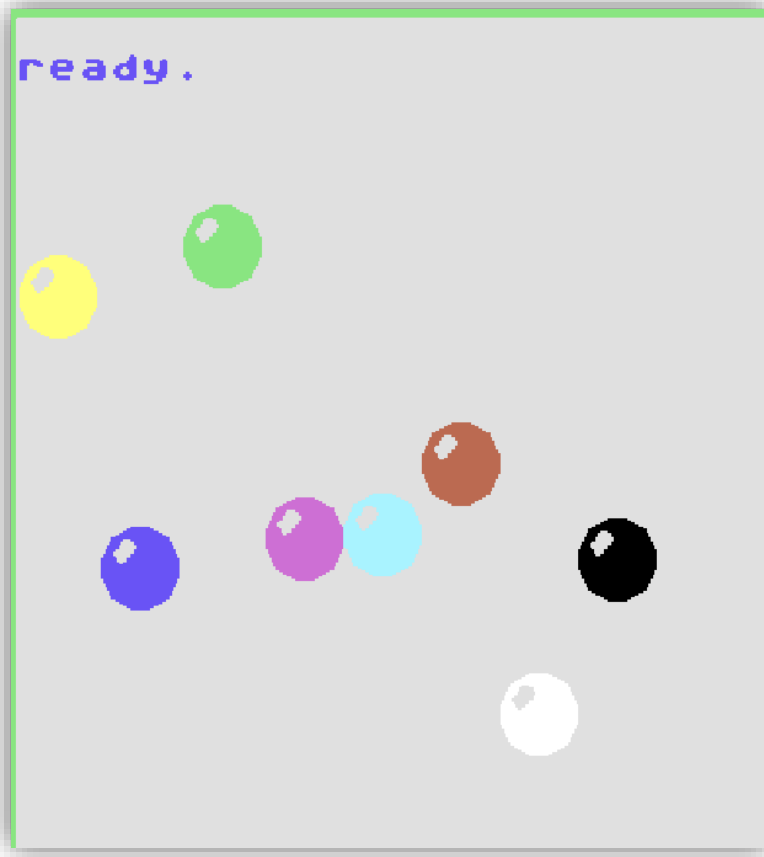
Example: This is a modified version of the 'bouncing ball' program given earlier. In this program, the trajectory is stored inside sprite no. 200 and the TRACK command is used to make all 8 sprites bounce across the screen one after another.

```
10 IF DFA(200) > 0 WIPE
20 SPRITE 200,86,1
25 N=DFA(200)
30 V=0
35 Y=330
40 FOR X=1 TO 344
50 OY=Y
60 V=V + 0.33
65 Y=Y + INT(V)
70 POKE N,1
80 IF Y >= 230 THEN V=-V * .75 : Y=230
90 DY=INT(Y - OY + 0.5)
100 IF DY < 0 THEN POKE N+1, DY+256 ELSE POKE N+1,DY
110 N=N+2
120 NEXT X
130 '
140 INIT
145 CONV 44,0,2,31
150 FOR I=0 TO 7
160 HCOL I,I
165 HSET I,31
170 HX I,0
173 HY I,30
176 HON I
180 CIF I > 0
190 REPEAT
200 UNTIL HX(I-1) >= 20
210 CEND
220 TRACK 200,I
230 NEXT I
```

Available on disk as "ex hsprite 10"

The first two lines of the program set up sprite no. 200. Lines 40 to 120 calculate the trajectory, using the same algorithm as the original 'bouncing ball' program. Variable oy is the previous vertical position, and it is used to calculate dy, the change in vertical position. is the current location in sprite no. 200. From line 140 onwards, the eight sprites are displayed, each using sprite definition no. 31. In line 180, the program waits until the previous sprite is at least 20 pixels out from the left of the screen.

LASER BASIC FOR THE COMMODORE 64



Using TRACK as a Function

TRACK can be used as a function to determine if a sprite is still being moved around the screen. The format is:

```
TRACK <sprite number>
```

If the sprite is still tracking, the function returns a value of -1 (TRUE). Otherwise, it returns zero (FALSE).

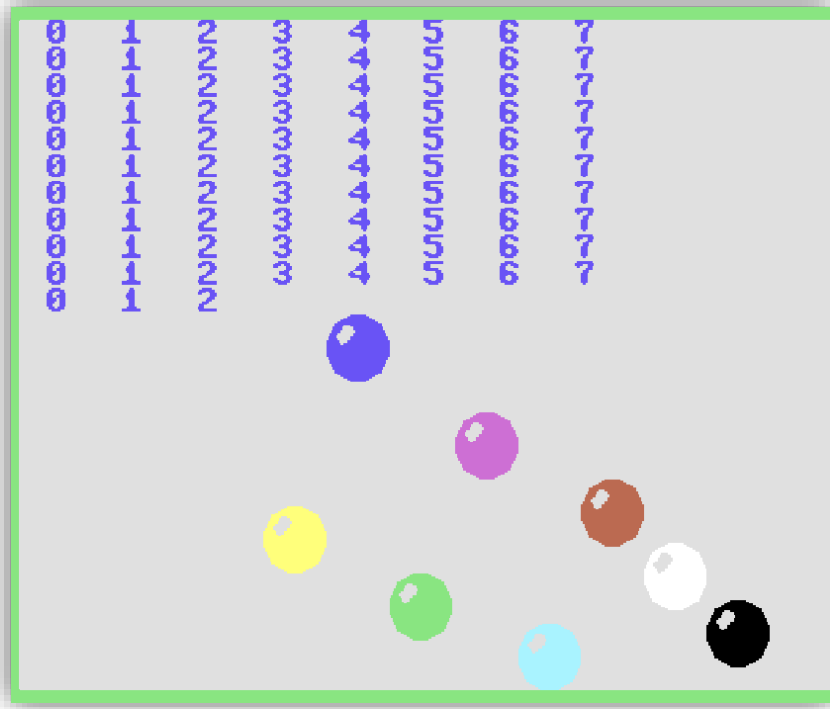
Example: Add the following lines onto the end of the previous example program:

```
210 `
220 REPEAT
230   FOR I=0 TO 7
240     IF TRACK(1) THEN PRINT I;
250   NEXT I
260   PRINT
270 UNTIL NOT TRACK (7)
280 END
```

Available on disk as "ex hsprite 10"

After all the sprites have been set moving across the screen, the program prints out the numbers of all sprites that are still tracking.

LASER BASIC FOR THE COMMODORE 64



FOUR-COLOUR MODE HARDWARE SPRITES

(H4COL, H2COL, H1COL, H3COL)

Each of the eight hardware sprites can be displayed in ordinary two-colour mode or four-colour mode, independent of the screen mode or the other sprites. The pixels in a four-colour mode hardware sprite are twice as wide as they are in two-colour mode and the sprite is only 12 pixels wide instead of 24.

The H4COL command puts a hardware sprite into four-colour mode:

```
H4COL <sprite number>
```

The sprite number is between 0 and 7. To put a hardware sprite back into ordinary two-colour mode, use the H2COL command:

```
H2COL < sprite number>
```

In two-colour mode, each pixel in a hardware sprite can be either transparent or coloured:

SOFTWARE SPRITE (BEFORE CONV)	HARDWARE SPRITE (AFTER CONV)
BACKGROUND	TRANSPARENT
FOREGROUND	SPRITE COLOUR (set by HCOL <sprite number> .<colour>)

In four-colour mode, each pixel can be one of four colours, hence two additional colours are required. These are the same for all eight hardware sprites, and are set up with the H1 COL and H3COL commands:

LASER BASIC FOR THE COMMODORE 64

H1COL < colour>

H3COL < colour>

The following table shows the correspondence between colours in four-colour software sprites and hardware sprites:

SOFTWARE SPRITE (BEFORE CONV)	HARDWARE SPRITE (AFTER CONV)
COLOUR 0 (BACKGROUND)	TRANSPARENT
COLOUR 1	SPRITE COLOUR NO.1 (set by H1 COL <colour>
COLOUR 2	SPRITE COLOUR NO.2 (set by HCOL <sprite no.>,<colour>)
COLOUR 3	SPRITE COLOUR NO.3 (set by H3COL <colour>)

Four-colour mode hardware sprites can be expanded with the HEXX and HEXY commands in the same way as two-colour sprites.

COLLISION DETECTION WITH HARDWARE SPRITES

(HIT)

Like software sprites, hardware sprites have a collision detection facility. This can detect collisions between two sprites, or a sprite and the screen.

The HIT function returns TRUE (-1) if a collision has taken place:

HIT <sprite number>+<offset>

The <offset> can be one of four values - 0, 8, 16 or 24 - and this determines the type of collision that is checked for. If it is zero, HIT checks for a collision between the specified sprite and any other sprite. If <offset> is 8, HIT checks for a collision between the sprite and the screen pixel data.

Once a sprite has collided with something, Laser BASIC still 'remembers' the collision even when the sprite moves away from the collision to another part of the screen. When HIT is used to test for a sprite-to-sprite collision, Laser BASIC will 'forget' any other sprite-to-sprite collisions that it had 'remembered' up to that time. Similarly, if HIT tests for a sprite to screen collision, Laser BASIC 'forgets' all sprite-to-screen collisions up to that time.

This would make it rather awkward to test for several sprite collisions at the same time, because when your program tests for the first sprite, it forgets any collisions that may have taken place with the other sprites to be tested. To get around this problem, you should use <offset> values of 16 and 24 instead of 0 and 8. These make Laser BASIC use the collisions that were 'remembered' the last time that HIT was used with an <offset> of 0 or 8. That is, an offset of 0 (or 8) is used to test for the first sprite, and an offset of 16 (or 24) is used for the rest:

```
200 IF HIT(1+0) THEN (TEST SPRITE 1)
210 IF HIT(2+16) THEN (TEST SPRITE 2)
220 IF HIT(3+16) THEN (TEST SPRITE 3)
```

LASER BASIC FOR THE COMMODORE 64

Another problem that you may come across is that collisions might have accidentally occurred when the sprites are being set up on the screen. This problem is easily avoided by putting the following line in your program after the sprites have been set up, to make Laser BASIC 'forget' any collisions'

```
12340 X=HIT(0) : X=HIT(8)
```

If you are working with four-colour mode sprites, only colours 2 and 3 will result in a collision being detected. Colour 0 and colour 1 are both treated as being transparent for the purpose of collision detection.

Example: This program places a software sprite a random distance out from the left of the hires screen and moves a tank (made from two hardware sprites) towards it until a collision occurs.

```
10 INIT
20 CONV 16,0,3,30
30 CONV 16,24,3,31
40 HSET 0,30
45 HSET 1,31
50 HCOL 0,GRAY1
55 HCOL 1,GRAY1
60 HX 0,24 : HX 1,48
70 HY 0,64 : HY 1,64
80 HON 0
85 HON 1
90 SCLR 0,1
95 PUTBLK 40,10 + RND(1) * 22,2
100 WINDOW 6
110 RASTER 100
114 MOVE 0,3,0
118 MOVE 1,3,0
120 X=HIT(9)
130 REPEAT: UNTIL HIT(9)
140 RASTER 100
150 MOVE 0,0,0
160 MOVE 1,0,0
170 PRINT "COLLISION"
```

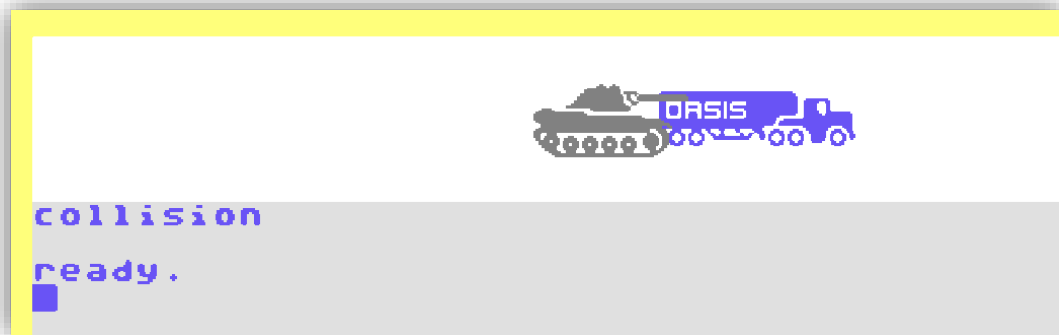
Available on disk as "ex hsprite 11"

Lines 10 to 80 set up hardware sprites 0 and 1 (which are the two halves of software sprite no. 16) and places them at the top left of the screen. Line 90 places sprite '10. 40 (a truck) at a random distance from the left of the screen.

In line 110, the tank which is made up of hardware sprites 0 and 1 is set moving across the screen. The RASTER command is used to ensure that both of the MOVE commands take effect at the same instant. When a text window is set up on the screen, the sprite positions are updated when the scan TV is at the bottom of the hires area, hence a scan line number of 100 is used in the RASTER command.

LASER BASIC FOR THE COMMODORE 64

The dummy HIT in line 120 ensures that any accidental collisions which have occurred are cleared. Line 130 waits until sprite no. 1, the right hand side of the tank, collides with the screen pixel data (i.e. the lorry). In line 140. the tank is stopped.



LASER BASIC FOR THE COMMODORE 64

MISCELLANEOUS COMMANDS

This section is devoted to four commands which don't really fit in anywhere else in this manual. These commands are for blanking out the entire screen and for controlling keyboard scanning.

DISPLAY BLANKING

(DBLANK, DSHOW)

The DBLANK command blanks the entire screen to the border colour. It has no parameters. DSHOW has the opposite effect - it turns the display on again. If you type DBLANK in immediate mode, the screen will flash off and on in a fraction of a second, because the display is always switched on when "ready" is printed.

These commands may be useful when you are setting up a screen to appear instantaneously. In this case, you could use DBLANK prior to setting up the screen, and DSHOW afterwards.

DISABLING KEYBOARD SCANNING

(KEYOFF, KEYON)

The KEYOFF command disables keyboard scanning on interrupt – this disables INPUT and GET but does not affect KB. This reduces the processor time used by the interrupt routine, and this may be important in time-critical sections of a program. KEYON turns keyboard scanning back on.

CHARACTER SPRITES AND SMOOTH SCROLLING

Up to this point, the manual has concentrated on the use of hires sprites. Hires sprites have two disadvantages:

- I. They are slow when large sprites are used
- II. They use up a lot of memory.

Because hires sprites are so slow, smooth-scrolling backgrounds cannot be created in hires mode. Character sprites can be used for this because they can be put on the screen much faster. Generally, the playing area is defined as one large character sprite (say 50 characters wide and 50 characters high), and only part of it is displayed on the screen at a time. A background of this size, if defined as a hires sprite, would use up over 40k of memory. Obviously, this is not practical. The equivalent character sprites only use up about 5k. Hardware sprites are usually used to provide any moving graphics on the screen, character sprites only being used for the background.

To use character sprites effectively, the character set must be redefined. This means that character sprites are rather more difficult to design, and they cannot be changed as easily as hires sprites. The sprite generator program has a facility for designing character sprites.

All the example programs in this section use the character sprite library, "spritesc". This can be loaded into the computer using RECALL "spritesc" for tape or DRECALL "spritesc" for disk users.

In order to refresh your memory, you might like to re-read the following sections before proceeding:

[GRAPHICS MODES](#)

[SPRITES](#)

[CHARACTER SET MANIPULATION](#)

LASER BASIC FOR THE COMMODORE 64

USING ATTRIBUTE COMMANDS WITH CHARACTER SPRITES

Character sprites are treated by Laser BASIC as hires sprites with no pixel data field. The attribute field holds the characters themselves, and the secondary attribute field holds the character attributes. The text screen is treated as sprite no. 255, with width 40 characters and height 25 characters. "The commands used with the attributes in hires sprites can

be used to manipulate character sprites. These commands are:

Command	Parameters
MAR	SPN, COL, ROW, WID, HGT
FLIPA	SPN, COL, ROW, WID, HGT
ATTL	SPN, COL, ROW, WID, HGT
ATTR	SPN, COL, ROW, WID, HGT
ATTUP	SPN, COL, ROW, WID, HGT
ATTDN	SPN, COL, ROW, WID, HGT
MOVATT	SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2
SWAPATT	SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2
SETA	SPN, COL, ROW, WID, HGT, A R
ATTGET	SPN, COL, ROW

Using SETA with Character Sprites

SETA can be used to fill a part of a character sprite with one character and attribute. To set up ATR with a character no. c and colour a, use ATR in conjunction with the MCOL3 command:

```
ATR=c : MCOL3a
```

Example: This short program places a block of purple 'a's at the top right hand corner of the text screen.

```
10 INIT  
20 ATR=1  
30 MCOL3 PURPLE  
40 ATT20N  
50 SETA 255,30,0,10,10,ATR
```

Note that there is an ATT20N command in line 40, so that the character attributes are set as well as the characters themselves.

LASER BASIC FOR THE COMMODORE 64



PUTTING AND GETTING CHARACTER SPRITES

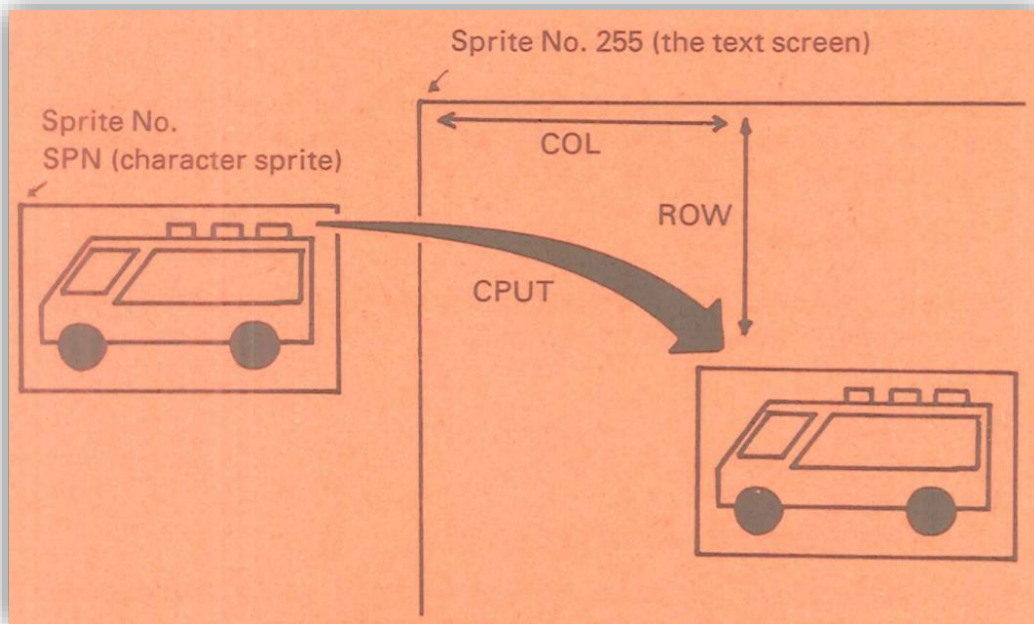
(CPUT, CGET, CSWAP)

Laser BASIC has three additional commands for use with character sprites.

