THE KEY TO ADVANCED GAMES DESIGN

MACHINE LIGHTNING

OASIS SOFTWARE

THE ADVANCED LEVEL GRAPHICS DEVELOPMENT SYSTEM FOR THE **COMMODORE 64.**

MACHINE LIGHTNING
by OASIS SOFTWARE

Copyright Notice

This manual

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

NOTE

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

CONTENTS

MACHINE LIGHTNING FOR THE COMMODORE 64

by David Hunter

The COMMODORE 64 is widely recognised as having the most powerful sound and graphics hardware available on any home computer, and as a result of this there is a rich selection of video games available for it, the most successful of which are written in machine code. The author of such a game has two major problems to overcome - he has to have some way of designing the graphics to be used in the game, and he has to have a set of debugged machine-code routines to place the graphics on the screen. Machine Lightning is designed to overcome these problems. It contains all the ingredients needed to produce commercial machine-code games, and consists of four compatible parts:

1. THE SPRITE GENERATOR

This is used to design and edit graphics to be used in the game. The sprites are saved to tape or disk in a form that can be used by the graphics routines.

2. BASIC LIGHTNING

Basic Lightning is an extension to the 64's BASIC interpreter which contains commands corresponding to the graphics routines in Machine Lightning. Thus, you can use Basic Lightning to test ideas easily before implementing them using Machine Lightning. Basic Lightning is also available separately.

3. 64-MAC/MON

This is a combined assembler/monitor which is used to write the game itself. Of course, you don't have to use it for writing video games; it can be used like any other assembler. 64-MAC/MON was used to write the graphics routines which are part of the Machine Lightning package, and it was also used to write Basic Lightning.

4. THE GRAPHICS ROUTINES

The graphics routines consist of 10K of machine code with routines to PUT and GET software sprites to and from the screen, scroll, enlarge or spin sprites and exchange data between two sprites or the screen. Collision detection is supported, as well as the 64's own hardware sprites.

Games written using the graphics routines can be marketed without restriction - all we ask is that you put a small credit on the packaging.

The sprite generator program and Basic Lightning are described in the Basic Lightning manual which is included along with this one. Before you can use the graphics routines, you will have to read the section on Basic Lightning's graphics commands.

Most of this manual is taken up by the instructions for 64 MAC/MON; Section 19 deals with the graphics routines.

64-MAC/MON provides a comprehensive set of over 70 commands for writing and debugging assembly language programs on the COMMODORE 64. It includes a line editor for the creation of source text, a full two-pass macro assembler, a symbolic disassembler, a machine code monitor and a tracer.

The editor automatically checks the syntax of lines as they are typed in, and formats the source text when it is listed. It includes block delete, move and copy as well as search and replace commands and automatic line numbering.

The assembler can be operated in either 'resident' or 'disk' mode. Resident mode is ideal for learning about assembly language or writing small programs - assembly is extremely fast, at over 20,000 lines per minute. Because text is tokenised when in memory, programs of over 2,500 lines can be written without having to use disk mode. In disk mode, linked files on floppy disk may be assembled. The size of program that can be written in this manner is only limited by the amount of mass storage available; about 8,500 lines of code in the case of the 1541 single floppy disk. The assembler also includes conditional assembly, cross-referencing and a printer pagination facility.

The machine-code monitor commands allow direct inspection and modification of memory - commands to list, move, relocate, compare, modify, search and disassemble blocks of memory are included. Up to 16 blocks which are printed as .BYTE, .WORD or .DBYTE directives when disassembling may be defined.

The tracer can single step through a machine-code program, displaying the register contents and the contents of up to 16 memory locations after executing each instruction. Options exist to suppress single stepping and register printing or to print only the program counter. Up to 16 locations can be defined at which the registers are always printed, even if register printing is disabled.

Two copies of 64 MAC/MON are supplied with the disk version - one is located in low memory and one in high memory. Only the low memory version is supplied on tape.