CPUT puts the character sprite no. SPN onto the text screen at position COL,ROW:

CPUT SPN, COL, ROW

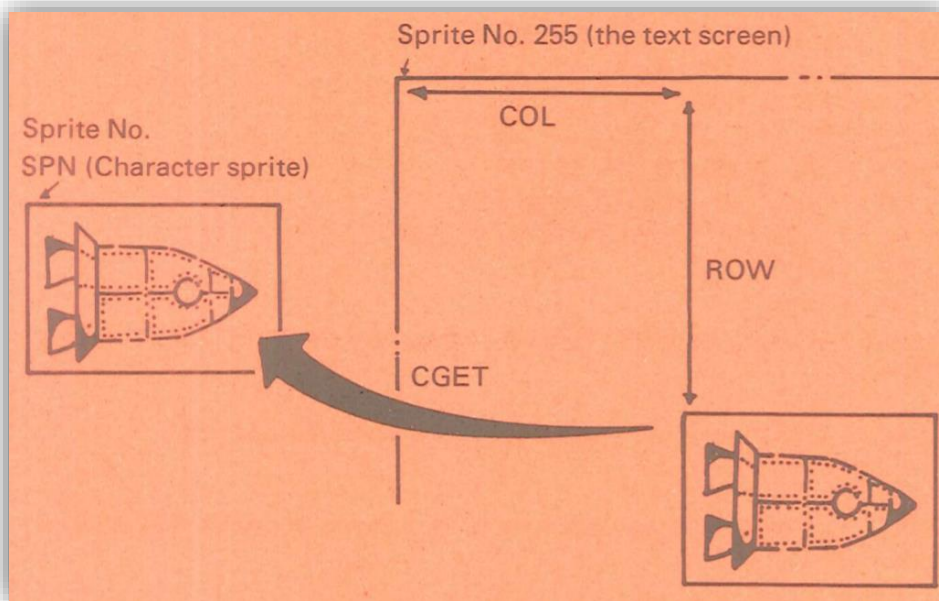
LASER BASIC FOR THE COMMODORE 64



CPUT is analogous to PUTBLK for hires sprites, although there is obviously no collision detection.

The CGET command gets a character sprite no. SPN from the text screen at position COL,ROW:

CGET SPN, COL, ROW

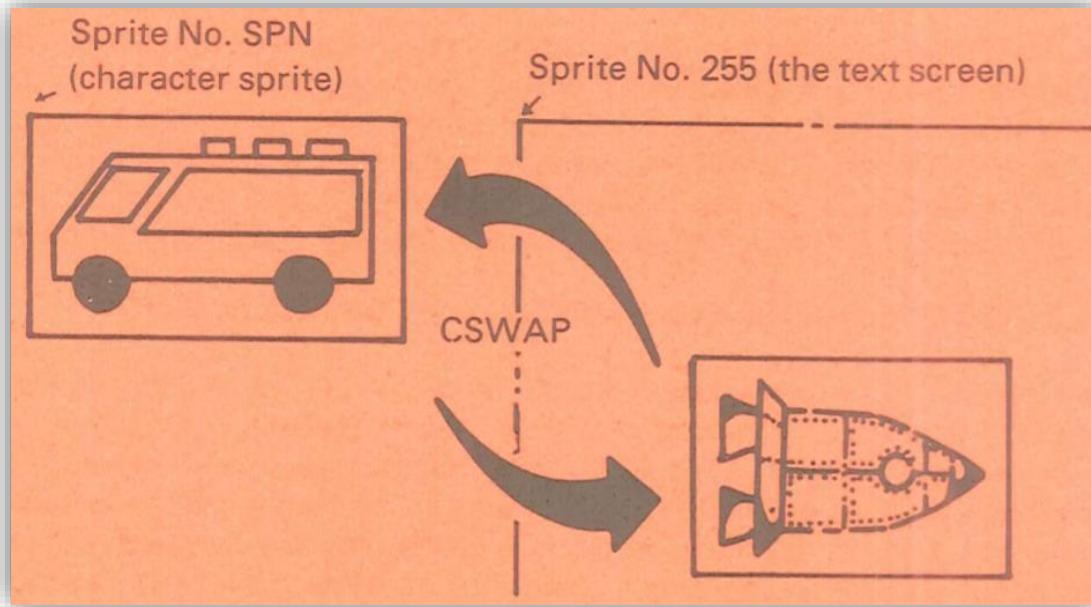


CGET is analogous to GETBLK for hires sprites.

CSWAP has no analogous command for hires sprites. It exchanges sprite no. SPN with the text screen at position COL,ROW:

LASER BASIC FOR THE COMMODORE 64

CSWAP SPN, COL, ROW



Example: This program places 16 copies of character sprite no. 6, a spaceship, at the top of the text screen. The spaceships are scrolled right over the screen using the ATTR command.

```
10 INIT
20 FOR I=0 TO 15
30   FOR J=0 TO 15
40     PUTCHR I,I,J,I + J * 16 + 512
50   NEXT J
60 NEXT I
70 ATT20N
80 FOR I=0 TO 6 STEP 2
90   FOR J=0 TO 30 STEP 10
100    CPUT 6,1 + J,I
110   NEXT J
120 NEXT I
130 RSYNC 51
140 RPT 400,ATTR 255,0,0,40,8
150 END
```

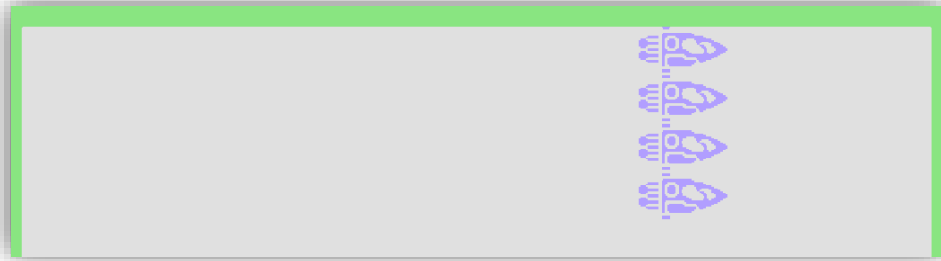
Available on disk as "ex chr sprite 02"

Remember that the character sprite library, "spritesc", must be loaded into the computer in order to run this program. Sprite no. 1 contains the character set to be used with these sprites. Lines 20 to 60 redefine the character set.

Note that only display codes 128 to 255 (i.e. the reversed characters) are redefined, so that the normal alphabetic characters can still be displayed on the screen. However, the redefined characters are noticeable when the cursor flashes.

LASER BASIC FOR THE COMMODORE 64

The ATT20N command is included in line 70, so that the character attributes are put on the screen as well as the characters. Lines 80 to 120 are a double loop which uses the CPUT command to place the sprite on the screen 16 times. The ATTR command is used in conjunction with RPT in line 140 to scroll the top 8 lines of the screen right 400 times. The RSYNC command in line 130 is included to make the scrolling smooth.



SMOOTH SCROLLING

(H38COL, H40COL, SCRX, SCRY)

Laser BASIC supports smooth scrolling of the entire screen in both horizontal and vertical directions. This means that you can place the screen in any of eight vertical and eight horizontal positions, each of which are one pixel apart. When displaying scrolling backdrops, smooth scrolling is used to scroll the screen by up to 7 pixels in one direction, after which a CPUT command is used to shift the entire contents of the screen by one character block.

Before using smooth scrolling, the entire screen must be shrunk to 38x24 characters, instead of the normal 40x25 characters. This covers up the blank areas which would appear at the edge of the screen when it is scrolled over. The H38COL and H40COL commands control the screen size

H38COL shrinks the screen to 38x24 characters

H40COL takes the screen back to normal size (40x25 characters)

If you are using hardware sprites in conjunction with the smaller screen, remember to take into account that the screen borders have moved when you position the sprites:

The left hand border has moved to the right by 8 pixels

The right hand border has moved to the left by 8 pixels

The top border has moved down by 4 pixels

The bottom border has moved up by 4 pixels

The smooth scrolling itself is carried out with the SCRX and SCRY commands:

SCRX <horizontal position> - scroll horizontally

SCRY <vertical position> - scroll vertically

For both commands, the position is a number from 0 to 7. With SCRX a position of zero corresponds to the screen's leftmost position and 7 is the rightmost. With SCRY, zero is the highest position and 7 is the lowest. The display normally uses a horizontal position of 0 and a vertical position of 3.

LASER BASIC FOR THE COMMODORE 64

SCRX and SCRY do not affect the position of hardware sprites.

Example: This program uses SCRX to move the screen back and forth by one character block.

```
10 INIT
20 FOR I=0 TO 200
30 PRINT I;
40 NEXT I
50 RSYNC 51
60 FOR K=1 TO 10
70 FOR J=1 TO 10
80 SCRX J
90 NEXT J
100 FOR J=6 TO 0 STEP-1
110 SCRX J
120 NEXT J
130 NEXT K
```

Available on disk as "ex chr sprite 03"

Lines 20 to 40 place some text on the screen. The RSYNC command in line 50 ensures that only one SCRX command can be executed in each display frame, ensuring that the scrolling is even. From line 70 to 120, the variable j is used to scroll the screen backwards and forwards.

Example: This program shows how to create a smooth scrolling background larger than the screen display. The background is scrolled under control of a joystick in control port 1. Scrolling can be speeded up with the fire button on the joystick.

```
10 INIT : RESERVE 14000
20 FOR I=0 TO 15
30 FOR J=0 TO 15
40 PUTCHR 1,I,J,512 + I + 16 * J
50 NEXT J
60 NEXT I
70 ATT2ON
80 TPAPER WHITE
90 CIF DFA (20) < 0
100 ATR=32
110 CSPRITE 20,80,50
120 FOR I=0 TO 79 STEP 5
130 FOR J=0 TO 49 STEP 5
140 MOVATT 2 + RND(1) * 11,0,0,255,255,20,I + RND(1) * 3,J +
RND(1) * 3
150 NEXT J
160 NEXT I: CEND
170 X=0 : Y=0 : CPUT 20,0,0 : KEYOFF : H38COL
180 `
190 IF JS1 <> 5 THEN 260
200 IF X=39 THEN 260
210 ROW = -Y
```

LASER BASIC FOR THE COMMODORE 64

```
215 REPEAT : X=X+1
220 RASTER 1 : COL = -X : SCRX 7 : CPUT
230 IF NOT FIRE1 : SCRX5 : RASTER1 : SCRX3 : RASTER1 : SCRX1
240 UNTIL JS1 <> 5 OR X=39
250 `
260 IF JS1 <> 1 THEN 320
270 IF X=0 THEN 320
280 ROW = -Y
285 REPEAT : X = X - 1
290 IF NOT FIRE1 : SCRX3 : RASTER1 : SCRX5 : RASTER1 : SCRX7
300 RASTER1 : COL=-X : SCRX 1 : CPUT
310 UNTIL JS1 <> 1 OR X=0
320 `
330 IF JS1 <> 3 THEN 390
340 IF Y=24 THEN 390
350 COL = -X
355 REPEAT : Y = Y + 1
360 ROW=-Y : RASTER 1 : SCRY 7 : CPUT
370 IF NOT FIRE1 : SCRY5 : RASTER1 : SCRY3 : RASTER1 : SCRX1
380 UNTIL JS1 <> 3 OR Y=24
390 `
400 IF JS1 <> 7 THEN 190
410 IF Y=0 THEN 320
420 COL = -X
425 REPEAT : Y = Y - 1
430 IF NOT FIRE1 : SCRY3 : RASTER1 : SCRY5 : RASTER1 : SCRX1
440 RASTER1 : ROW=-Y : SCRY1 : CPUT
450 UNTIL JS1 <> 7 OR Y=0
460 GOTO 190
```

Available on disk as "ex chr sprite 04"

The background is 80x50 characters, and in line 10, extra memory is reserved for sprites to accommodate this. In lines 20 to 60, the new character set is copied down from sprite 1. Lines 100 to 150 set up sprite 20 the background. This is only executed if sprite 20 does not already exist, because of the CIF statement in line 90. Note that ATR is set to 32, the display code for a space, in line 100, so that sprite 20 is filled with spaces when it is defined in line 110. In lines 120 to 160, sprites 2 and 12 are placed in sprite no. 20 at random, with the MOVATT command. Although WID and HGT are both set to 255, the window will be automatically clipped to the size of the sprite.

In line 170, x and y are the screen's current position within the background. Since the keyboard scanning is switched off the only way to escape from this program is by using RUN/STOP-RESTORE.

Line 190 onwards is the main loop which checks if the joystick is in use, and moves the background accordingly. It consists of four very similar pieces of program, one for each direction - left, right, up and down.

LASER BASIC FOR THE COMMODORE 64

Lines 260 to 310 check for joystick right. Lines 260 and 270 jump to the next section of the program at line 320 if the joystick is not in the correct direction, or the background cannot be moved right (i.e. $x=0$).
The

REPEAT-UNTIL loop from line 280 to 310 repeats the scrolling action until the joystick is released, or the background cannot be moved further to the right. Since the vertical position is always the same, ROW is set up outside the loop. Inside the loop, the smooth scrolling command SCRX is only used if the fire button is not pressed. All the SCRX commands are synchronised to the display scan. After this, the CPUT command completes the scrolling by placing a portion of the background on the screen at its new position. The timing in this part of the program is critical you may notice a slight flicker on the bottom line of the screen. However this disappears when the program is compiled.

LASER BASIC FOR THE COMMODORE 64

SOUND COMMANDS

The SID (sound interface device) inside your Commodore 64 is the most powerful sound synthesiser in any personal computer. Laser BASIC has a set of 18 commands which allow you to use the SID's capabilities to the full.

To make a sound from the SID chip, your program must do six things.

1. Initialise SID.
2. Set up the master volume.
3. Set up the frequency.
4. Set up the envelope.
5. Set up the waveform.
6. Sound the note.

INITIALISING THE SID

(SIDCLR)

The SIDCLR command has no parameters, and it simply clears out the SID chip. Generally, it only needs to be used once, at the beginning of a program. In this respect it is rather similar to the IN'T command for graphics. However, SIDCLR does not alter the SID master volume.

SETTING THE MASTER VOLUME

(VOLUME)

The SID chip has three sound channels, or voices, and the volume of all these is set by the VOLUME command:

VOLUME <volume>

The minimum volume is zero (silent) and the maximum is 5

SETTING THE FREQUENCY

(FRQ)

The FRO command sets the frequency, or pitch, of a voice:

FRQ <voice>,<frequency>

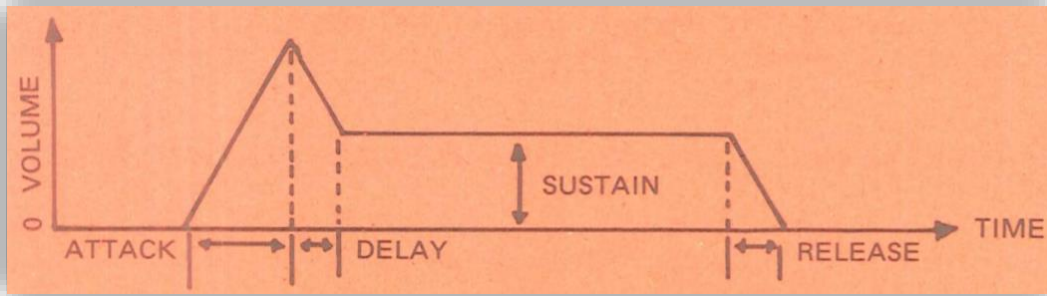
The <voice>" can be 1, 2 or 3. The <frequency> is a number from 0 to 65535. This is not in Hz (cycles per second); to use a frequency in Hz it must be multiplied by 16.4. A table of frequencies for all notes of the musical scale can be found in [Appendix I](#).

SETTING THE ENVELOPE

(ADSR)

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

LASER BASIC FOR THE COMMODORE 64



After being struck, the volume rises to its peak value at a rate determined by the 'attack'. It then falls to the 'sustain' level at a rate determined by the 'decay'. At the end of the note, the volume falls away to zero at the 'release' rate. This 'envelope' shape can be set up using the ADSR command:

```
ADSR <voice>,<attack>,<decay>,<sustain>,<release>
```

The <voice> is a number from 1 to 3; all the other parameters are numbers from 0 to 15.

SETTING THE WAVEFORM

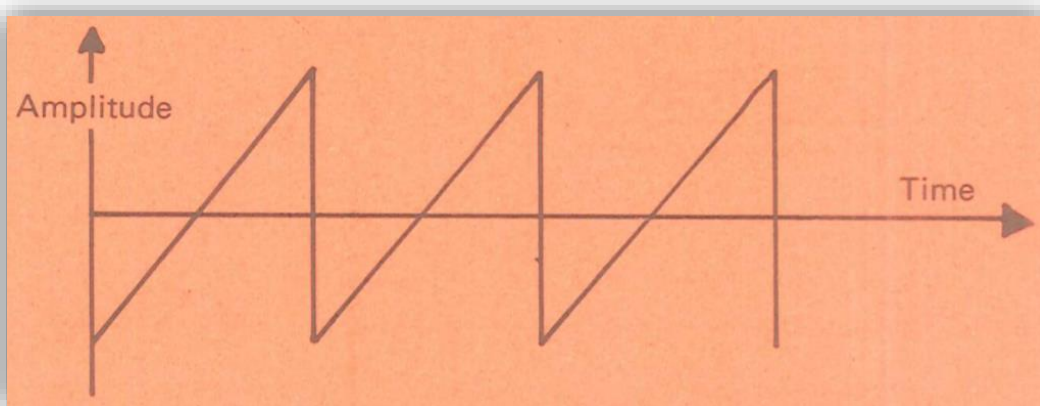
The SID chip can generate four different types of waveform, each one giving a different sound, or timbre:

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

Each voice can be set to use a different waveform.

SAWTOOTH WAVEFORM (SAW)

If a sawtooth wave were displayed on an oscilloscope, it would look like this:



To set one of the SID's sound channels to the sawtooth waveform the SAW command is used:

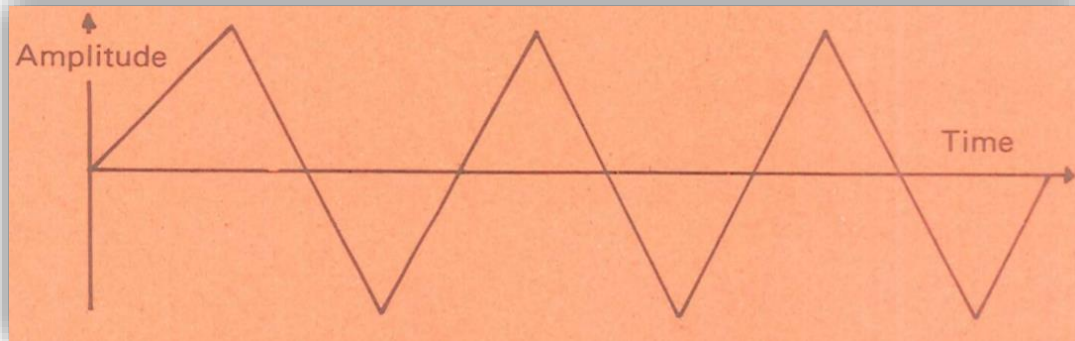
```
SAW <voice>
```


LASER BASIC FOR THE COMMODORE 64

The <voice> can be 1, 2 or 3.

TRIANGLE WAVEFORM (TRI)

A triangle wave displayed on an oscilloscope looks like this:



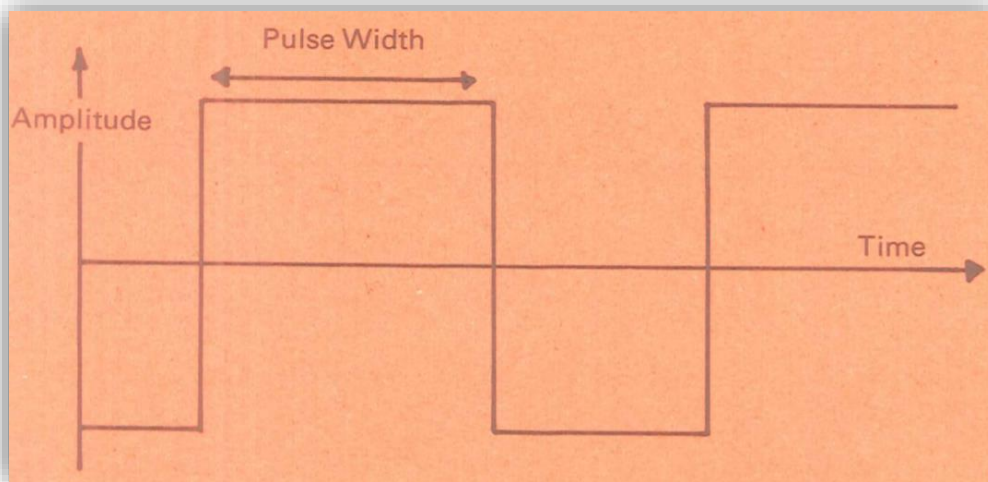
To set a sound channel to the triangle waveform, the TRI command is used:

```
TRI <voice>
```

The <voice> can be 1, 2 or 3.

PULSE WAVEFORM (PULSE)

A pulse wave looks like this when displayed on an oscilloscope:



To set a sound channel to use pulse waves, the PULSE command is used:

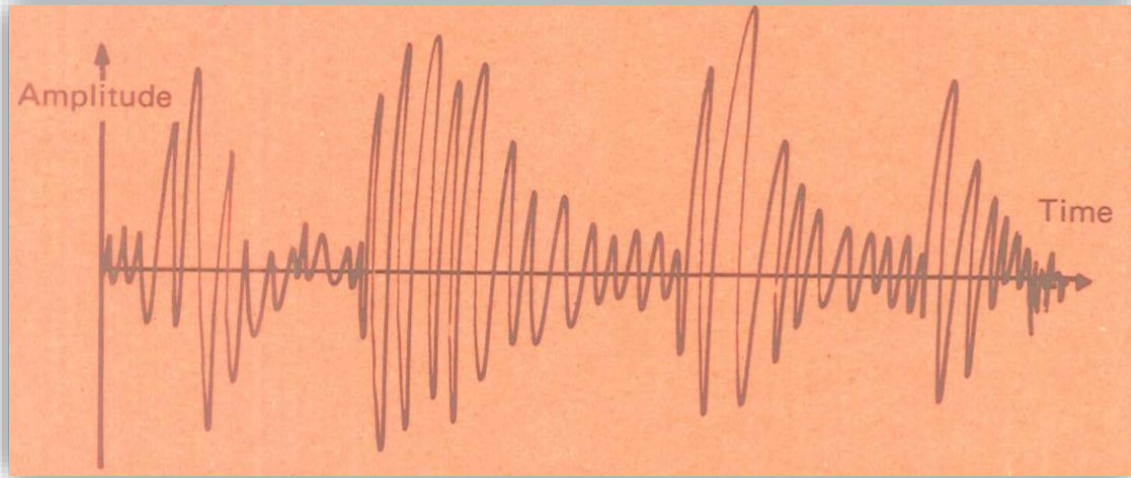
```
PULSE <voice>,<pulse width>
```

The <voice> can be 1, 2 or 3. The <pulse width> is a number from 0 to 4095. Varying the pulse width allows a wide variety of different sounds to be generated.

LASER BASIC FOR THE COMMODORE 64

WHITE NOISE (NOISE)

White noise is basically random, and gets its name from the fact that, like white light, it contains energy at all frequencies. If displayed on an oscilloscope, it would look like this:



White noise is very useful for generating realistic explosions. The NOISE command makes a sound channel generate white noise:

```
NOISE <voice>
```

MAKING A SOUND (MUSIC)

Once a sound channel has been set up, you can use the MUSIC command to make a sound. It has two parameters:

```
MUSIC <voice>,<length>
```

The <voice> can be 1, 2 or 3. The <length> is measured in 60ths of a second and lies between 1 and 255. If you use a length of zero, the sound stays on until it is turned off by another MUSIC command with a non-zero length. The length of a note is the time from the start of the attack to the start of the release. After the note has started, your program continues to execute - the note is turned off automatically by Laser BASIC while your program does something else.

Example: This short program sounds a 440Hz note for a third of a second.

```
10 SIDCLR
20 VOLUME 15
30 FRQ    1,7217
40 ADSR   1,59 8,5,9
50 TRI    1
60 MUSIC  1,20
```

Line 10 - initialise the SID.

Line 20 - set the volume to the maximum setting.

Line 30 - set the frequency to 440Hz ($44 \times 16.4 = 7217$).

LASER BASIC FOR THE COMMODORE 64

Line 40 - set up the envelope.

Line 50 - set up the waveform (triangle wave).

Line 60 - sound the note.

Once you have typed in and run this program, try changing ne 50 to hear what the different waveforms sound like:

```
Sawtooth: 50 SAW 1
```

```
Pulse waves: 50 PULSE 1,250
              50 PULSE 1,500
              50 PULSE 1,1000
              50 PULSE 1,1500
              50 PULSE 1,2000
```

```
White noise: 50 NOISE 1
```

By changing the envelope, white noise can be used to simulate gunfire or explosions:

Change line 50 to:

```
50 NOISE 1
```

then change line 40 to:

```
40 ADSR 1,1,9,3,9
```

If you run the program, a sound like gunfire is produced. Now try changing line 30 to:

```
30 FRO 1,3000
```

and line 40 to:

```
40 ADSR 1,0,13,5,12
```

The sound effect now sounds more like an explosion. By experimenting with this program, you will discover many more sound effects which can be produced. Remember that only one voice is being used - you may like to try producing multiple sound effects by using more than one voice.

Example: This program produces an interesting effect by varying the frequency of a sound channel during a note:

```
10 SIDCLR
20 VOLUME 15
30 ADSR 1,1,9,3,9
40 TRI 1
50 MUSIC 1,30
60 FOR J=1 TO 30
70   FOR I=12000 TO 28000 STEP 800
80     FRQ 1,I
90   NEXT I
```

LASER BASIC FOR THE COMMODORE 64

```
100 NEXT J
110 END
```

Available on disk as "ex sound 01"

Example: This program produces a sound like a police siren:

```
10 SIDCLR
20 VOLUME 15
30 ADSR 1,15,15,15,15
40 PULSE 1,1000
50 X=10000
60 Y=8409
70 MUSIC 1,255
80 FOR J=0 TO 32
90 FRQ 1,11
100 FOR I=0 TO 200
110 NEXT I
120 FRQ 1,Y
130 FOR I=0 TO 200
140 NEXT I
150 IF J=7 THEN X=9803 : Y=8244
160 NEXT J
170 SIDCLR
180 END2
```

Available on disk as "ex sound 02"

The first four lines set up voice no. 1. The variables x and y are the frequencies of the upper and lower notes of the siren; these are changed in line 150 so that it sounds as if a police car is passing.

PLAYING MUSIC UNDER INTERRUPT

(PLAY, RPLAY)

Laser BASIC has a facility which allows you to play music under interrupt. Like the TRACK command for hardware sprites (section [TRACKING SPRITES \(TRACK\)](#)), the music is held in a coded form inside the pixel data field of a software sprite. One software sprite is required for each voice to be used. Details of the format used for storing music will be given later.

The PLAY command has three parameters:

```
PLAY SPN, COL, ROW
```

In this case SPN, COL and ROW are all sprite numbers - SPN is the sprite to be played by voice 1, COL is the sprite for voice 2 and ROW is the sprite for voice 3. To have sprite 3 played by voice 1, sprite 9 by voice 2 and sprite 15 by voice 3, you would use:

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

LASER BASIC FOR THE COMMODORE 64

To keep a voice silent, so that it can be used for other purposes such as generating sound effects, you must specify sprite no. 0. To have sprite 4 played by voice 1, and the other two voices silent, you would use:

```
PLAY 4,0,0
```

To stop all three voices playing music, you would use:

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

Example: To run this program you must have the sprite library "sprites" loaded into the computer. Sprites 13 and 14 are both hires spdirt7rites containing music

```
10 SIDCLR
20 VOLUME 15
30 PULSE 1,1508
40 PULSE 2,1880
50 ADSR 1,2,3,5,2
60 ADSR 2,2,3,5,2
70 PLAY 13,14,0
80 END
```

Available on disk as "ex sound 03"

Before using the PLAY command, the volume, waveform and envelope must be set up as normal and lines 10 to 60 of the program do this for voices 1 and 2. In line 70, the music in sprites 13 and 14 is played on voices 1 and 2 voice 3 remains silent.

With the PLAY command, the music is only played once - to play it repeatedly the RPLAY command can be used instead of PLAY. In this case, the music is repeated until PLAY 0,0,0 is used to silence all the voices.

Try changing line 70 in the previous example program to:

```
70 RPLAY 13,14,0
```

and run it again.

Using PLAY as a Function PLAY can be used as a function to determine if music is still playing on any of the three voices. If music is still being played, it returns a value of -1; otherwise it returns a value of 0.

Example: This is a modified version of the previous example which does not go back into immediate mode until the music has stopped.

```
10 SIDCLR
20 VOLUME 15
30 PULSE 1,1600
40 PULSE 2,1000
50 ADSR 1,2,3,5,2
60 ADSR 2,2,3,5,2
70 PLAY 13,14,0
```

LASER BASIC FOR THE COMMODORE 64

```
80 REPEAT
90 UNTIL NOT PLAY
100 END
```

Available on disk as "ex sound 04"

The REPEAT -UNTIL loop in lines 80 to 90 waits until the music has stopped.

When music is being played under interrupt, you must not delete any sprites with WIPE. Delete lines 80 and 90 from the above program. Now run it and then type WIPE 3; listen to what happens!

Format for Storing Music inside Sprites

Each note takes up 4 bytes in the pixel data field of the software sprite.

This consists of:

2 bytes frequency	stored in normal high byte/low byte order.
1 byte length	measured in 60ths of a second.
1 byte gap	the length of time between the end of the note and the start of the next one. It is measured in 60ths of a second.

As when using the TRACK command for hardware sprites, a short BASIC program can be used to poke the music into a sprite:

```
10 DATA C, 34334,C#,36376,D, 38539
20 DATA D#,30830,E, 43258,F, 45830
30 DATA F#,48556,G, 51443,G#,54502
40 DATA A, 57743,A#,61176,B, 64814
50 DIM N$(11),N(11)
60 FOR I=0 TO 11
65 READ N$(I),N(I)
70 NEXT I
75 SN=200 `SPRITE NUMBER
80 IF DFA(SN) > 0 THEN WIPE
90 REPEAT: N=N+1 : READ A$ : UNTIL A$ = "."
100 N=N-1 : IF N - 3 * INT(N/3) THEN STOP
110 K=INT((N+3)/6)
115 I=INT(SQR(K))
120 REPEAT
130 IF K/I = INT(K/I) AND K/I < 256 THEN EXIT
140 I=I-1
145 UNTIL I=0
150 IF I=0 : K=K+1 : GO TO 110
160 SPRITE SN,K/I,I : M=DFA(SN)
170 RESTORE 310
175 FOR C=1 TO N STEP 3
180 READ K$ : A$=LEFTS(K$,LEN(K$)-1)
190 B$ = RIGHT$(K$,1)
200 IF BS < "0" OR B$ > "7" THEN STOP
```

LASER BASIC FOR THE COMMODORE 64

```
210 I1=0: I2=-1
215 REPEAT
220 IF A$ = N$(I1) THEN I2=I1
230 I1=I1 + 1
235 UNTIL I1=I2 OR I2 >= 0
240 DOKE M, N(I2) / ( 2^(7 - VAL(B$))) + .5
250 READ J : POKE M+2,J
260 READ J : POKE M+3,J
270 M=M+4
275 NEXT C
280 IF K * 6 <> N THEN DOKE M+2,257
290 SIDCLR : VOLUME 15 : TRI1
300 ADSR 1,4,3,5,2 : PLAY SN,0,0
310 `
320 DATA C3 , 16 , 16
330 DATA D3 , 16 , 16
340 DATA E3 , 16 , 16
350 DATA C3 , 16 , 16
360 DATA G3 , 80 , 16
370 DATA A3 , 16 , 16
388 DATA G3 , 16 , 16
390 DATA F3 , 16 , 16
400 DATA C3 , 16 , 16
410 DATA F3 , 16 , 16
420 DATA G3 , 16 , 16
430 DATA F3 , 16 , 16
440 DATA E3 , 16 , 16
450 DATA D3 , 16 , 16
460 DATA .
```

Available on disk as "ex sound 05"

The DATA statements from line 320 onwards contain:

note duration, gap

The note consists of one of the twelve notes of the musical scale:

c c# d d# e f f# g g# a a# b

followed by an octave number from 0 to 7. 0 is the lowest octave, and 7 is the highest.

The duration of the note, and gap until the next note are both numbers under 255.

The program takes care of calculating the correct size of the sprite. The music is placed in sprite no. 200, although this can easily be changed in line 70. Lines 290 and 300 play the music on voice 1 once it has been put into the sprite.

Completed sprites which have been created with this program can be saved off to tape or disk with STORE or DSTORE.

LASER BASIC FOR THE COMMODORE 64

FILTERING

(CUTOFF, PASS, RESONANCE, FILTER)

The SID chip allows filtering of the sound from one or more of its voices. Filtering removes a particular frequency component (i.e. low frequencies or high frequencies), and this alters the timbre of the sound which is produced.

SETTING THE CUTOFF FREQUENCY

(CUTOFF)

The CUTOFF command sets up the cutoff frequency used by the SID's filter:

CUTOFF <frequency>

The <frequency> is a number from 0 to 65535; this is the frequency in Hz multiplied by 16.4.

SELECTING THE FILTER

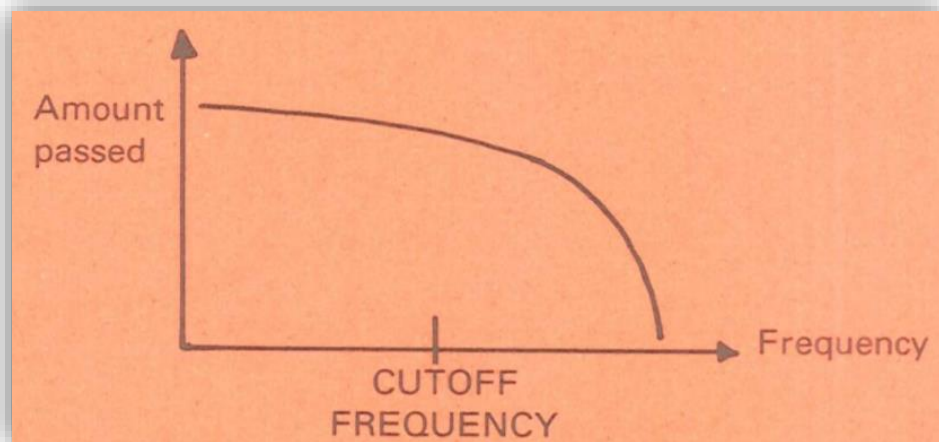
(PASS)

There are four filters in the Commodore's SID chip:

I. Low Pass Filter

This is selected by: **PASS 0**

As its name implies, this filter only allows low frequency signals to pass, and attenuates (cuts down) high frequency signals:

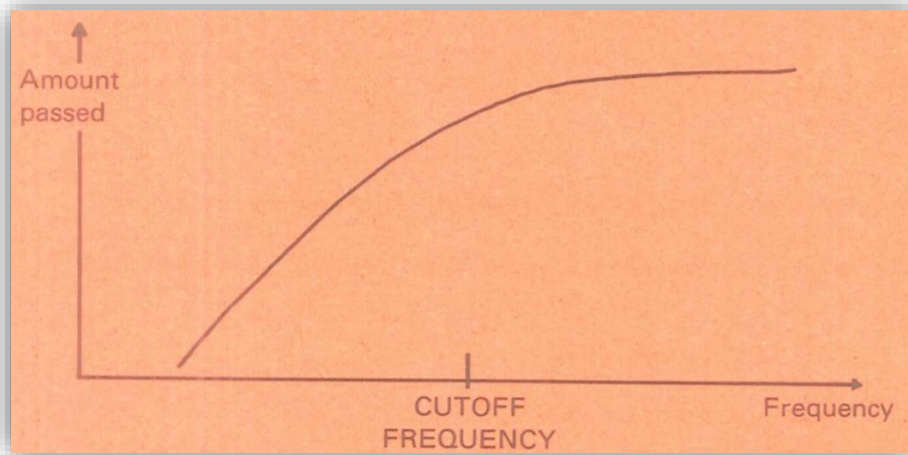


II. High pass filter

This is selected by: **PASS 1**

This filter only passes high frequencies:

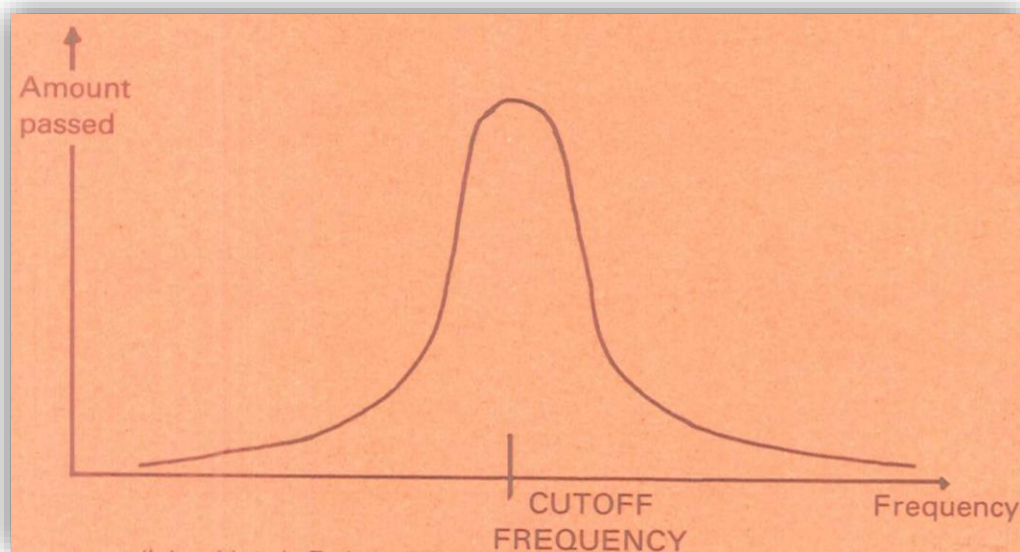
LASER BASIC FOR THE COMMODORE 64



III. Band Pass Filter

This is selected by: **PASS 2**

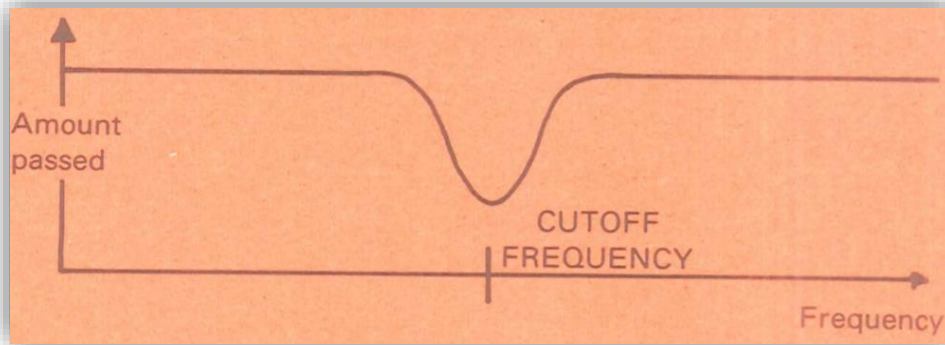
This filter only passes frequencies around the cut off:



IV. Notch Reject Filter

This is selected by: **PASS 3**

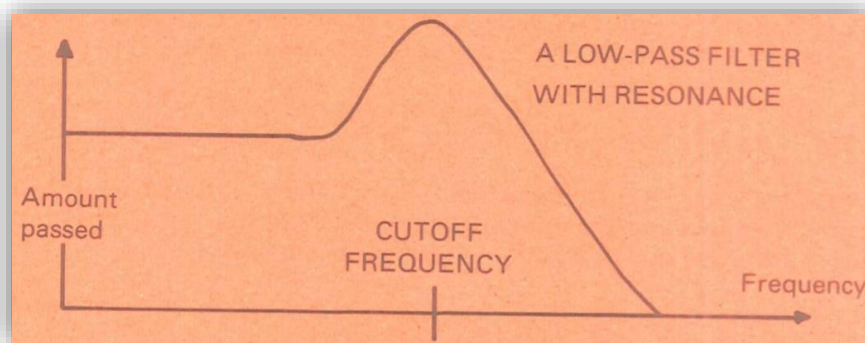
LASER BASIC FOR THE COMMODORE 64



This filter passes all frequencies away from the cut off, while attenuating the cut off frequency:

RESONANCE (RESONANCE)

The SID's filter can also resonate around the cut off frequency. This means that frequencies around the cutoff are amplified relative to other frequencies.



The RESONANCE command is followed by a number from 0 to 15:

RESONANCE <number>

Zero indicates no resonance, i.e. normal operation; 15 gives maximum resonance.

ENABLING THE FILTER

(FILTER)

Once the filter has been set up using CUTOFF, PASS, and, if necessary RESONANCE, it must be turned on with the FILTER command.

FILTER <voice>, TRUE	the output from the specified voice goes through the filter.
FILTER <voice>, FALSE	The output of the specified voice goes straight to the loudspeaker without going through the filter.

LASER BASIC FOR THE COMMODORE 64

Example: This program generates a realistic explosion by applying lowpass filtering with resonance to white noise.

```
10 SIDCLR
20 VOLUME 15
30 FRQ 1,3000
40 ADSR 1,0,13,5,12
50 NOISE 1
60 CUTOFF 3800
70 PASS 0
80 RESONANCE 12
90 FILTER 1,TRUE
100 MUSIC 1,20
110 END
```

Available on disk as "ex sound 06"

RING MODULATION

(RING)

Ring modulation combines two voices, and can be used to generate very complex waveforms. It is turned on with the RING command:

```
RING <voice>,TRUE - turn on ring modulation
RING <voice>,FALSE - turn off ring modulation
```

With ring modulation enabled on voice 1, it is replaced by a ring modulated combination of voice 1 and voice 3. There is no need to set up voice 3's envelope since it is not used for ring modulation.

Similarly, voice 2 is replaced by voice 2 modulated with voice 1, and voice 3 is replaced with voice 3 modulated with voice 2.

Example: Due to the non-harmonic overtone structure produced by ring modulation, it can be used to generate realistic bell effects:

```
10 SIDCLR
20 VOLUME 15
30 ADSR 1,1,9,8,12
40 TRI 3
50 TRI 1
60 RING 1,TRUE
70 FILTER 1,TRUE
80 PASS 0
90 CUTOFF 2560
100 READ A `read first freq.
110 WHILE A <> -1
120 PROCBELL (A)
130 READ D `read duration.
135 PRINT "FREQ: ";A;TAB(12);"DUR: ",d
140 FOR B=1 TO D
```

LASER BASIC FOR THE COMMODORE 64

```
150 NEXT B
160 READ A `read next freq.
170 WEND
180 END
190 `
200 LABEL BELL (FR)
210 FRQ I,FR
220 FRQ 3,FR * 1.9766
230 MUSIC 1,40
240 PROCEND
250 `
260 DATA 1514,700 ,1201,700 ,1340,700 ,900
270 DATA 1400,900 ,700 ,1348,700 ,1514,700
280 DATA 1201,1400,1201,1500,1201,1500
290 DATA 1201,1500,1201,1500,1201,1500
300 DATA 1201,1500,1201,1500,1201,1500
310 DATA 1201,1500,1201,1500,1201,1500
320 DATA -1
```

Available on disk as "ex sound 07"

```
run
freq: 1514 dur: 700
freq: 1201 dur: 700
freq: 1340 dur: 700
freq: 900 dur: 1400
freq: 900 dur: 700
freq: 1348 dur: 700
freq: 1514 dur: 700
freq: 1201 dur: 1400
freq: 1201 dur: 1500
freq: 1201 dur: 1500
freq: 1201 dur: 1500
freq: 1201 dur: 1500
freq: 1201 dur: 1500
freq: 1201 dur: 1500
freq: 1201 dur: 1500
freq: 1201 dur: 1500
freq: 1201 dur: 1500
freq: 1201 dur: 1500
freq: 1201 dur: 1500
freq: 1201 dur: 1500
```

LASER BASIC FOR THE COMMODORE 64

SYNCHRONISATION

(SYNC)

Like ring modulation, synchronisation combines two voices. This is enabled by the SYNC command which has the same syntax as RING

SYNC <voice>,TRUE - turn on synchronisation

SYNC <voice>,FALSE - turn off synchronisation

With synchronisation enabled on voice 1, it is replaced by voice 1 synchronised with voice 3. There is no need to set up voice 3's envelope since it is not used for synchronisation. Similarly, voice 2 can be replaced by voice 2 synchronised with voice 1 and voice 3 is replaced by voice 3 synchronised with voice 2.

Example: Synchronisation can be used to mimic the sound of engines. This program imitates a motorbike.

```
10 SIDCLR
20 '
30 VOLUME 15
40 ADSR 1,15,5,15,15
50 TRI 1
60 TRI 3
70 SYNC 1, TRUE
80 READ J `read first.
90 WHILE J <> -1
110 READ K,M
120 MUSIC 1,100
130 FOR I=J TO K STEP M
140   FRQ 3,I
150   FRQ 1,I * 2.5
160 NEXT I
170 READ J `read next
180 WEND
190 DATA 270, 750,1.4,
200 DATA 650,1000,1.4
210 DATA 750,1150,1.4
220 DATA 1000,1400,1
230 DATA -1
```

Available on disk as "ex sound 08"

DYNAMIC EFFECTS

(OSC, ENV, MUTE)

Many interesting sound effect can be generated by changing a voice's frequency dynamically during a note or sound. To make this easy to do, Laser BASIC lets you access the digitised output of channel3's oscillator and envelope generator. These can be used to modify voice 1 or 2's frequency

LASER BASIC FOR THE COMMODORE 64

Before you lose voice 3 for this purpose, it must be turned off with the MUTE command so that it cannot be heard.

MUTE TRUE - turn off voice 3

MUTE FALSE - turn voice 3 on again

OSC and ENV are both functions which return the output of voice 3's oscillator and envelope generator respectively. These are both represented as integers between 0 and 255. For ENV to function correctly, voice 3's oscillator must be enabled (i.e. a waveform must be set up with SAW, TRI, NOISE or PULSE).

Example: This program uses OSC to create a vibrato effect on voice 1.

```
10 SIDCLR
20 VOLUME 15
30 MUTE TRUE
40 TRI 1
50 TRI 3
60 FRQ 1, 7000
70 FRQ 3, 80
80 ADSR 1, 3, 4, 5, 10
90 MUSIC 1, 100
100 MUSIC 3, 0
110 FOR I=0 TO 230
120 FRQ I, 7000 + OSC
130 NEXT I
140 END
```

Available on disk as "ex sound 09"

Line 10 - clear the SID chip.

Line 20 - set the master volume.

Line 30 - disable voice 3.

Line 40 - set voices 1 and 3 to triangle waves.

Line 50 - set frequencies of both voices.

Line 60 - set voice 1's envelope.

Line 70 - sound the note on voice 1 - the inaudible note on voice 3 continues indefinitely.

Lines 80 to 100 - continually modifies voice 1's frequency using OSC to generate a vibrato effect.

LASER BASIC FOR THE COMMODORE 64

MULTI TASKING

Laser BASIC has a simple multi-tasking facility which makes possible to execute up to three different parts of a program simultaneously. This is done using a technique called time-slicing, where one part of the program is executed for 1/120th of a second before switching over to the next

Multi-tasking must be used with care in order to obtain good results especially when graphics commands are being used.

ALLOCATING MEMORY

(ALLOCATE)

Each of the three concurrent tasks has its own set of BASIC variables which are independent of each other. Task number zero is running all the time and therefore memory does not have to be allocated for it. The ALLOCATE command sets aside memory for tasks 1 and 2.

```
ALLOCATE <task 1 memory>,<task 2 memory>
```

For example, ALLOCATE 100,200 reserves 100 bytes for task 1 and 200 bytes for task 2. ALLOCATE 150,0 reserves 150 bytes for task 1 and none for task 2. ALLOCATE performs a CLR and should therefore go near the beginning of your program.

However, if the RESERVE command is used, it must go before ALLOCATE because RESERVE de-allocates any memory reserved for concurrent tasks.

STARTING A CONCURRENT TASK

(TASK)

Once enough memory has been allocated for a task, it can be invoked with the TASK command:

```
TASK <task number>.<line number>  
or TASK <task number>.<label>
```

The task number can be either 1 or 2. Task number zero is the main program which runs all the time.

To avoid conflict, certain commands cannot be used in tasks 1 or 2:

ALLOCATE	CLOSE	CMD	CSPRITE	DLOAD
DRECALL	DSAVE	DSTORE	GET	GET#
INPUT	INPUT#	LIST	LOAD	MERGE
OPEN	PRINT	PRINT#	RECALL	RESERVE
RESET	RUN	SPRITE	STORE	TASK
VERIFY				

Example: This program uses two concurrent tasks to print out the numbers from 1 to 100 and flash the text background colour simultaneously.

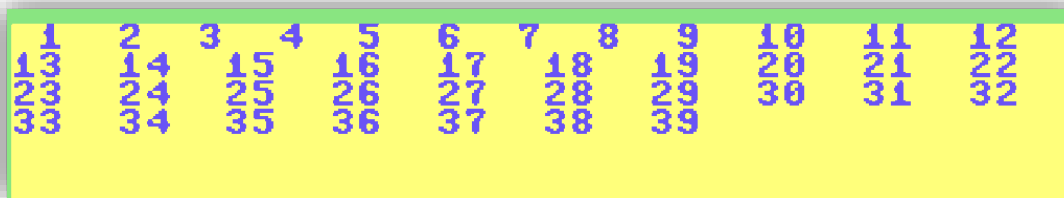
```
10 INIT  
20 ALLOCATE 50,0  
30 TASK 1,80
```

LASER BASIC FOR THE COMMODORE 64

```
40 FOR I=1 TO 100
50 PRINT I;
60 NEXT I
70 END
80 REPEAT
90 FOR I=0 TO 15
100 TPAPER I
110 NEXT I
120 UNTIL FALSE
```

Available on disk as "ex multitask 01"

In line 20,50 bytes are allocated for task 1. This is more than enough since only one variable is used. Line 30 starts execution of task 1 starting at line 80. The two sections of program from line 40 and line 80 now execute in parallel. Note that both tasks use variable i, but because each task has its own set of variables, they do not become; confused. Both tasks stop executing when task 0 reaches the END statement in line 70.



1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22		
23	24	25	26	27	28	29	30	31	32		
33	34	35	36	37	38	39					

STOPPING A CONCURRENT TASK

(HALT)

You can think of the HALT command as being the opposite to TASK - HALT stops a concurrent task from executing. If HALT is used on its own without parameters, it stops the current task from executing. It cannot be used in task 0 because task 0 runs all the time. HALT can also be followed by a number; either 1 or 2 to stop a different task from executing.

Example: This program shows how HALT can be used to stop a task. It repeatedly inputs a string from the keyboard. If the string is "scroll", Multi-tasking is used to set up a scrolling title at the top of the screen.

Typing "done" exits to command mode.

```
10 REPEAT
20 INPUT AS
30 CIF AS = "SCROLL"
40 HALT 1
50 PRINT CHR$(147); "LASER BASIC"
60 TASK 1,100
70 CEND
80 IF AS = "DONE" THEN END
90 UNTIL FALSE
```


LASER BASIC FOR THE COMMODORE 64

```
100 ATT20N
110 RPT 400,ATTR 255,0,0,40,1
120 HALT
```

Available on disk as "ex multitask 02"

Lines 40 to 60 are executed if the string was "scroll"; in line 40 task no. 1 is cancelled, in case it is already executing. Line 50 prints the title at the top of the screen to be stored. Line 60 starts concurrent task no. 1 executing at line 100. In lines 100 and 110, RPT and ATTR scroll the top line of the screen. In line 120, task 1 halts itself with the HALT command.

ERROR MESSAGES WITH MULTI-TASKING

When an error occurs or the program ends in task 0, the error message and "ready" message are printed as usual. In task 1 and task 2, the computer first waits for the current command in all other tasks to finish. Then, the task number is printed out inside angle brackets before the error message (if there is one), like this:

```
<2>
?type mismatch error in 1420
ready.
```

To see this, type in the following and try to run it:

```
10 TASK 1,30
20 GOTO 20
30 !"#§
```

COMMUNICATION BETWEEN TASKS

(COMMON%, IMPORT)

Often different tasks have to pass numbers between one another – for example. the speed of a sprite on the screen. The simplest way to do this is with the array COMMON%. This is an array which holds 64 integers from -32768 to 32767, and is the same for all three concurrent tasks. Note that the elements in COMMON% are not set to zero by CLR or RUN.

Example: This program inputs numbers from the keyboard and changes the text screen's border colour simultaneously. The rate that the border colour changes depends on the numbers input from the keyboard.

```
10 ALLOCATE 50,0
20 TASK 1,80
30 REPEAT
40 INPUT A
50 COMMON%(1)=A
60 UNTIL A<0
70 STOP
80 REPEAT
90 FOR I=1 TO COMMON%(1)
100 NEXT I
```

LASER BASIC FOR THE COMMODORE 64

```
110 J=J + 1
120 IF J=16 THEN J=0
110 TBORDER J
120 UNTIL FALSE
```

Available on disk as "ex multitask 03"

COMMON%(1) is used to hold the rate that the border colour is to change. Task 0 from lines 30 to 60 inputs numbers from the keyboard and places them in COMMON%(1). Task 1 from line 80 onwards uses COMMON%(1) as the length of the delay between changing colours. Variable i is used in the delay loop, and j is the current border colour.

IMPORT is much more powerful, but not as efficient, as COMMON%. It can be used to pass real numbers and strings between tasks. It takes the form of a function

```
IMPORT (<task number>,<expression>
```

Tile <expression> . which can be numeric or string, is evaluated inside the specified task (0, 1 or 2). Using IMPORT is much slower than COMMON% because the task must finish executing its current command before IMPORT can evaluate the expression.

USING MULTI-TASKING WITH GRAPHICS

(EI, D1)

If multi-tasking is being used, it is best to try to confine the graphics to hardware sprites as far as possible. This is because a problem arises if Laser BASIC switches over to another task in the middle of a PUT or scrolling command. If half the window has been updated when Laser BASIC switches over to another task, the window will remain that way for at least a tenth of a second, until the task resumes operation. This results in an unpleasant flickering effect on the screen. One way of avoiding this is to use the D1 and EI commands. D1 disables the switching over between tasks, and EI re-enables it. Thus, D1 is put before the block-move or scrolling command, and EI goes after it.

LASER BASIC FOR THE COMMODORE 64

THE SPRITE GENERATOR PROGRAM

The Sprite Generator Program is designed to compliment Laser BASIC. Laser BASIC has commands for manipulating sprites and screen data but does not have the facility to directly design sprites. This means there are two phases to games creation. The first involves designing and editing your sprites with the sprite generator program, and the second involves writing the game itself with Laser BASIC. For those of us who are not artistically inclined, there are two sets of previously defined sprites ready to use.