The low memory version occupies memory from $0800 to $47FF. The BASIC ROM from $A000 to $BFFF is switched out of memory after 64 MAC/MON has loaded, of after using RUN/STOP-RESTORE. However, 64 MAC/MON will still operate correctly if you re-enable the ROM by setting bit 0 of location 1.

The high memory version resides from $9000 to $CFFF. The BASIC ROM is enabled whenever memory is accessed by one of the monitor commands. This version is incompatible with the graphics routines.

Note that in both cases, full use of the zero page by the user's programs is allowed.

1. LOADING

1.1 LOADING FROM DISK

After switching on the computer system, insert the floppy disk into the drive and type the following:

```
LOAD "ML",8,1
```

After about ten seconds you are asked to press "L" or "H" to select between the low and high memory versions. Once you have done this it takes approximately one minute to load.

1.2 LOADING FROM TAPE

After switching on type SHIFT RUN-STOP and start the tape recorder.

1.3 When 64-MAC/MON has loaded, the following message is printed:

```
64-MAC/MON V1.2L

COPYRIGHT 1984 DAVID HUNTER

NEW (Y/N)? Y
TEXT MEMORY?
```

Unless you wish to reserve memory for your own machine code routines, hit RETURN - this reserves memory from $4800 to $CFFF for use as text storage, giving 34816 bytes free. Otherwise, type in the lower and upper limits of memory to be used, separated by a comma.

A 'BYTES FREE' message is then printed, follwed by the READY prompt.

Note that the computer's memory is completely cleared after loading, but it remains unaltered after subsequent NEW commands.

The following locations in pages 2 and 3 are altered by 64-MAC/MON:

$0200 to $0258	used for temporary string storage.
$028A	repeat key flag.
$0291	disables SHIFT keys.
$0300 to $030B	re-vectored to 64-MAC/MON warm start.
$0314 to $0315	IRQ re-vectored to allow use of function keys.

2. 6502 ASSEMBLY LANGUAGE

This section is not intended to teach assembly language programming - if you are a novice to the subject, we suggest that you read '6502 Assembly Language Programming' by Lance A. Leventhal, which is published by McGraw-Hill. Another worthwhile text is '6502 Assembly Language Subroutines' by Leventhal and Saville, published by Osborne/McGraw-Hill. However, the information presented here should suffice if you have knowledge of another microprocessor.

2.1 NOTATION

2.1.1 A <label> consists of a letter followed by up to fourteen of the following characters:

'A'..'Z' , '0'..'9' , ':' , '.' , '$'

Examples: COMPARE$NAMES OUTPUT3 T9

2.1.2 A <numerical constant> consists of one of the following:

"%" followed by a binary number
"@" followed by an octal number
a decimal number
"$" followed by a hexadecimal number
"'" followed by an ascii character (followed by an optional second quote)

Examples: %00001101 $ACD9 '7 19 '&'

2.1.3 An <expression> consists of <numerical constant>s and/or <label>s separated by the following operators:

"+" add
"-" subtract
"*" multiply
"/" divide
"?" exclusive-or
"&" logical and

There is no operator precedence, and brackets may not be used (this only applies to <expression>s that are included as part of an assembly language program). If '<' is placed before an expression, it is converted to a '&255' at the end of the expression when printing; similarly, '>' is converted to '/256'.

Examples: NUMBER$BASE+3 <INTERPRETER-1 INPUT$BUFFER/256 'Z'+1

2.1.4 A <string constant> is a number of ASCII characters enclosed in single quotes. If one of the characters is to be a quote then two successive quotes must be used.

Examples: 'BPLBMIBCCBCSBNEBEQBVCBVS' '''' '$@''%'

2.2 ASSEMBLY LANGUAGE STATEMENTS

There are three types of assembler statements: directives, instructions and comments.

2.2.1 Directive Statements

These may be considered as instructions which are obeyed at assembly time rather than run time. A directive statement consists of the following:

<label> <directive> <operand> <comment>

The label and comment fields are optional, and the operand field is not required in some cases. This assembler supports 27 directives, details of which are given below:

2.2.1.1 .BYTE directive

This is used to define single-byte constants. It should be followed by a number of <expression>s and/or <string constant>s separated by commas.