To load the sprite generator from disk, type `LOAD "SPTGEN",8` after resetting the computer. Once it has loaded, type `RUN`. To load it from tape put either tape 1 or tape 2 in the recorder at side B and press `SHIFT RUN/STOP` in the normal way.

GLOSSARY

THE ARCADE SPRITE SET

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

THE DEMONSTRATION SPRITES

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

THE CHR\$ SQR

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

THE HI-RES SCREEN

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

THE CHR\$ SQR CURSOR

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

HI-RES WINDOW

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

SPRITE LIBRARY

This refers to the set of sprites you are currently working with and can contain up to 254 sprites or use 7167 bytes.

LASER BASIC FOR THE COMMODORE 64

SOFTWARE SPRITES

A sprite is a programmable graphics character. The Sprite Generator Program can develop up to 254 sprites of user selectable dimensions, up to a total of 7k. All sprite commands also apply to the screen which can be treated as a fixed dimension sprite and is given the number 0.

HARDWARE SPRITES

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

INKS and PAPER

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

THE COMMODORE KEY

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

THE FUNCTION KEYS

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

MODE:1

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

The Cursor Keys

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

The "O" and "U" Keys

These keys will <O>wnload or <U>pload between the CHR\$ square and the hi-res screen. Once you have designed your character using the cursor keys or Numerical Data Input ("N" key), typing "O" will write the contents of the CHR\$ square to the current cursor position on the hi-res screen. The current cursor position is displayed in the pane' as COL and ROW, but pressing the SPACEBAR will flash the current hi-res window.

LASER BASIC FOR THE COMMODORE 64

When "0" is pressed, data will be downloaded into the top left of this window. If ATTR (displayed in the panel and toggled by pressing "A") is zero then the current INK 3 and PAPER colours will not be downloaded with the pixel data. If ATTR is 1 then these attributes will be set in the downloaded character. Similarly, pressing "U" will upload from the hi-res screen to the CHR\$ square and if ATTR is 1 then INK 3 and PAPER will take on the values of the uploaded character from the hi-res screen. If ATTR is zero then attributes INK 3 and PAPER remain unchanged.

Note that pressing "U" will destroy whatever is currently held in the CHR\$ square and if the character square being picked up (top left of screen window) is empty then this will have the effect of clearing the CHR\$ square.

The "A" Key

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

The "E" Key

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

Once the INK (or PAPER) to be changed has been selected, a second prompt "NEW VALUE" is displayed. This should be a value in the range 0 to 15. The number input can be in decimal or hex preceded by "\$", i.e. typing \$E is equivalent to typing 14. Again, an out of range value will cause the prompt to repeat. The colours and their corresponding values are as follows:

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

The "N" Key

The 'N' key is used to enter a character as a series of 8 decimal or hexadecimal numbers. As before, hex numbers should be preceded by a "\$". Pressing "N" will cause the CHR\$ cursor to stop flashing and releasing "N" will display the prompt "BYTE 0: N" where BYTE 0 means the first of the 8 bytes making up the character in the CHR\$ square and N is the value it currently holds. If the character is blank then N

LASER BASIC FOR THE COMMODORE 64

will be zero, if not then the value will be the binary equivalent of the pattern in that byte. If you do not wish to change this value then just press ENTER otherwise enter the new value. In either case, pressing ENTER will advance to BYTE : 1 and so on through the 8 bytes. You will, of course, need to be familiar with binary arithmetic to make use of this facility.

The "C" Key

Pressing "C" simply clears the CHR\$ square.

The SPACEBAR Key

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

The COMMODORE Key

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

The LEFT SHIFT Key

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

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

The panel is updated at the completion of the change.

The "." Key

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

The "+" Key

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

MODE:2

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

The Cursor Keys

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

The "O" and "U" Keys

These keys will <D>ownload or <U>pload CHR\$ data to or from the hi-res screen but this time 4 colours are involved so the attribute flag ATTR is somewhat more significant. If the attribute flag (ATTR) is on (=1) then pressing "O" will cause the hi-res character to take on the current INK and PAPER colours, if ATTR is off (0) then attributes of the hi-res screen window will be used. Initially these will need to be

LASER BASIC FOR THE COMMODORE 64

set, so characters are normally downloaded with ATTR "ON" to begin with. In fact, it is only rarely that ATTR will be set to 0 (off) during mode 2 operations. Pressing "U" with ATTR on (=1) will lift the character from the hi-res screen together with its INK and PAPER colours and reset the current values in the panel if these were different from those of the character being uploaded. Pressing "U" with ATTR off (=0) will lift the character but display it in the character cell with the current INK and PAPER colours.

The "A" Key

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

The "E" Key

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

The "C" Key

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

The SPACEBAR Key

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

The COMMODORE Key

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

The LEFT SHIFT Key

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

The "-" Key

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

The "+" Key

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

MODE:3

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

The "W" and "I" Keys

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

The "W" and "F" Keys

LASER BASIC FOR THE COMMODORE 64

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

The "W" and "M" Keys

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

The "W" and "C" Keys

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

The "S" Key

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

The "R" Key

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

The HOME Key

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

The RIGHT SHIFT and HOME Keys

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

The "V" Key

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

The "E" Key

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

The "A" Key

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

The SPACEBAR Key

LASER BASIC FOR THE COMMODORE 64

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

The COMMODORE Key

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

The LEFT SHIFT Key

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

The "-" Key

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

The "+" Key

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

MODE:4

This mode can be thought of as dealing with data movement. Most of the operations, though not all, deal with movement between the screen and a sprite. There are also operations available in this mode to convert hires sprites into character sprites.

The "G" Key

This function is used to convert the screen window into a sprite. When "G" is pressed the screen window will cease flashing. Releasing "G" will produce the prompt "ENTER SPRITE NUMBER". If the number entered is not in the range 1 to 255 then "SPRITE PARAMETER OUT OF RANGE <CR>" will be printed. Note that <CR> means "PRESS RETURN TO CONTINUE". If the sprite number entered is the number of a sprite which has already been defined, then "SPRITE ALREADY DEFINED PRESS <CR>" will be printed.

If you wish to redefine the sprite then you will need to destroy it using the "W" key or use the more sophisticated "M" commands (see this section). The sprite created using the "G" key will have the dimensions and contents of the screen window. If ATTR is "on" it will have the attributes of the screen window, if ATTR is "off" it will have the current INK and PAPER attributes displayed in the panel. After the sprite has been created the values of SPND (the end of sprite space), SPRITE (the start address of the sprite created) and FREE MEMORY, are updated on the panel display.

The "P" Key

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

If ATTR is "on" then the sprite attributes will also be <P>ut to the screen, if ATTR is "off" then the sprite will have the attributes of the hi-res screen area that it occupies. Note that data on the hi-res screen will be overwritten by the sprite being <P>ut.

The "C" Key

LASER BASIC FOR THE COMMODORE 64

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

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

The sprite generated will have all zeros in the pixel data and have the current INKs and PAPER attributes irrespective of ATT8. The panel display is updated.

The "W" Key

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

The "I" Key

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

The "R" Key

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

Errors will be generated if:

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

The "A" Key

Toggles ATTR as in previous modes

The "M" Key

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

The seven operations available are:

LASER BASIC FOR THE COMMODORE 64

The "B" Operation

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

The "A" Operation

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

The "O" Operation

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

The "E" Operation

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

The "O" Operation

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

The "X" Operation

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

The "Y" Operation

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

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

The SPACEBAR Key

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

The COMMODORE Key

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

The LEFT SHIFT Key

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

LASER BASIC FOR THE COMMODORE 64

- II. Define a set of hires sprites the same size as the Character sprites are to be. The hires sprites must be entirely made up from one-character blocks from the character set sprite defined in (i) above.
- III. Use the "1" function to change all the hires sprites into character sprites.

The "T" function first prompts for the source hires sprite. If it does not exist, the usual error message is given. The program prompts for the destination character sprite. If this already exists, an error message is given, and function is aborted. The computer now prompts for the character set sprite number. Finally, you are prompted for the mode to be used. Press "2", "4" or "E" and RETURN for two-colour, four-colour or extended background colour modes. The way that attributes are converted depends on the mode:

two-colour mode:	the character colour attribute is the hires foreground colour.
four-colour mode:	the character colour attribute (colour no. 3) is the same as the hires colour no. 3.
extended background colour mode: colour.	the character attribute is the hires foreground. The background colour used (BG0, BG1, BG2 or BG3) is the remainder upon dividing the hires foreground colour by 4.

If at any time the computer comes across a character it doesn't recognise, it prints an error message giving the location of the character and then aborts the function. Note that in extended background colour mode, only characters 0 to 63 are recognised, instead of 0 to 255.

MODE:5

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

The "S" Key

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

The "L" Key

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

DISK COMMANDS

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

The "E" Key

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

The "R" Key

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

The "C" Key

LASER BASIC FOR THE COMMODORE 64

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

The "I" Key

Pressing will initialise the disk after an error.

The "V" Key

Pressing "V" will validate the disk.

The "0" Key

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

A SAMPLE SESSION WITH THE SPRITE GENERATOR

If you have not already loaded the Sprite Generator then load this first. Remember to turn the computer off and on first if you have already loaded Laser BASIC.

For the purposes of this sample session, the arcade sprites should be used; they should be loaded using the following procedure:

1. If you are using tape, the sprites are straight after the Sprite Generator so this tape should be placed in the cassette player.
2. Type the Commodore key and "5" to select MODE:5 Hold these two keys down until MODE:0 changes to MODE:5 on the panel display You are now in MODE:5.
3. Hold down "L" until the cursor in the CHR\$ SQR stops flashing Release "L" and the prompt "ENTER FILENAME" Will appear. Enter DEM01 and the arcade sprites will load.
4. Type the Commodore key and "0" to select MODE:0 and you are now ready to begin the session.

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

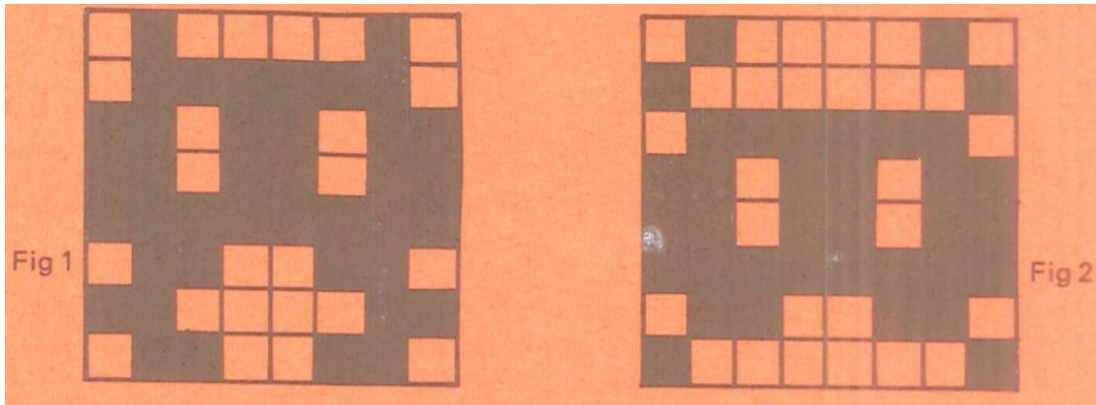
The Cursor Keys

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

In order to set a particular pixel, move the cursor to the required position, release the cursor keys and simply press "3". Move the cursor to the next position you want to set and press "3" again. To unset a pixel, position the cursor and press "4". If neither of these keys ("3" or "4") is pressed, the cursor moves non destructively. That is to say it moves around the screen without affecting the cells it moves across. If, however, you wish to draw lines of 3 or 4 pixels it is very annoying to keep releasing the cursor keys and pressing the "3" alternately. To draw a line, press "3" then the appropriate cursor keys. Once the cursor is moving "3" can be released and as long as the cursor keys are held down, a line will be drawn.

Likewise, lines can be wiped out by holding down the cursor keys in conjunction with "4". Now spend a few minutes getting used to the cursor keys by creating the invader shown here as Fig. 1.

LASER BASIC FOR THE COMMODORE 64



The "0" and "U" Keys

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

Other MODE:1 Functions

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

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

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

let's look at Fig. 1.

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

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

LASER BASIC FOR THE COMMODORE 64

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

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

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

BYTE 0 is 66?

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

BYTE 1 is 126?

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

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

The hi-res screen window can also be moved from within MODE:1. Let's move it one character to the right. First press SPACEBAR and the current screen window will flash. Whilst the SPACEBAR is held down, the cursor keys can be used to move the hi -res window around. When the cursor keys are released, the panel display will indicate the current window position. If you have moved the window correctly one character right, you should see COL:11 ROW:0. Before we download the new character let's look at the use of colours. Let's start by switching the attribute flag on. Hold down "A" until ATTR displays a 1 on the panel. Let's make this new sprite RED and BLACK.

1. Press "E" until the CHR\$ cursor stops flashing.
2. Release "E" and the prompt "ENTER INK TO BE EDITED" will appear.
3. Enter 3.
4. The prompt "NEW VALUE" will now appear. Enter 2 to make the INK RED.
5. Repeat the above but this time enter the INK TO BE EDITED as 4 and the new value as 0.
6. Press "D" to download the new character, together with it's attributes.

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

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

LASER BASIC FOR THE COMMODORE 64

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

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

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

Now design the character in Figure 3.

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

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

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

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

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

Now use "E" to edit INK 3 (currently BLUE) to become PURPLE (4). Now press "A" to switch ATTR to 0. Press "C" to clear the CHR\$ SQR and then press "U" to upload. This time the character re-appears. With the INKs and PAPER colours that are currently displayed in the pane it should now be clear that if ATTR is

LASER BASIC FOR THE COMMODORE 64

zero colours are not moved with the character but if ATTR is 1 the destination takes the attributes of the source. This is true in MODE:1 and MODE:2.

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

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

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

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

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

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

Let's now look at the fine scroll commands. In order to do this we'll first need to extend the hi-res window horizontally by one character. To do this press LEFT SHIFT and RIGHT ARROW. Let's now scroll this with wrap, right by 1 pixel. To do this press "R" and then RIGHT ARROW. Now release both keys and repeat a further 7 times until the sprite has scrolled by a full character to the right. Notice that attributes are not scrolled with the character. To return the pixel data, to its attributes press "R" and then RIGHT SHIFT and the cursor keys until the separation is completed.

Finally, let's look at another very useful function. Although the hires screen itself only occupies the top 15 ROWS, it is possible to store sprites under the text panel. Set ATTR to 0 using the "A" key. Now press "V" and use the vertical cursor keys to scroll the whole hi-res screen upward. The sprite will disappear off the top of the screen but if you keep pressing, will eventually emerge from the bottom of the hi-res screen. This covers the use of MODE3 commands, let's move on to MODE:4. Before exiting set ATTR to 0 then press RIGHT SHIFT and HOME to clear the hires screen.

LASER BASIC FOR THE COMMODORE 64

Now press the commodore key and 0 followed by the commodore key and 4 to enter MODE:4.

MODE:4 operations are chiefly concerned with sprite operations as opposed to hires window operations. Press "P" and in response to the prompt "ENTER SPRITE NUMBER" input 2. Now use the SPACEBAR and curser keys to move the hi-res window to the right of sprite 2. Press "P" again and this time enter sprite number as 3. Now move the window back to the top left of sprite 2. Use LEFT SHIFT and the curser keys to enlarge the hi-res window to embrace both sprites 2 and 3. Press "G". In response to the prompt "ENTER SPRITE NUMBER", input 2. "SPRITE ALREADY DEFINED <CR>" will appear. You have tried to redefine sprite 2. Ok let's wipe sprite 2. Press "W" and in response to "ENTER SPRITE NUMBER" type 2. Now type "G" again, enter a sprite number of 2 and this time no error will occur. Move the hi-res window down below the sprite above and press "P" entering 2, to put the new sprite (comprised of the old sprites 2 and 3) to the hi-res screen.

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

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

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

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

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

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

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

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

Press "R" until the CHR\$ SQR cursor ceases flashing and release. Enter 100 as sprite number, 10 as number in series and 30 as delay factor. Repeat this procedure with different delays.

LASER BASIC FOR THE COMMODORE 64

This completes the sample session.

FUNCTION KEY SUMMARY

MODE:1

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

MODE:2

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

MODE:3

KEYS	FUNCTIONS
W & I	Invert screen window.
W & F	Flip screen window.
W & M	Mirror screen window.
W & C	Clear screen window.
S & Cursor Key's	Fine scroll screen window without wrap.
R & Cursor Keys	Fine scroll screen window with wrap
HOME	Move hi-res window to top left of hi-res screen.

LASER BASIC FOR THE COMMODORE 64

RIGHT SHIFT HOME	Clear hi-res screen and move hi-res window to top left of hi-res screen.
V & Cursor Keys	Scroll hi-res screen vertically with wrap.
E	Enter INK/PAPER editing mode.
A	Toggle attribute flag.
SPACEBAR	Flash screen window.
SPACEBAR & Cursor Keys	Move hires window.
COMMODORE KEY	Change mode.
LEFT SHIFT & Cursor Keys	Modify hires window dimensions.
-	Fill hi-res window with current attributes.
+	Set INKs and PAPER to those of top left hi-res window.

MODE:4

KEYS	FUNCTIONS
G	Get hires window and define as a sprite.
P	Put sprite to the current hires window position.
C	Create sprite of user defined dimensions.
W	Wipe sprite and reclaim memory.
I	Interrogate sprite.
R	Run animated sequence of sprites.
A	Toggle attribute flag.
M then B	Block shift sprite.
M then A	AND sprites.
M then O	OR sprites.
M then E	XOR sprites.
M then S	Spin sprites.
M then X	XPAND sprite in X direction
M then Y	XPAND sprite in Y direction
SPACEBAR	Flash screen window.
SPACEBAR & Cursor Keys	Move hi-res window.
COMMODORE KEY	Change mode.
LEFT SHIFT & Cursor Keys	Modify hi-res window dimensions
-	Fill hi-res window with current attributes.
+	Set INKs and PAPER to those of top left hi-res window.
T	Transform a hires sprite into a character sprite.
U	Upload Commodore character set into a 16x16 sprite.

MODE:5

KEYS	FUNCTIONS
S	SAVE sprites.
L	LOAD sprites.
E	ERASE sprite file (disk only)
R	RENAME sprite file (disk on y)

LASER BASIC FOR THE COMMODORE 64

C	Copy sprite file (disk only).
L	INITIALISE DISK (disk only).
V	VALIDATE DISK (disk only).
D	List directory (disk only).
COMMODORE KEY	COMMODORE KEY Change mode.

LASER BASIC FOR THE COMMODORE 64

APPENDICES

APPENDIX A : LASER BASIC KEYWORD SUMMARY

This summary is a list of all the keywords present in Laser BASIC with a brief description of each. All the commands in Commodore BASIC are also included for completeness.

Most keywords can be abbreviated using a full stop. The abbreviation is given next to each command.

Angle brackets are used to enclose symbols, and should not be typed in.

The most commonly used are:

- <expression>
- <string>
- <parameter>
- <line number>
- <label>

The character "I" is used to mean "or". Text inside square brackets is optional, and text inside curly brackets can appear zero or more times. With a few exceptions, the keywords are divided into 5 main types:

- COMMAND**
- FUNCTION**
- CONSTANT**
- OPERATOR**
- SPRITE VARIABLE**

COMMANDS are sometimes called statements - examples are PRINT, GOTO and CLR. A FUNCTION returns a value, so it must be used in an expression. A CONSTANT can be used in place of a number. OPERATORS are placed between strings or numbers in an expression. For details of SPRITE VARIABLES, refer to [SPRITE VARIABLES](#).

In many of the graphics commands, sprite variables are used as parameters. In this case, if the parameters are missed out, the existing values are used. See section [WHY ARE SPRITE VARIABLES USED?](#) for further details.

A shorthand notation is used when describing logical operations on windows and sprites. (Refer to section [THE CONCEPT OF SPRITE WINDOWS](#) for details of sprite windows):

Short Hand	Means
S1	the whole of sprite no. SPN
S2	the whole of sprite no. SPN2
W1	a window inside sprite no. SP at position COL,ROW and dimensions WID, HGT
W2	a window inside sprite no. SPN2, at position COL2,ROW2 and dimensions WID,HGT
->	is copied to

LASER BASIC FOR THE COMMODORE 64

AND	ANDed with
OR	means "ORed with"
XOR	means "XORed with"

Example: W1 XOR W2 -> W1 would be read as "A window Inside sprite no.SPN, at position COL,ROW and dimensions WID,HGT, XORed with a window inside sprite no. SPN2, at position COL2,ROW2 and dimensions WID,HGT is copied to a window inside sprite no. SPN2 at position COL2,ROW2 and dimensions WID,HGT."

As you can see, the abbreviated version is somewhat more concise

	Abbr	Description
.BLUE	.B.	CONSTANT: returns value of 14.
.GREEN	.G .	CONSTANT: returns value of 13.
. RED	.R.	CONSTANT: returns value of 10.
ABS<expression>	AB.	FUNCTION: return absolute value.
ADSR voice, attack, decay, sustain, release	AD.	COMMAND: specify envelope.
AFA(SPN)		FUNCTION: returns attribute field address of a sprite. -1 if non-existent Sets WID and HGT to size of sprite.
AFA2(SPN)	AF.	FUNCTION: returns secondary attribute field address of a sprite. -1 if non-existent. Sets WID and HGT to size of sprite.
ALLOCATE <expression>,<expression>	AL.	COMMAND: allocates memory, in bytes, for background tasks 1 and 2.
AND	A.	OPERATOR: bitwise logical AND.
AND%AND SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2	AND%A.	COMMAND: W1 AND W2 -> W1 ; W1 AND W2 -> W2
AND%BLK SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2	AND%.	COMMAND: W1 AND W2 -> W1 ; W1 -> W2
AND%OR SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2	AND%O.	COMMAND W1 AND W2 -> W1 ; W1 OR W2 -> W2
AND%XOR SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2	AND%X.	COMMAND: W1 AND W2 -> W1 ; W1 XOR W2 ->W2
ASC (<string>)	AS.	FUNCTION: return ASCII code of first character in string.
ATN (<expression>)	AT.	FUNCTION arctangent.
ATR		SPRITE VARIABLE: current attribute value.
ATT20N	ATT2.	COMMAND turn on the use of both sets of attributes.
ATTDN SPN, COL, ROW, WID, HGT	ATTD.	COMMAND: scroll down attributes in sprite window by one-character block.
ATTGET SPN, COL, ROW	ATTG	COMMAND: retrieve attribute from sprite and put it in A""M.
ATTL SPN, COL, ROW, WID, HGT		COMMAND: scroll attributes in window left one character

LASER BASIC FOR THE COMMODORE 64

ATTOFF	ATTOF.	COMMAND: turn off the use of attributes.
ATTON	ATTO.	COMMAND: turn on the use of primary attributes only.
ATTR SPN, COL, ROW, WID, HGT	ATT.	COMMAND: scroll attributes in window right one character.
AUTO [<step size>]	AU	COMMAND: turn on auto line numbering. Omit step size to turn Of .
ATTUP SPN, COL, ROW, WID, HGT	ATTU.	COMMAND: scroll attributes in window up one character
BG0 <colour>	BG.	COMMAND: set background colour no. 0 in extended colour text mode
BG1 <colour>		COMMAND: set background colour no. 1 in extended co our text mode
BG2 <colour>		COMMAND: set background colour no. 2 in extended colour text mode
BG2 <colour>		COMMAND: set background colour no. 2 in extended colour text mode
BGND <colour>	BGN.	COMMAND: set up ATRL for background colour in 2-colour mode.
BLACK	BLA.	CONSTANT return value of zero.
BLK%AND SPN, COL, ROW, WID, HGT, SPN2, COL2. ROW2	BLK%A.	COMMAND' W2 -> W1 ; W1 AND W2 -> W2
BLK%BLK SPN COL, ROW, WID, HGT, SPN2, COL2, ROW2	BL.	COMMAND: W2 -> W1 ; W1 -> W2
BLK%OR SPN COL, ROW, WID, HGT, SPN2, COL2, ROW2	BLK%O.	COMMAND: W2 -> W1 ; W1 OR W2 ->> W2
BLK%XOR SPN,COL.ROW, WID, HGT,SPN2.COL2.ROW2	BLK%X.	COMMAND: W2 -> W1 ; W1 XOR W2 -> W2
BLUE	BLU.	CONSTANT return value of 6.
BOX SPN,ROW.COL.WID.HGT	B.	COMMAND: set a rectangular block of pixels in a sprite. Colour used depends on MODE.
BROWN	BR.	CONSTANT return value of 9.
CASE <expression>	CA.	COMMAND: used In conjunction with OF and CASEND to select between several courses of action.
CASEND	CA.	COMMAND: denotes end of a CASE statement.
CCOL	CC.	SPRITE VARIABLE: collision column.
CELSE	CE.	COMMAND: analogous to 'ELSE'; used with CIF and CEND.
CEND	CEN.	COMMAND: end of a multi-line IF statement.
CFN <label> [<parameter> {,<parameter> }]	CF.	FUNCTION: calls a multi-line function.
CGET SPN, COL, ROW	CG.	COMMAND: GET a character sprite from the text screen a COL. ROW
CHAR SPN, COL, ROW,NUM	CHA.	COMMAND: place character no. NUM inside sprite at COL,ROW. Various options are available by adding a number to NUM: See table .

LASER BASIC FOR THE COMMODORE 64

CHR\$ <expression>	CH.	FUNCTION: converts ASCII code into a one-character string.
CIF <expression>	CI.	COMMAND: the commands up to the next CELSE or CEND are only executed if the <expression> is non-zero.
CLOSE <file number>	CLO.	COMMAND: closes a file.
CLR	CL.	COMMAND: clears BASIC variables and stack.
CMD file number [<string>]	CM.	COMMAND. switches output from the screen to the specified file. The <string> when specified is sent to the file.
COL		SPRITE VARIABLE: column within sprite.
COL2	COL.	SPRITE VARIABLE: column within second sprite.
COMMON%<expression>	COM.	ARRAY: this is a pseudo-integer array that can be accessed by all three concurrent tasks. It contains 64 elements.
CONT	C.	COMMAND: Re-starts execution of a program after a STOP, or use of the RUN/STOP key.
CONV SPN COL,ROW,SPN2		COMMAND: Convert software sprite at COL,ROW into hardware sprite definition no SPN2.
COS <expression>		FUNCTION cosine.
CPUT SPN COL,ROW	CPU.	COMMAND. put character sprite onto the text screen at COL,ROW.
CPYAND SPN, SPN2	CPYA.	COMMAND. S1 AND S2 -> S2
CPYBLK SPN, SPN2	CP.	COMMAND: S1 -> S2
CPVOR SPN, SPN2	CPYO.	COMMAND: S1 OR S2 -> S2
CPYXOR SPN, SPN2	CPYX.	COMMAND: S1 XOR S2 -> S2
CROW	CR.	SPRITE VARIABLE: collision row.
CSPRITE SPN, WID, HGT	CS.	COMMAND: create character sprite of size WID,HGT.
CSWAP SPN, COL, ROW	CSW.	COMMAND: exchange character sprite with the screen at position COL,ROW.
CUTOFF <frequency>	CU.	COMMAND: set cutoff frequency (0 .. 65535) for sound filter
CYAN	CY.	CONSTANT: returns value of 3.
DATA <constant> {< constant> }	D.	COMMAND: list of constants for READ statements.
DBLANK	DB.	COMMAND: blank the screen.
DEEK <expression>	DEE.	COMMAND: two-byte PEEK.
DEF FN <name><variable>=<expression>		COMMAND: set up a user-defined function (numeric only)
DFA (SPN)		FUNCTION: returns data field address of a sprite. Sets WID and HGT to the size of the sprite.
DI		COMMAND: temporarily disable multi-tasking.
DIM <variable><subscript> {,<subscript> }, {<variable><subscript> {.<subscript>} }}	D1.	COMMAND: Dimensions an array of variables.

LASER BASIC FOR THE COMMODORE 64

DIR [<string> ,<device number>]		COMMAND: prints disk directory. Default <string> is "\$" and default <device number> is 8.
DISABLE	DIS.	COMMAND: disables RUN/STOP key.
DLOAD <filename> [,<device number>]	DL.	COMMAND: load a program from disk. Default <device number> is 8.
DMERGE <filename> [,<device number>]	DM.	COMMAND: merges a sprite file with sprites already in memory. Default device number is 8.
DOKE <location>.<expression>	DO.	COMMAND 2 byte POKE.
DRAW SPN,COL ROW,COL2,ROW2	DR.	COMMAND: draw a line in sprite SPN from COL,ROW to COL2,ROW2.
DRECALL <filename> [.<device number>]	DRE.	COMMAND: load in new sprites from disk.
DSAVE <filename> [.<device number>]	DS.	COMMAND: save a BASIC program to disk. Default device number is 8.
DSHOW	DSH.	COMMAND~ re-enable screen display (after DBLANK).
DSTORE <filename> [.<device number>]	DST.	COMMAND: save all sprites to disk. Default device number is 8.
DTCTOFF	DTCTOF.	COMMAND: turn off software sprites collision detection.
DTCTON	DT.	COMMAND: turn on software sprites collision detection.
EBACK	EB.	COMMAND: puts display into extended background colour mode.
EI		COMMAND: re-enable multi-tasking (after 01).
ELSE	EL.	ELSE is an extension to BASIC's IF ... THEN statement. If the condition is false, the statements after ELSE are executed. If it is true, the statements between THEN and ELSE are executed.
END	E.	COMMAND: finishes a program's execution.
ENV		FUNCTION: returns the output from oscillator 3's envelope generator (0 .. 255).
EXIT	EXI.	COMMAND: leaves a loop prematurely. Can be used with FOR-NEXT, REPEAT- UNTIL or WHILE-WEND.
EXP <expression>	EX.	FUNCTION: exponential (2.71828183 raised to the power of the expression).
EXX SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2		COMMAND: expand the window in sprite no. SPN into a second window twice as wide in sprite SPN2.
EXY SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2		COMMAND: expand the window in sprite no. SPN into a second window twice as high in sprite SPN2.
FALSE	FA.	CONSTANT return value of 0.
FGND <colour>	FG.	COMMAND: set ATR so that the foreground in two-colour hires mode is the colour specified.

LASER BASIC FOR THE COMMODORE 64

FILL SPN, COL, ROW, NUM		COMMAND: fill in an area of sprite SPN, starting at COL,ROW with the current colour set by MODE. In four-colour mode, NUM specifies the border colour over which filling cannot proceed. See section FILLING IN BLANK AREAS (FILL) for further details
FIRE1	FI.	FUNCTION: Returns TRUE if the fire button on joystick no. 1 is pressed.
FIRE2		FUNCTION: Returns TRUE if the fire button on joystick no. 2 is pressed.
FLIP SPN, COL, ROW, WID, HGT		COMMAND: Turn the sprite window upside-down.
FLIPA SPN, COL, ROW, WID, HGT	FL.	COMMAND: turn the attributes in the sprite window upside-down.
FN <name>(<expression>)		FUNCTION: references a function defined using DEF FN.
FOR <variable>=<start> TO <limit> [STEP<increment>]	F.	COMMAND Specifies the start of a loop in which a variable is used as a counter. The loop ends with a NEXT statement.
FRE <expression>	FR.	FUNCTION returns the number of bytes free for the BASIC program and
FRO <voice>,<frequency>		COMMAND: sets the frequency (0 .. 65535) the specified voice (1 .. 3).
GET <variable> {,<variable> }	GE.	COMMAND: reads keys presses typed by the user into a variable.
GET# <file number>,<variable> {,<variable>}		COMMAND: reads characters one at a time into variables from the fie specified.
GETAND SPN, COL, ROW		COMMAND: copy hires screen at COL,ROW into sprite no. SPN, ANDing with sprite pixels.
GETBLK SPN, COL, ROW	GETB.	COMMAND: copy hires at COL,ROW into sprite no. SPN.
GETOR SPN, COL, ROW	GETO.	COMMAND: copy hires sprite screen at COL,ROW into sprite no. SPN, ORing with existing sprite pixels.
GETXOR SPN, COL, ROW	GETX.	COMMAND: copy hires screen at COL,ROW into sprite no. SPN, XORing with existing sprite pixels.
GOSUB <label> <line number>	GOS.	COMMAND: call a subroutine at the specified label or me number.
GOTO <label> <line number>	G.	COMMAND: jump unconditionally to the specified label or line number
GRAY1	GRA.	CONSTANT: returns a value of 11.
GRAY2		CONSTANT: returns a value of 12.
GRAY3		CONSTANT: returns a value of 15.
GREEN	GR.	CONSTANT: returns a value of 5.
H1COL <colour>	H1.	COMMAND: sets multicolour sprite colour no. 1.
H2COL <sprite no.>	H2.	COMMAND: puts a hardware sprite into two-colour mode.

LASER BASIC FOR THE COMMODORE 64

H38COL	H3.	COMMAND: put display into 38-column mode.
H3COL <colour>	H3C.	COMMAND: sets multicolour sprite no. 3.
H40COL <sprite number>	H4.	COMMAND: Put display into 40-column mode.
H4COL <sprite number>	H4C.	COMMAND: put a hardware sprite into four-colour mode.
HALT [<task number>]	HA.	COMMAND: stop execution of the specified background task (this can be 1 or 2). If the task number is not specified, the task which is currently running stops itself
HBORDER <colour>	HB.	COMMAND sets the border colour used in hires mode.
HCOL <sprite number>	HC.	COMMAND: set a hardware sprite's colour.
HEX\$ <expression>	H.	FUNCTION: converts a number into a string containing its hexadecimal representation.
HEXX <sprite number>		COMMAND: expand a hardware sprite by a factor of 2 In the X-direction.
HEXY <sprite number>		COMMAND: expand a hardware sprite by a factor of 2 n the Y -direction
HGT	HG.	SPRITE VARIABLE: height of window in first sprite.
HIRES	HI.	COMMAND: go into hires mode.
HIT <sprite number>		FUNCTION: returns TRUE if the sprite has hit another sprite. Add 8 to detect sprite to background collisions. Add 16 to use collisions read by previous HIT command.
HOFF <sprite no.>	HOF.	COMMAND: turn off a hardware sprite.
HON <sprite no.>	HO.	COMMAND: turn on a hardware sprite.
HPAPER <colour>	HP.	COMMAND: sets the paper (background) colour used n four-colour hires mode.
HSET <sprite no.>,<definition no.>	HS.	COMMAND: associates a hardware sprite with its definit on.
SHX <sprite no.>	SH.	COMMAND: shrink a hardware sprite to its normal size In the X-direction.
SHY <sprite no.>		COMMAND: Shrink the hardware sprite to normal size in the Y -direction.
HX <sprite no >,<position>		COMMAND: set up a hardware sprite's X-position on the screen.
IF <expression> <statement> I		COMMAND: the course of action taken depends on the value of the <expression>.
IF <expression> THEN <line number> I		If It is TRU E, the statements after THEN, and up to ELSE (if It exists) are executed.
IF <expression> THEN <statement> {<statement>}		Otherwise, if there is no ELSE the rest of the line is ignored, and
[ELSE <statement> {<statement>}]		If there is an ELSE, the statements after it are executed.
IMPORT <task number>,<expression>	IM.	FUNCTION. re urns the value of the expression as evaluated by the specified task
INC		SPRITE VARIABLE: inclination of polygon.

LASER BASIC FOR THE COMMODORE 64

INIT	INI.	COMMAND: initialises graphics.
INK <colour>		COMMAND: set up colour used when printing on the screen.
INPUT ["prompt";] <variable> {,<variable> }		COMMAND: input values to the specified variables from the keyboard.
INPUT# <file number>,<variable>,<variable>	I.	COMMAND: input variables from the specified device.
INT <expressions>		FUNCTION: returns the value of the expression rounded down to the nearest integer.
INV SPN, COL, ROW, WID, HGT		COMMAND: invert all the pixels in the specified sprite window.
JS1	J.	FUNCTION: returns the direction of joystick no. 1 (0 .. 8).
JS2		FUNCTION: returns the direction of joystick no. 2 (0 .. 8).
KB <expression>		FUNCTION: returns TRUE if the specified key is pressed (Each key has a number from 0 to 63).
KEYOFF	KEYOF.	COMMAND: disable keyboard scanning.
KEYON	KE.	COMMAND: re-enable keyboard scanning (after KEYOFF).
LABEL <name>[[[VAR]<parameter> {,[VAR]<parameter> }]]	LA.	COMMAND: defines a label for use by GOTOs, GOSUBs and RESTOREs, or a procedure or multi-line function. In the former case, LABEL is only followed by the <name>.
INCASE	LC.	COMMAND: go into lower case.
LEFT\$ <string>,<integer>	LEF.	FUNCTION: returns a string consisting of the leftmost <Integer.> characters of the <string>.
LEN <string>		FUNCTION: returns the number of characters in the string.
LET <variable>=<expression>	L.	COMMAND: assigns a value to a variable. The word LET can be missed out.
LIST [[<first line>],[<last line>]]	LI.	COMMAND: list lines of the BASIC program.
LOAD [<filename>,<device>]	LO.	COMMAND: Load a BASIC program from tape. The default device number is 1
LOCAL <variable> {,<variable> }	LOC.	COMMAND: Used inside a procedure or multi-line function to create local variables which are independent of those in the main program.
LOG <expression>		FUNCTION: natural logarithm.
LORES	LOR.	COMMAND: Go into lores (text) mode.
LPX	LP.	FUNCTION: return X-position of lightpen.
LPY		FUNCTION: return Y-position of lightpen.
MAR SPN, COL, ROW, WID, HGT	MA.	COMMAND: mirror attributes left-to-right in the sprite window.
MCO1 <colour>	MC.	COMMAND: set ATR so that colour no. 1 in four-colour mode is the colour specified.
MCO2 <colour>		COMMAND: set ATR so that colour no. 2 in four-colour mode is the colour specified.