Examples:

```
        POWERSOF2     .BYTE     1,2,4,8,16,32,64,128
        HEXCHARS      .BYTE     '0123456789ABCDEF',0
```

2.2.1.2 .DBYTE directive

This has the same syntax as .BYTE but it generates two-byte constants in high-byte/low-byte order.

2.2.1.3 .WORD directive

This is the same as .DBYTE but the constants are in low-byte/high-byte order.

2.2.1.4 .PAD directive

The .PAD directive is used to pad out an area of program with NOP bytes.

```
Examples:     .PAD *&$FF00+256-*
              .PAD 6
```

2.2.1.5 .END directive

This is used to mark the end of an assembly language program. It is optional if the assembler is being used in resident mode.

2.2.1.6 .BLOCK directive

This directive is used to reserve space - it is followed by an expression which is added to the location counter.

```
Examples:     INPUT$BUFFER       .BLOCK 72
              XPOS               .BLOCK  2
```

2.2.1.7 = (equals) directive

The '=' directive is used to equate a label to an <expression>.

```
Examples:     INTERRUPTPERIOD=3906/SAMPLERATE
              CR             =13
```

It is important to realise that these calculations are carried out at assembly time, not run-time.

2.2.1.8 '*' is a reserved symbol which refers to the location counter during assembly. The program location counter may be set like this: *=$9000, and blocks of memory may also be reserved:

```
INPUT$BUFFER     *=*+72
```

2.2.1.9 .ORG directive

This is used to set the program location origin.

Example: .ORG $9000

Although this appears to be the same as *=$9000, there is a subtle difference between them which is explained in section 9.3.

2.2.1.10 .DEFMAC directive

This directive should be placed at the start of a macro definition. The label preceding the directive defines the macro name. It should be followed by a list of formal parameter labels separated by commas.

2.2.1.11 .ENDMAC directive

.ENDMAC is used at the end of a macro definition.

2.2.1.12 To call a macro in the program body, its name should be preceded by a colon and followed by a list of actual parameter expressions separated by commas.

Example...

```
1230 OUTPUT     .DEFMAC START,MODE
1240            LDA #START&255
1250            LDY #START/256
1260            LDX #MODE
1270            JSR PRINT
1280            .ENDMAC
.
.
3450            BNE LOOP
3460            :OUTPUT ALPHA+6,3
.
```

In resident mode, macros can be defined anywhere in the text- either before or after they are used, although it is best to keep them near the top of the program as this speeds up assembly. When assembling programs in disk mode, all macro definitions must be in the first file.

If a symbol is defined inside a macro and the macro is called more than once then a 'label defined twice' error message will probably be printed. To circumvent this problem, use the '*' symbol as in the following example:

```
.
100 DELAY       .DEFMAC DEL
110             LDX #DEL
120             DEX
130             BNE *-1
140             .ENDMAC
.
.
960             :DELAY 10
.
.
990             :DELAY 20
.
```

2.2.1.13 .IFEQ directive

If the expression following this directive is zero, assembly continues as normal, otherwise code generation is suppressed until the next .ELSE or .IFEND.

2.2.1.14 .IFNEQ directive

If the expression following this directive is non-zero, assembly continues as normal, otherwise code generation is suppressed until the next .ELSE or .IFEND.

2.2.1.15 .IFPOS directive

If the expression following this directive is in the range 1 to 32767, assembly continues as normal, otherwise code generation is suppressed until the next .ELSE or .IFEND.

2.2.1.16 .IFNEG directive

If the expression following this directive is in the range 32768 to 65535, assembly continues as normal, otherwise code generation is suppressed until the next .ELSE or .IFEND.

2.2.1.17 .IFEND directive

This is used at the end of a conditional assembly .IF construct- assembly after it proceeds as normal.