LASER BASIC FOR THE COMMODORE 64

MCOL3 <colour>		COMMAND: setATR so that colour no. 3 in four-colour mode is the colour specified.
MERGE [<filename>,<device no.>]]	ME.	COMMAND: merge a sprite file on tape with those already in memory. The default device number is 1.
MID\$ <string>,<start>[,<length>])	M.	FUNCTION: returns a string consisting of <length> characters from <string>, starting at character no. <start>. If the <length> is missed out, MID\$ gives all the characters from <start> onwards.
MIR SPN, COL, ROW, WID, HGT		COMMAND: mirror the sprite window, left-to-right.
MODE <expression>		COMMAND: sets the colour used for plotting points. (4+Iinvert).
MONO	MON.	COMMAND: Go into two-colour mode.
MOVAND SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2	MOVA.	COMMAND: W1 AND W2 -> W2
MOVATT SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2	MOVAT.	COMMAND: move attributes from window 1 to window 2.
MOVBLK SPN, COL, ROW, WID, HGT. SPN2, COL2, ROW2	MOV.	COMMAND: W1 -> W2
MOVE SPN, COL, ROW		COMMAND: sets hardware sprite no. SPN moving automatically across the screen. Every 50th of a second, it is moved by COL pixels in the X-direction and ROW pixels in the Y-direction.
MOVOR SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2	MOVO.	COMMAND. W1 OR W2 -> W2
MOVXOR SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2	MOVX.	COMMAND W1 XOR W2 -> W2
MULTI	MU.	COMMAND: go into multi-colour mode.
MUSIC <voice>.<length>	MUS.	COMMAND: sound a note - length is in 60ths of a second.
MUTE <flag>	MUT.	COMMAND: enables muting of voice 3 if <flag> <>0, disables if <flag> = 0.
NEW		COMMAND' delete the current BASIC program and all variables.
NEXT [<variable>] {,<variable> }	N.	COMMAND: used with FOR to establish the end of a FOR-NEXT loop.
NOISE <voice> abbr:	NO!.	COMMAND: set up a voice (1,2 or 3) to generate white noise.
NORM	NOR.	COMMAND Disable switching of attribute fields (after SWITCH).
NOT	NO.	OPERATOR: logical NOT.
NUM	NU.	SPRITE VARIABLE: number of sides/ pixels.
OF <expression>		COMMAND: this is used in conjunction with CASE and CASEND. If the <expression> after OF is the same as the <expression> after CASE the commands up to the next OF or CASEND are

LASER BASIC FOR THE COMMODORE 64

		executed. OF OR results on the execution of the following statements only if none of the OFs so far have been executed.
OLD	OL.	COMMAND: recover a NEWed program.
ON <expression> GOTO I GOSUB I RESTORE <line 'lo.> I <label>{'<Line no.> I <label> }	O.	COMMAND: the ON statement is used for GOTO, GOSUB or RESTORE to
OPEN <file number>[,<device>[.<address>[.<file name>]]]	OP.	COMMAND: Opens a channel for input or output to a peripheral device.
OR		OPERATOR: logical OR.
OR%BLK SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2	OR%.	COMMAND: W1 OR W2 -> W1 ; W1 -> W2
OR%OR SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2	OR%O.	COMMAND: W1 OR W2 -> W1 ; W1 OR W2 -> W2
OR%XOR SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2	OR%X.	COMMAND W1 OR W2 -> W1 : W1 XOR W2 -> W2
ORANGE	ORA.	CONSTANT returns value of 8.
OSC	OS.	FUNCTION: returns output of voice 3's oscillator (0..255).
OVER	OV.	COMMAND' gives the screen priority over the specified hardware sprite.
PASS	PA.	COMMAND set sound filter to low pass (0), high pass (1), band pass (2) or notch reject (3) mode.
PEEK <location>	PE.	FUNCTION: returns an integer from 0 to 255 which is read from the specified memory location.
PI		CONSTANT: returns value of 3.1415926536.
PLAY	PLA.	COMMAND; play music from sprite no. SPN on voice 1, sprite COL on voice 2 and sprite ROW on voice 3.
PLAY	PLA.	FUNCTION: returns TRUE if music is still playing under interrupt.
PLOT SPN, COL, ROW	PL.	COMMAND: sets a point at co-ordinate COL,ROW inside sprite number SPN.
POINT (SPN COL,ROW)	POI.	FUNCTION: interrogate a pixel within a sprite.
POKE <location>,<value>	P.	COMMAND: writes a one-byte value into a memory location.
POLY SPN, COL, ROW, WID, HGT, NUM, INC	POL.	COMMAND: draw a polygon with NUM sides and inclination INC
POS <expression>		FUNCTION: returns the current position on the logical screen line which is 0 to 79. The argument is ignored.
PRINT [@<column>,<row>] {; I ,I <expression>}	?	COMMAND: print data items to the screen . CMD may be used to direct this to any other device.
PRINT# <file number>, {; I ,I <expression> }	PR.	COMMAND: print data items to the specified output file.

LASER BASIC FOR THE COMMODORE 64

PROC <label>[<parameter> {,<parameter> }]]		COMMAND: calls a procedure, using the specified parameters, if any
PROCEND	PRO.	COMMAND: end of procedure.
PULL	PU.	COMMAND: remove one level of subroutine nesting from the stack.
PULSE <voice>,<width>	PULS.	COMMAND: set up specified voice to generate a pulse waveform <width> can be from 0 to 4095.
PURPLE	PUR.	CONSTANT: returns a value of 4.
PUTAND SPN, COL, ROW	PUTA.	COMMAND: put a sprite on the hires screen at position COL,ROW, ANDing with the existing screen pixels.
PUTBLK SPN, COL, ROW	PUT.	COMMAND: put a sprite on the hires screen at position COL, ROW, overwriting the ,existing pixels.
PUTCHR SPN, COL, ROW, NUM	PUT.	COMMAND: Copy a character from COL,ROW inside sprite no. SPN into character set definition no. NUM.
PUTOR SPN COL,ROW	PUTO.	COMMAND: put a sprite on the hires screen at position COL,ROW, ORing with the existing screen pixels.
PUTXOR SPN, COL, ROW	PUTX.	COMMAND: put a sprite on the hires screen at position COL,ROW, XORing with the existing screen pixels.
RASTER <scan line>	RA.	COMMAND: synchronise the next graphics command to the specified video scan line.
READ <variable> {,<variable> }	R.	COMMAND: read values into variables from constants in DATA statements.
RECALL [<filename>[.<device>]]	REC.	COMMAND: load in sprites from tape. Default device number is 1.
RED		CONSTANT: Returns a value of 2.
REM		COMMAND: Ignore the rest of this line (used for comments).
RENUM [<first line>[.<step size>]]	REN.	COMMAND: renumber all line numbers in the program, including GOTOs, GOSUBs and RESTOREs. The default values for the first line number and step size are both 10.
REPEAT	REP.	COMMAND: Used at the top of a loop in conjunction with UNTIL when it is required to execute the loop at least once.
RESEQ		COMMAND: renumber sprites.
RESERVE <expression>	RESER.	COMMAND: reserve storage for sprites. Minimum is 7168 bytes.
RESET	RESE.	COMMAND: erase all sprites from memory.
RESONANCE <value>	RESO.	COMMAND: specify resonance of sound filter (0 to 15).

LASER BASIC FOR THE COMMODORE 64

RESTORE [<line number> <label>]	RES.	COMMAND: resets current data item used by READ to specified line number. or top of program if line is not specified.
RETURN	RET.	COMMAND: return from a subroutine.
RIGHT\$ <string>.<number>	RI.	FUNCTION: returns a string consisting of the rightmost < number> characters of the <string>.
RING <voice>.<flag>	RIN.	COMMAND: enables ring modulation of a voice if the <flag> is non-zero disables it if the <flag> is zero.
RND <expression>	RN.	FUNCTION: generates a random number between 0 and 1.0. If the argument is positive, a pseudo-random number is generated from the last number. If the argument is zero. the random number comes from the computer's hardware clock, and if it is negative. a new 'seed' is used for each function call.
ROW		SPRITE VARIABLE: row inside first sprite.
ROW2	RO.	SPRITE VARIABLE: row inside second sprite.
RPLAY SPN, COL, ROW	RPL.	COMMAND: play music repeatedly from sprite no. SPN on voice 1, sprite no COL on voice 2 and sprite no. ROW on voice 3.
RPT <expression>.<scrolling command>	RP.	COMMAND. execute a scrolling command <expression> times.
RSYNC <scan line>	RS.	COMMAND· synchronise all following graphics commands to the specified screen scan line.
RUN [<line number>]	RU.	COMMAND· clear all BASIC variables and start executing the program in memory, from the first line, or the line number specified.
S2COL	S2.	COMMAND: puts the Laser BASIC software into two-colour mode.
S4COL	S4.	COMMAND: puts the Laser BASIC software into four-colour mode.
SAVE [<filename>[.<device>]]	SA.	COMMAND· save a BASIC program to tape. The default device number is 1
SAW <voice>		COMMAND: set a voice (1,2 or 3) to produce a sawtooth wave.
SCAN (SPN, COL, ROW, WID, HGT)	SCA.	FUNCTION: returns TRUE if the sprite window contains any non background pixel data.
SCL 1 SPN, COL, ROW, WID, HGT		COMMAND: scroll the sprite window left by one pixel without wrap.
SCL2 SPN, COL, ROW, WID, HGT		COMMAND: scroll the sprite window left by two pixels without wrap
SCLB SPN, COL, ROW, WID, HGT		COMMAND: scroll the sprite window left by eight pixels (one character block) without wrap.
SCLR SPN, ATR	SC.	COMMAND: clear a sprite, using the specified attribute.

LASER BASIC FOR THE COMMODORE 64

SCR1 SPN, COL, ROW, WID, HGT		COMMAND: scroll the sprite window right by one pixel without wrap
SCR2 SPN, COL, ROW, WID, HGT		COMMAND: scroll the sprite window right by two pixels without wrap.
SCRB SPN, COL, ROW, WID, HGT		COMMAND: scroll the sprite window right by eight pixels (one character block) without wrap.
SCROLL SPN, COL, ROW, WID, HGT, NUM	SCRO.	COMMAND: scroll the sprite window vertically by NUM pixels. Positive values of NUM scroll up, negative values scroll down.
SCRX <offset>	SCA.	COMMAND: shift entire screen right by <offset> pixels. Normal <offset:> value is zero.
SCRY <offset>SCA.		COMMAND: shift entire screen down by <offset> pixels. Normal <offset> value is 3.
SETA SPN, COL, ROW, WID, HGT, ATR		COMMAND: set all attributes in the sprite window to ATA.
SET TR <colour>,<colour>.<colour>	SE.	COMMAND: set up attribute value. This command is never used in Laser BASIC.
SFRE	SF.	FUNCTION: returns number of bytes left free for sprites.
SGN <expression>	SG.	FUNCTION: returns sign of argument: -1 if negative. 0 if zero and 1 if positive.
SIDCLR	SID.	COMMAND: reset sound chip.
SIN <expression>	SI.	FUNCTION: sine.
SIZE <array name>.<dimension>	SIZ.	FUNCTION: returns the size of an array in the specified dimension. If <'dimension'+0 the number of dimensions is returned.
SPC <expression>	SP.	Used inside a PRINT statement to print the specified number of spaces.
SPIN SPN.COL.ROW.WID, HGT. SPN2. COL2. ROW2	SPI.	COMMAND: rotate window 1 by 90 degrees clockwise into window 2.
SPN		SPRITE VARIABLE: first sprite number.
SPN2	SPN.	SPRITE VARIABLE: second sprite number.
SPRITE SPN, WID, HGT	SPR.	COMMAND: create a hires sprite of size WID.HGT.
SQR <numeric>	SQ.	FUNCTION: square root.
STOP	S.	COMMAND: stop execution of the program and return to direct mode.
STORE [<filename>,<device>]]	STOR.	COMMAND: save sprites to tape. Default device number is 1.
STR\$ <expression>	STR.	FUNCTION: converts the numeric argument into its string representation.
SWAPATT SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2	SW.	COMMAND: swap attributes between the two windows.
SWITCH	SWI.	COMMAND: switch over attribute fields.
SYNC <voice>,<flag>	SYN.	COMMAND: enable synchronisation on the specified voice if <flag> is non-zero or disable it if <flag> is zero.

LASER BASIC FOR THE COMMODORE 64

SYS <location>	SY.	COMMAND: call a machine code subroutine.
TAB <expression>	T.	Used within a PRINT statement to move the cursor to the specified position on the screen.
TAN <expression>		FUNCTION: tangent.
TASK <task number>,<label>	TAS.	COMMAND: start execution of a background task at the specified line.
TBORDER <colour>	TB.	COMMAND: set the border used in the text modes.
TEXT SPN, COL, ROW,<text string>.<offset>	E.	COMMAND: place a string of text inside sprite SPN at COL, ROW. Different values of <"offset"> can be used to give different text styles. See table Offset Table .
TPAPER <"colour">	TP.	COMMAND set hardware sprite no. SPN2 moving under interrupt, using data from sprite no. SPN.
TRACK <sprite no.>	TRA.	FUNCTION: returns TRUE if the specified hardware sprite is still moving under interrupt.
TRI <voice>		COMMAND set a voice to generate triangular waves.
TRUE	TR.	CONSTANT returns a value of -1.
UCASE	UC.	COMMAND. go into upper case.
UNDER <sprite number>	UND.	COMMAND: makes the screen go under the specified hardware sprite.
UNSYNC	UNS.	COMMAND· disable synchronisation to display scan (after RSYNC).
UNTIL	UN.	COMMAND: this is put at the end of a REPEAT - UNTIL loop. Execution of the loop repeats until the <expression> is non-zero.
USR <expression>	U.	FUNCTION: user-defined function; starting address of machine code subroutine held in 785-786.
VAL <string>	VA.	FUNCTION: returns the numeric value of the data in the <string>.
VERIFY [<filename>[.<device>]] V.	COMMAN D: verify a BASIC program held on tape.	
VOLUME <number>	VO.	COMMAND: set sound master volume (0 to 15).
WAIT <location>.<mask1 >[.<mask2>]		COMMAND: repeatedly takes the contents of memory <location>, ANDs with <mask1 > and XORs with <mask2> until it is non-zero. The default value of <mask2> is zero.
WCLR SPN, COL, ROW, WID, HGT, ATR	WC.	COMMAND: clear the sprite window, including attributes.
WEND	WE.	COMMAND: marks the end of a WHILE loop.

LASER BASIC FOR THE COMMODORE 64

WHILE <expression>	WH.	COMMAND: used at the top of a loop which is to be executed zero or more times. The loop is only executed while the <expression> is non-zero
WHITE	WHIT.	CONSTANT: returns a value of 1.
WID	WI.	SPRITE VARIABLE: width of first sprite window.
WINDOW <expr>	WIN.	COMMAND: set up hires window on the text screen.
WIPE SPN	WIP.	COMMAND: delete sprite.
WRAP SPN, COL, ROW, WID, HGT, NUM	WRA.	COMMAND: scroll window vertically with wrap by NUM pixels. Positive values of NUM scroll up, negative values scroll down.
WRL 1 SPN, COL, ROW, WID, HGT	WRL.	COMMAND: scroll window left by one pixel with wrap.
WRL2 SPN, COL, ROW, WID, HGT		COMMAND: scroll window left by two pixels with wrap.
WRL8 SPN, COL, ROW, WID, HGT		COMMAND: scroll window left by eight pixels (one-character block) with wrap.
WRR1 SPN, COL, ROW, WID, HGT	WR.	COMMAND: scroll window right by one pixel with wrap.
WRR2 SPN, COL, ROW, WID, HGT		COMMAND: scroll window right by two pixels with wrap.
XOR%AND SDN, COL, ROW, WID, HGT, SPN2, COL2, ROW2	XOR%A.	COMMAND: W1 XOR W2 -> W1 ; W1 AND W2 -> W2
XOR%BLK SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2	X.	COMMAND: W1 XOR W2 -> W1 ; W1 -> W2
XOR%OR SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2	XOR%O.	COMMAND: W1 XOR W2 -> W1 ; W1 OR W2 -> W2
XOR%XOR SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2	XOR%X.	COMMAND: W1 XOR W2 -> W1 ; W1 XOR W2 -> W2

Display Table

Display	Add
ASCII	0
reverse ASCII	256
display code	512
double width ASCII	1024
reverse double width ASCII	1280
double width display code	1536

Offset Table

<offset>	Display Result
0	normal
256	inverse
1024	double-width
1280	double-width inverse

LASER BASIC FOR THE COMMODORE 64

2048	double-spaced
2304	double-spaced inverse

APPENDIX B: ERROR MESSAGES

This list only contains error messages which are not part of Commodore BASIC.

Error	Description
?until error	An UNTIL command was found without a corresponding REPEAT.
?wend error	A WEND command was found without a corresponding WHILE.
?exit error	An EXIT command was found which is not inside a loop.
?local error	A LOCAL command was found which is not at the beginning of a procedure or function definition.
?procend error	A PROCEND command was found where a procedure has not been called.
?out of memory error	There is insufficient sprite space left.
?sprite error	The sprites present in memory are corrupted. Normally you should type RESET after this error report.
?redefd sprite error	You have attempted to re-define a sprite which already exists.
?no such sprite error	You have attempted to use a sprite which doesn't exist.
?wipe error	You have tried to delete the hires or text screens (sprites 0 or 255).
?type mismatch error	You have tried to reference the pixel data in a character sprite.