2.2.1.18 .ELSE directive

This works like the ELSE statement in extended BASICs- if code generation is suppressed, it is enabled, and vice-versa.

2.2.1.19 Examples of conditional assembly:

```
.
17450 OUTPUT    .IFEQ CBM64
17460           JSR $FFD2        ;CBM64 OUTPUT ROUTINE
17470           BCS ERROR1
17480           .ELSE
17490           STX TEMP
17500           TAX
17510           JSR $0238        ;ORIC ATMOS OUTPUT ROUTINE
17520           LDX TEMP
17530           PHA
17540           LDA KEYCHAR
17550           CMP #$83         ;CONTROL-C?
17560           BEQ ERROR2
17570           PLA
17580           .IFEND
.
```

```
.
2750               LDA #PRINTERON&255
2760               LDY #PRINTERON/256
2770               BIT PRINTERFLAG
2780               BMI PRINTMESSAGE
2790 PRHI         =PRINTEROFF/256
2800               .IFNEQ PRINTERON/256-PRHI
2810               LDY #PRHI
2820               .IFEND
2830               LDA #PRINTEROFF&255
2840 PRINTMESSAGE JSR OUTPUT$MESSAGE
.
```

2.2.1.20 .PRINT directive

This directive should be followed by a <string constant> which is simply printed when the directive is encountered during assembly.

Example:

```
19270 MESSAGES
19280 MSG1        .BYTE '?SYNTAX ERROR',0
19290 MSG2        .BYTE 'NUMBER TOO BIG',0
.
.
19440 MSG17       .BYTE 'FOUND ',0
19450 MSGEND
19460             .IFNEQ MSGEND-MESSAGES/256
19470             .PRINT 'MESSAGE TABLE IS LONGER THAN 256 BYTES'
19480             .END
19490             .IFEND
```

2.2.1.21 .LIST directive

This directive turns on the generation of an assembler listing, except if object code is being assembled to disk or tape.

2.2.1.22 .NOLIST directive

This turns off the generation of an assembler listing.

2.2.1.23 .PAGE directive

If an assembler listing is being output to the printer, this directive will start a new page.

2.2.1.24 .PAGEIF directive

This should be followed by an <expression>- if this is greater than the number of lines left on the page, a new page is taken, otherwise one line is skipped. This only takes place if an assembler listing is being output on the printer.

Example:

```
.PAGEIF 24
```

2.2.1.25 .SKIP directive

This is used to print a certain number of blank lines when assembling a listing to the printer.

Example: .SKIP 2

If the number is left out, a default value of 1 is assumed.

2.2.1.26 .TITLE directive

This should be followed by a <string constant> which will be printed at the top of each new page on the printer.

Example: .TITLE 'C64 MACRO ASSEMBLER'

2.2.1.27 .WIDTH directive

This sets the number of characters printed per line on the printer.

Example: .WIDTH 96

2.2.1.28 .HEIGHT directive

This sets the number of lines printed per page on the printer.

Example: .HEIGHT 66

2.2.1.29 .INTNUM directive

This initialises the printer page number to zero.

2.2.1.30 .FILE directive

This directive is used to link files together in disk mode. At the end of each file, there should be a .FILE directive followed by a <string constant> consisting of the name of the next disk file.

Example: .FILE 'ASM4'

2.2.1.31 All directives, apart from .PAGE, .END and .ENDMAC may be abbreviated to their first three letters.

Example: .BYT $C9,$A9,$89

2.2.2 Instruction statements

An instruction statement consists of:

<label> <opcode mnemonic> <operand> <comment>

The <label> and <comment> fields are optional. Details of the allowable <opcode mnemonic>s and <operand>s are given in sections 16 and 17 of this manual respectively.

If during the first pass of assembly an instruction which has both zero page and absolute addressing modes has as its operand an undefined expression, as in this example:

```
10              *=12345
20              LDA FIVE
30 ABCDEF       JMP ABCDEF
40 FIVE         =5
```

and the expression is evaluated during the second pass as being less than 256, the assembler will insert an extra NOP byte before the next label definition during the second pass.

2.2.3 Comment statements

A comment statement consists of the following:

; <comment>

The <comment> may be any commentary whatsoever.

3. ARITHMETIC EXPRESSIONS IN COMMAND MODE

3.1 In command mode, line numbers, memory locations and so on are expressed as <expression>s, as defined in 2.1.3, with the difference that brackets may be used and operator precedence exists. A '#' is used to represent the logical-or operator. A full stop may be used to represent the last result from the CALC command, and !<label> gives the line number in which a label is defined.

3.2 A <string> is defined as a series of characters bounded by one of the following delimeters:

! " # $ % & ' ( ) * + , - . /

If the <string> is to be followed by an end-of-line, the delimeters may be omitted.

4. USING THE EDITOR

4.1 The screen editor may be used as in BASIC. The RUN/STOP key terminates a listing at any time. CTRL slows down printing and the SPACE bar can be used to temporarily halt a listing- pressing it again restarts the listing.

4.2 FUNCTION KEYS

The function keys may be defined as follows, where n is the number of the function key:

Fn=<string>

The back-arrow key at the top left of the keyboard can be used to represent RETURN.

Examples: F7=% .BYTE %
               F4=ASM,L

4.3 If you type in a line number followed by a line of 6502 assembly language, the editor will put the line into memory according to its line number (these may be 1 to 65535). A line number followed by RETURN deletes that line, and a line with number zero will be put immediately after the last line entered or deleted.

If the editor finds a syntax error in a line of source code, it prints an arrow pointing to the error and an error message. This feature can be suppressed using the EDITOR command.

In any situation which could result in the destruction of the source text, the editor will prompt with 'ARE YOU SURE (Y/N)? ' before proceeding.

In the following list of commands, each one is followed by its abbreviated version in brackets.

4.4 EDITOR (ED.) command

This command puts the assembler into 'EDITOR' mode which disables the automatic syntax checking of lines; in this mode the text is not tokenised and therefore cannot be assembled. Entering or leaving this mode destroys any text that is in memory.

4.5 RESIDENT (RES.) command

This command puts the assembler into 'RESIDENT' mode in which programs may be assembled directly from memory. Further details are given in section 9 of this manual.

4.6 DISK (DISC or DI.) command

This command puts the assembler into 'DISK' mode in which programs may be assembled from disk. Further details are given in section 10 of this manual.

4.7 LIST (L.) command

This is used to list lines of text. It may be followed by one or more line specifications (separated by semicolons) of the following types:

```
<line number>
<first line>,<last line>
,<last line>
<first line>,
```

Missing out the line specifications will list the whole source text.

Examples:
```
LIST
LIST 23790
L. !PRINTMNEMONIC,
LIST ,100;110;150,160
L. 23990
LIST 100,100+70
```

4.8 PRINT (P.) command

This is the same as LIST but no line numbers are printed.

4.9 DELETE (D.) command

This command is used to delete lines from the source text- the syntax is the same as for LIST. The editor will list the lines and then prompt with 'ARE YOU SURE (Y/N)? ' before the lines are actually deleted - hit 'Y' to carry out the deletion.

Examples:

```
D. 23770
DEL 1440;2680;3725,3737
```

4.10 RENUMBER (R.) command

This renumbers lines in the text. It may take any of the following forms:

RENUMBER
first line no.=10, step size=10

RENUMBER X
first line no.=X, step size=10

RENUMBER X,Y
first line no.=X, step size=Y

If renumbering would cause a line number greater than 65535 to be generated, the text is renumbered from line 1 in steps of 1. After renumbering, the last line number+step size is printed.

Examples:

```
RENUMBER
R.10000
REN. 100,25
```

4.11 MEM (M.) command

This command returns a message giving the free memory total and the current editor mode. If the source file is very long there will be a delay of a few seconds while symbol table garbage collection is carried out.