LASER BASIC FOR THE COMMODORE 64

APPENDIX C : DIFFERENCES BETWEEN BASICS

LIGHTNING AND LASER BASIC

Laser BASIC is an updated version of an older program, Basic Lightning. It can run most programs written in Basic Lightning. The major differences are

1. Some of the command names have changed:

BASIC LIGHTNING	LASER BASIC
3CRLX	SCRX
SCRLY	SCRY
XPANDX	EXX
XPANDY	EXY
SPRCONV	CONV
.ON	HON
.OFF	HOFF
.SET	HSET
.4COL	H4COL
.2COL	H2COL
.COL0	H1COL
.COL1	H3COL
XPANDX	HEXX
SHRINKX	HSHX
XPANDY	HEXY
SHRINKY	HSHY
.XPOS	HX
.YPOS	HY
.COL	HCOL
.OVER	OVER
.UNDER	UNDER
STRPLOT	TEXT

This should not cause any problems, since the command names used in a Basic Lightning program will change automatically when the program is loaded into Laser BASIC.

2. Some new commands have been added:

AUTO	CSPRITE	CSWAP	EBACK
BG2	NORM	MCOL1	FGND
DI	RSYNC	RENUM	CPUT
FILL	BGO	BG3	KEYON
MCOL2	BGND	INIT	SFRE
CGET	RASTER	BG1	SWITCH
KEYOFF	MCOL3	EI	UNSYNC

3. The space available for sprites initially has been reduced from 8191 to 7167 bytes.
4. The number of concurrent tasks allowed has been reduced from 5 to 3. This means that ALLOCATE now takes 2 parameters instead of 4

LASER BASIC FOR THE COMMODORE 64

5. The CUTOFF command now accepts frequencies scaled from C to 65535 instead of 0 to 2047.
6. SIDCLR does not alter the SID chip's master volume.
7. A Turbo-tape facility has been added.

LASER BASIC FOR THE COMMODORE 64

APPENDIX D: GLOSSARY OF TERMS

actual parameters	Variable or expression used when call a procedure or function using PROC or CFN
ADSR	Attack-decay-sustain-release envelope
AND	In terms of pixel data, a logical operation indicating that the destination pixel is set only if both source and destination pixels are already set.
attack	The rate at which a musical note reaches its peak volume.
attribute	Data associated with each character block indicating the colours to be used for displaying pixel data. In two-colour mode one byte of attribute data is associated with each character block. In four-colour hires, mode two bytes of attribute data are associated with each character block.
bit-map mode	A screen display mode, enabled by the HIRES command, in which each pixel can be individually set or cleared.
BLK	A prefix/suffix used for data movement commands indicating that pixel data at the destination of a move command is to be over written.
Compiler	A program which takes a program written in BASIC or some other high-level language and translates it into machine code.
decay	The rate at which a musical note falls from its peak volume to the sustain volume.
envelope	The shape of the volume of a musical note over time.
filter	An electronic circuit which removes certain frequencies from a signal.
Formal parameters	The variables used as parameters in a procedure or function definition.
four-colour mode	One of the screen display modes in which each pixel can take on one of four colours. The pixels in this mode are twice as wide as in two-colour mode.
Frequency	Cycles per second.
hi-res mode	See "bit map mode"
label	symbolic name associated with a program statement.
local variable	In a procedure or function , a variable which is created inside the procedure and destroyed upon exit.
lores mode	See "text mode".
MACHINE LIGHTNING	A games-writing utility in which the graphics commands available in LASER BASIC are controlled by an assembly language program.
Multi-colour mode	See "four-colour mode".
Multi -tasking	A mode in which the computer runs several programs at once.
OR	In terms of pixel data, a logical operation indicating that the destination pixel is set if either the source or destination pixels are
Pixel	An individual 'point' on the screen which can only be accessed individually in bit-map mode.
procedure	A piece of code which can be called with parameters.
release	The rate at which a musical note falls from sustain volume to nc volume
S2COL mode	A mode, enabled by the S2COL command, in which LASER BASIC's graphics commands operate on pixel data to be displayed in two colour mode.
S4COL mode	A mode, enabled by the S4COL command in which LASER BASIC's graphic s commands operate on pixel data to be displayed in four colour mode.
SID	Sound Interface Device.

LASER BASIC FOR THE COMMODORE 64

sprite variables	A set of 13 pseudo-variables which can be used to pass parameters to some of the graphics commands.
Structured programming	The coding of algorithms without the use of unconditional jumps.
sustain	The volume level for the sustain of a musical note.
syntax	Programming language sentence structure.
text mode	A mode in which only characters can be displayed on the screen and individual pixels cannot be accessed.
two-colour mode.	One of the hi-res screen display modes in which each pixel can take on one of two colours.
variable parameter	A parameter of a procedure which is altered by the procedure.
XOR	In terms of pixel data. a logical operation indicating that the destination pixel is set if one but not both of the source and destination pixels are already set.

LASER BASIC FOR THE COMMODORE 64

APPENDIX E : SCREEN DISPLAY CODES

UPPER	LOWER	CODE	UPPER	LOWER	CODE	UPPER	LOWER	CODE
@	@	0	,	,	44		V	86
A	a	1	,	,	45		W	87
B	b	2	-	-	46		X	88
C	c	3	/	/	47		Y	89
O	d	4	0	0	48		Z	90
E	e	5	1	1	49			91
F	f	6	2	2	50			92
G	g	7	3	3	51			93
H	h	8	4	4	52			94
I	i	9	5	5	53			95
J	j	10	6	6	54	SPACE	SPACE	96
K	k	11	7	7	55			97
L	l	12	8	8	56			98
M	m	13	9	9	57			99
N	n	14	:	:	58			100
O	a	15	;	;	59			101
P	P	16	<	<	60			102
Q	q	17	=	=	61			103
R	r	18	>	>	62			104
S	s	19	?	?	63			105
T	t	20			64			106
U	u	21		A	65			107
V	v	22		B	66			108
W	w	23		C	67			109
X	x	24		D	68			110
Y	Y	25		E	69			111
Z	z	26		F	70			112
		27		G	71			113
		28		H	72			114
		29		I	73			115
		30		J	74			116
		31		K	75			117
SPACE	SPACE	32		L	76			118
!	!	33		M	77			119
"	"	34		N	78			120
#	#	35		O	79			121
\$	\$	36		P	80			122
%	%	37		Q	81			123
&	&	38		R	82			124
'	'	39		S	83			125
()	40		T	84			126
)	(41						
*	*	42						
+	+	43						

LASER BASIC FOR THE COMMODORE 64

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

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

APPENDIX F : ASCII AND CHR\$ CODES

PRINTS	CHR\$	PRINTS	CHR\$	PRINTS	CHR\$
	0	1	49		98
	1	2	50		99
	2	3	51		100
	3	4	52		101
	4	5	53		102
WHITE	5	6	54		103
	6	7	55		104
	7	8	56		105
	8	9	57		106
	9	:	58		107
	10	:	59		108
	11	<	60		109
	12	=	61		110
RETURN	13	>	62		111
	14	?	63		112
	15	#	64		113
	16	A	65		114
DOWN	17	B	66		115
RVSON	18	C	67		116
HOME CSR	19	D	68		117
DELETE	20	E	69		118
	21	F	70		119
	22	G	71		120
	23	H	72		121
	24	I	73		122
	25	J	74		123
	26	K	75		124
	27	L	76		125
RED	28	M	77		126
RIGHT	29	N	78		127
GREEN	30	O	79		128
BLUE	31	P	80	ORANGE	129
SPACE	32	Q	81		130
!	33	R	82		131
"	34	S	83		132
#	35	T	84	f1	133
\$	36	U	85	f3	134
%	37	V	86	f5	135
&	38	W	87	f7	136
,	39	X	88	f2	137
(40	Y	89	f4	138

LASER BASIC FOR THE COMMODORE 64

)	41	Z	90	f6	139
*	42		91	f8	140
+	43		92	SHFT-RTN	141
,	44		93		142
"	45		94		143
.	46		95	BLACK	144
/	47		96		
0	48		97		

CODES 192-223 SAME AS 96-127

CODES 224-254 SAME AS 160-190

CODE 255 SAME AS 126

Note graphics characters are not shown in the table above.

APPENDIX G: MEMORY MAP

From	To	Description
\$0000	\$00FF	zero page
\$0100	\$01FF	6502 stack
\$0200	\$03FF	system variables
\$0400	\$05FF	miscellaneous storage for graphics commands
\$0600	\$57FF	* Laser BASIC interpreter
\$5800	\$9FFF	* BASIC program and variables
\$A000	\$BBFF	* sprites
\$BC00	\$BFFF	miscellaneous storage for graphics commands
\$C000	\$C7FF	character set and/or hardware sprites
\$C800	\$CBFF	text screen
\$CC00	\$CFFF	hires screen primary attributes
\$D000	\$D7FF	1/0 devices
\$D800	\$DBFF	text screen secondary attributes hires screen secondary attributes
\$DC00	\$DFFF	1/0 devices
\$E000	\$FFFF	hires pixel data
\$E000	\$FFFF	kernel ROM

* These can vary depending on whether multi-tasking or turbo tape are selected, and whether RESERVE has been used.

LASER BASIC FOR THE COMMODORE 64

APPENDIX H : SPRITE LIBRARIES

ARCADE SPRITE LIBRARY - "SPRITESA"

SPRITE NO.	DESCRIPTION	INK3	PAPER	WID	HGT
1	VINTAGE CAR	3	0	4	2
2	VAN	5	0	4	2
3	DRAGSTER	10	0	4	2
4	DUCK	7	0	3	3
5	DANCER	1	0	2	4
6	SPACESHIP #1	14	0	4	2
7	MOUSE	15	0	2	2
8	SPACESHIP #2	0	3	4	2
9	TANK #1	5	0	4	2
10	BI-PLANE	2	15	4	2
11	HELICOPTER #1	5	0	4	2
12	SPACESHIP #3	2	15	4	2
13	SPACE-TANK	6	15	4	2
14	SPACESHIP #4	0	15	4	2
15	JET-FIGHTER #1	0	15	4	2
16	TANK #2	5	0	6	3
17	JET-FIGHTER #2	0	3	5	2
18	SPACESHIP #5	3	0	5	2
19	SPACESHIP #6	1	0	6	2
20	JET-FIGHTER #3	5	0	5	2
21	ALIEN	6	3	4	2
22	SPACESHIP #7	0	1	5	2
23	SPACESHIP #8	0	1	6	2
24	OCEAN-LINER	6	3	7	2
25	TRI-PLANE	2	3	4	2
26	BULLDOZER	0	3	5	2
27	SPACESHIP #9	3	0	5	2
28	TANK #3	0	3	6	3
29	HELICOPTER #2	6	3	7	2
30	HOVERCRAFT	2	1	5	3
31	SUBMARINE	11	3	8	2
32	TANK #4	11	7	6	3
33	JET-FIGHTER #4	11	3	5	3
34	CROCODILE	11	3	6	3
35	FROG	5	0	3	4
36	RABBIT	1	4	2	4
37	GHOST	7	0	2	2
38	PAC-MEN	5	0	2	2
39	FLY	3	0	3	2
40	TRUCK	6	1	8	2
41	LUNAR-LANDER	1	0	3	4
42	SPACESHIP #1	0	0	15	5

LASER BASIC FOR THE COMMODORE 64

43	TEDDY-BEAR	7	0	4	5
44	BALL	7	0	3	3
45	TIN-SOLDIER	7	0	3	6
46	DIAMOND	3	6	3	3
47	DAGGER	7	6	5	2
48	JET-FIGHTER #5	0	15	5	2
49	SPACE BUGGY	0	15	5	3
50	CANNON	0	15	5	3
100 TO 108	USED FOR ANIMATION DEMONSTRATION	0	15	2	3

DEMO SPRITE LIBRARY - "SPRITESB"

SPRITE NO. DESCRIPTION INK 3 PAPER WID HGT

SPRITE NO.	DESCRIPTION	INK3	PAPER	WID	HGT
1	TORTOISE	3	0	4	2
2	RAT	15	0	4	2
3	HARE	1	0	4	2
4	FLOWER	0	2	4	
5	CAR	7	0	4	2
6	HEDGE	5	3	3	1
7	JUMPING ALIEN#1	1	0	4	4
8	JUMPING ALIEN#2	1	0	4	4
9	JUMPING ALIEN#3	1	0	4	4
10	JUMPING ALIEN#4	1	0	4	4
11	FALLING ALIEN	1	0	2	4
12	GIRDER SECTION	11	0	2	2
13	MONSTER	0	15	3	3
14	SPACESHIP	0	15	3	3
15	PLANET	0	15	3	3
16	ROCKET	15	0	;2	10
17	BASE	12	0	6	2
18	QUILL	0	15	3	3
19	CHESS PIECE	0	1	2	4
20	INVADERS#1		0	11	7
21	INVADERS#2		0	11	7
35	LANDSCAPE	6	3	60	2
80	OASIS LOGO	1	0	13	4
81	TOP OF LOCO	6	0	11	2
82	WHEELS OF LOCO#1	15	0	11	1
83	WHEELS OF LOCO#2	15	0	11	1
84	WHEELS OF LOCO#3	15	0	11	1
85	WHEELS OF LOCO#4	15	0	11	1
86	TOP OF COACH	15	0	10	2
87	WHEELS OF COACH#1	12	0	10	1
88	WHEELS OF COACH#2	12	0	10	1

LASER BASIC FOR THE COMMODORE 64

89	RAILS	15	0	8	1
100	DUMMY SPRITE			5	2
101	CHARACTER SPRITE #1			1	1
102	CHARACTER SPRITE #2			1	1

CHARACTER SPRITES AND MUSIC SPRITES - "SPRITESC "

NO.	TYPE	DESCRIPTION	COLOUR	WID	HGT
1	hires	character set		16	16
2	character	vintage car	3	4	2
3	character	van	5	4	2
4	character	dragster	10	4	2
5	character	duck	7	3	3
6	character	spaceship #1	14	4	2
7	character	mouse	15	2	2
8	character	spaceship #2	0	4	2
9	character	tank	5	4	2
10	character	bi-plane	2	4	2
11	character	helicopter	5	4	2
12	character	spaceship #3	2	4	2
13	hires	music demo voice 1		37	4
14	hires	music demo voice 2		151	1

APPENDIX I: MUSIC NOTE VALUES

This appendix contains a complete list of note frequency values to be used with the FRO command.

Octave	FRQ	Note	FRQ
C-0	268	C#-4	4547
C#-0	284	D-4	4817
D-0	301	D#-4	5103
D#-0	318	E-4	5407
E-0	337	F-4	5728
F-0	358	F#-4	6069
F#-0	379	G-4	6430
G-0	401	G#-4	6812
G#-0	425	A-4	7217
A-0	451	A#-4	7647
A#-0	477	B-4	7101
B-0	506	C-5	8583
C-1	536	C#-5	8094
C#-1	568	D-5	9634
D-1	602	D#-5	10207
D#-1	637	E-5	10814
E-1	675	F-5	11457
F-1	716	F#-5	12139
F#-1	758	G-5	12860

LASER BASIC FOR THE COMMODORE 64

G-1	803	G#-5	13625
G#-1	851	A-5	14435
A-1	902	A#-5	15294
A#-1	955	B-5	16203
B-1	1012	C-6	17167
C-2	1072	C#-6	18188
C#-2	1136	D-6	19269
D-2	1204	D#-6	20415
D#-2	1275	E-6	21629
E-2	1351	F-6	22915
F-2	1432	F#-6	24278
F#-2	1517	G-6	25721
G-2	1607	G#-6	27251
G#-2	1703	A-6	28871
A-2	1804	A#-6	30588
A#-2	1911	B-6	32407
B-2	2025	C-7	34334
C-3	2145	C#-7	36376
C#-3	2273	D-7	38539
D-3	2408	D#-7	40830
D#-3	2551	E-7	43258
E-3	2703	F-7	45830
F-3	2864	F#-7	48556
F#-3	3034	G-7	51443
G-3	3215	G#-7	54502
G#-3	3406	A-7	57743
A-3	3608	A#-7	61176
A#-3	3823	B-7	64814
B-3	4050		
C-4	4291		

Appendix J: C64 and Vice information

US PC keyboard

Action or character	PC US Keyboard
Return Key	Enter
Up Arrow Display Character	Shift 6 (Carat)
Left Arrow Display Character	Shift – (Minus)
Stop Run	Esc key
Delete Character	Backspace
Insert Blank	CTRL-Backspace/Shift-Backspace

Warning: in Vice pressing Shift P pauses execution – to resume go to settings, speed settings and uncheck Pause Emulation.

LASER BASIC FOR THE COMMODORE 64

Appendix K: Package Information

This package was assembled as part of an initiative to make more programming languages available for the C64 Mini.

For more info on the C64 please visit <https://retrogames.biz/thec64-mini>.

Recommend beginning with the Laser Basic Manual c64.pdf to get started.

File Name	Description
Laser Basic Manual c64.docx	Revised Laser Basic Users Manual in MS Word format
Laser Basic Manual c64.pdf	Revised Laser Basic Users Manual exported to PDF.
Laser Compiler Manual c64.pdf	Revised Laser Basic Compilers Manual.
Laser Basic v1.3.d64	Laser Basic Interpreter disk in c1541 format.
Laser Basic Examples.d64	Examples programs from the laser basic users manual.
Laser Basic Compiler.d64	Laser basic compiler.
Laser Basic Manual org.pdf	Original Laser Basic Users Manual.
Laser Compiler Manual org.pdf	Original Laser Basic Compilers Manual.

LASER BASIC FOR THE COMMODORE 64

NOTES

LASER BASIC FOR THE COMMODORE 64

TECHNICAL ENQUIRY CARD

WE AT OASIS BELIEVE IN GIVING FULL TECHNICAL SUPPORT TO ALL OUR PRODUCTS. TO ASSIST US, PLEASE FILL IN THIS CARD AND RETURN TO THE ADDRESS BELOW. TELEPHONE ENQUIRIES, OR LETTERS NOT ACCOMPANIED BY THIS CARD CANNOT BE ANSWERED. A REPLACEMENT CARD WILL BE ENCLOSED WITH OUR REPLY. ALL ENQUIRIES WILL BE ANSWERED AT OUR FIRST OPPORTUNITY.

MAKE AND MODEL OF COMPUTER.

SOFTWARE PURCHASED.

VERSION No.

PLACE AND DATE OF PURCHASE.

YOUR NAME.

ADDRESS.

TELEPHONE No.

AGE OCCUPATION.

PLEASE WRITE YOUR ENQUIRY HERE.

Please return to:

OASIS SOFTWARE,
12 WALLISCOTE ROAD,
WESTON-SUPER-MARE, AVON,
ENGLAND. BS23 1UG.