4.12 NEW (N.) command

This erases the source program in memory and then prompts for the memory to be reserved for source text.

4.13 AUTO (AU.) command

This puts the computer into AUTO mode- after a line number and text is entered, the next line number is automatically printed. The default value of the step size is 10. This can be changed by placing the new value after the command. If the step size is zero, line number 0s only are printed. To stop the printing of the numbers, enter a blank line.

Examples:

```
AU.
AUTO 5
```

4.14 MANUAL (MA.) command

This brings the computer out of AUTO mode.

4.15 MOVE (MO.) command

This command is used to move a block of lines from one part of the text to another. It takes the following form:

MOVE <new line number>=<first line>,< last line>

After the lines have been moved, the first line is renumbered to the <new line number>, the rest being renumbered to line 0. If the <new line number> already exists, an error message is printed.

Examples: MOVE 1475=2510,2850
MO. 23000=570,810

4.16 COPY (CO.) command

This is similar to MOVE, the difference being that the lines are not deleted from their original position once they have been moved.

4.17 FIND (FI.) command

This command is used to find the location of a sequence of characters in the text. It takes the following forms:

FIND <string>
FIND <string> <line specification>

The second form should be used if it is desired to search only a part of the text. When the <string> is found, the line in which it appears is printed.

Examples: FIND "JSR"
FIND & HEXOUT& 1140,2930

4.18 CHANGE (CH.) command

This command is used to change all occurrences of a particular series of characters. It should be followed by two strings and a line specification. The delimiter at the end of the first string should not be duplicated at the start of the second string.

Examples: CH.!HEXOUT!HEX$BYTE$OUT!
CHANGE %JSR OUTCH%JSR OUTCHR% 3980,7620

5. EDITOR ERROR MESSAGES

The following error messages can be generated by the editor:

OUT OF FUNCTION KEY SPACE
All the function key definitions may not total more than 119 chracters.

NUMBER TOO BIG

EXPRESSION TOO COMPLEX
An expression has too many levels of nested parenthesis.

DIVISION BY ZERO

LABEL TOO LONG
A label was found which is longer than 15 characters; the editor uses only the first 15.

LABEL DOES NOT BEGIN WITH A LETTER

"A" IS A RESERVED LABEL

6502 OP-CODES ARE RESERVED LABELS
A 6502 op-code mnemonic was used as part of an expression.

BAD INDEX
Index must be X or Y. This message is also generated if either (<expression>,Y) or (<expression>),X are encountered.

BAD DIRECTIVE
A string was found after a full stop which is not one of the legal directives.

* FULL *
You have run out of memory space.

FILE ERROR

STRING TOO LONG
The maximum length allowable is 64 characters.

OUT OF RANGE
An attempt was made to move a block of text to a location within itself.

SYNTAX ERROR
The error does not fall into one of the above categories.

6. LOADING AND SAVING

6.1 LOAD (LO.) command

This is used to load source files into memory. It should be followed by a <string> for the filname.

To merge a file onto the program in memory, follow the filename with a comma (optional) and the line number where the text is to be inserted. The text is always inserted before the line specified if it exists. Using this facility, subroutines that have been previously written, debugged and saved can be incorporated into a program.

If a line is loaded which is not valid, it is listed on the screen to be corrected after loading.

```
Examples:    LOAD XBASIC
             LOAD %HEXPRINT% 200
             LO. !A1!
             LOAD
             LOAD (SOURCE(,300
```

6.2 SAVE (SA.) command

This saves the source text to disk or tape. The filename may be followed by a line specification (as defined in Section 4.7) if only part of the source file is to be saved.

```
Examples:    SAVE $PART6$ 100,400
             SAVE
             SAVE @0:ASM1
```

Source text is saved in a compressed format: all unnecessary spaces are removed, and any that remain are removed and bit 7 of the next character is set.

6.3 FSAVE (F.) command

This command is similar to SAVE, but the text is saved formatted as it would be printed, without any text compression.

6.4 MLOAD (ML.) command

This is used to load machine-code files that have been saved in the standard format. To load it at a different address than it was saved at, follow the filename with the new start address; this may be preceded by a comma.

```
Examples:    MLOAD LOADER
             MLOAD
             MLOAD 'SPRITES',$8000
```

6.5 MSAVE (MS.) command

This saves machine-code in the standard format. The command should be followed by the filename, start address and end address, all separated by commas.

```
Examples:    MSAVE %TITLE SCREEN%,$A000,$C000
             MSAVE $OUTPUT PATCH$,$C000,$C180
```

6.6 OLOAD (OL.) command

This loads object code that has been saved in ASCII format; this is the format used for object code files generated by the assembler.

The command should be followed by the filename.

```
Example:     OLOAD 'OBJECT'
```

Each byte of data to be stored is converted into two half bytes which are translated into their ASCII equivalents ('0' to 'F'). Each output record begins with a ';' character. The next byte is the number of data bytes contained in the record. The record's starting address High (1 byte, 2 characters), starting address Low (1 byte, 2 characters) and data (maximum 24 bytes, 48 characters) follow. Each record is terminated by the record's checksum (2 bytes, 4 characters) and a carriage return.

The last record saved has zero data bytes (indicated by ;00). The starting address field is replaced by a four digit hexadecimal number representing the total number of data records contained in the file, followed by the record's usual checksum digits.

Examples:

```
;180000FFEEDDCCBBAA0099887766554433221122334455667788990AFC
;0000010001
```

Note: A program is supplied with Machine Lightning under the filename "LOADER" which loads data in ASCII format and can be run from BASIC.

6.7 OSAVE (OS.) Command

This saves object code in ASCII format, details of which are given above. The command is followed by a filename, and the start and end addresses separated by commas. More than one block of data may be saved by separating several start and end address pairs with semicolons.

Examples:

```
SAVE &TESTFILE&,$4C00,$5000
SAVE $PART6$,$6000,$7000;$81F3,$8230
```

6.8 OC+ and OC- commands

After executing the OC+ command, all object code is saved in a compact format in which bytes are output directly to disk rather than being printed in hex and the record start character is ':' rather than ';'.

The OC- command is used to revert to the normal format.

7. USING A PRINTER

7.1 CENTRO (C.) command

The assembler is set to use a serial bus printer (device number 4) upon initialisation. To use it with a centronics interface printer connected to the user port, type 'CENTRO+'. To revert to the serial bus printer, use 'CENTRO-'.

7.2 CTRL (CT.) command

This can be used to send a series of control codes to the printer for initialisation purposes. It should be followed by one or more <expression>s separated by commas.

Example:

```
CTRL 27,'M
```

7.3 * command

Placing an asterisk before any command will direct its output to the printer.

Examples:

```
*L.4920,5260
*ASM,L,C
```

7.4 Printer Pagination

At the top of each new page, a heading consisting of a title and a page number is printed. The following commands are available (some are also assembler directives):

7.4.1 INTNUM command

This sets the current page number to zero.

7.4.2 SETPAGE command

This is used to define the paper size. The command can exist in either of the following forms:

```
SETPAGE X    Sets the paper width to X.
SETPAGE X,Y  Sets the paper width to X and page height to Y.
```

The minimum value for either of these parameters is 16; the maximum is 127. To disable paging, set the page length to zero.

7.4.3 SKIP command.

This is used to skip a certain number of lines.

Example:

```
SKIP 15
```

7.4.5 TITLE command

This sets the title that is printed at the top of each new page.

Example:

```
TITLE SOURCE CODE LISTING
```

7.5 Setting up the Printer.

After loading the assembler, position the print head at the top of a new page. This ensures that subsequent page headings will be properly aligned.

The paper width and height are set to 80 and 66 respectively after 64-MAC/MON has been loaded.

8. DOS SUPPORT

8.1 @ (or >) command

This sends a command to the disk drive. The legal commands are:

8.1.1 Format a Disk: @N<drive number>:<disk name>,XX

XX is a unique 2-character identifier; omitting it results in all files being deleted rather than re-formatting the whole disk.

Example: @N0:DISK 1,99

8.1.2 Delete a File: @S<drive number>:<filename>

Pattern matching using '*' and '?' may be used to delete groups of files.

Examples:
>S1:ASM*
@S0:OBJECT

8.1.3 Rename a File: @R<drive number>:<new file name>=<old file name>

Example: @R0:PROGRAM=P6

8.1.4 Validate a Disk: @V<drive number>

This reconstructs the Block Availability Map on the disk. If you suspect that a disk is corrupted, this command will prevent further corruption of files. It should also be used if you have any files on the disk that are not properly closed.

8.1.5 Duplicate Disk: @D<destination drive number>=<source drive number>

Example: @D1=0

8.1.6 Copy File: @C<drive number>:<new file>=<drive number>:<old file>

This command can also be used to concatenate several files:

@C<drive number>:<new file>=<drive number>:<file 1>,<drive number>:<file 2>

A maximum of four files can be joined in this manner.

Examples:
>C0:PROG2=0:PROG1
@C1:SOURCE=0:A1,0:A2

8.1.7 Print Directory: @$<drive number>:<filename>

The 'DIR' command can be used instead of '@$'

Examples:

```
>$
@$0:ASM*
@$1:MONITOR$7000
DIR $0:A*
DIR
```

8.1.8 Read Error Channel: @ or > or ERR

This will print out an error number, error name, and track and sector numbers.

Further details of these commands can be found in your disk drive manual. For a single disk drive, the <drive number> should always be 0.

8.2 Pattern Matching

Pattern matching can be used with LOAD and DOS commands. The two symbols used are '*' (signifies 'with anything following') and '?'. '?' matches with any character.

For example, 'COM????R*' can be used to specify 'COMMODORE' and 'COMPUTERS' but not 'COMBINATION'.

8.3 DEVICE (DEV. or #) command

This is used to change the divice number used in LOAD, SAVE and DOS commands.

Example: 'DEVICE 1' will allow loading and saving of files from tape.

8.4 Using more than one Disk Drive Unit.

Drive numbers 2 to 7 may be used to specify device numbers 9 to 11 as shown in the table below:

USER FILENAME	DEVICE	DISK FILENAME
0:filename	8	0:filename
@0:filename	8	@0:filename
1:filename	8	1:filename
@1:filename	8	@1:filename
2:filename	9	0:filename
@2:filename	9	@0:filename
3:filename	9	1:filename
@3:filename	9	@1:filename
4:filename	10	0:filename
@4:filename	10	@0:filename
5:filename	10	1:filename
@5:filename	10	@1:filename

6:filename	11	0:filename
@6:filename	11	@0:filename
7:filename	11	1:filename
@7:filename	11	@1:filename

N.B. Due to the lack of bus arbitration on the serial bus, the system will hang if you try to write to one disk drive while reading from another.

8.5 The '↑' Filename .

If you save a file under the filename '↑', the assembler will use the filename in a comment on the first line of the source program currently in memory. For example, if the first line is:

```
10      ;@0:PROG
```

typing 'SAVE ↑' is equivalent to typing 'SAVE @0:PROG'.

9. THE ASSEMBLER IN RESIDENT MODE

In RESIDENT mode, the source file is held in memory.

9.1 ASM (A.) command

This command assembles the source file. It may be followed by letters, preceded by commas, which are used to select various options:

- L A full assembler listing is generated.
- M Assembles directly to memory.
- O Assembles object code to tape or disk.
- C A concordance listing is generated.

Examples:

```
ASM,O
A.,L,C
A.
```

9.1.1 Each line of the assembler listing consists of the following:

line number, address (in hex), object code, source line.

9.1.2 If object code is assembled to tape or disk, you are prompted for the object filename before assembly starts. The object code file is closed if any errors occur. Note that the object code generated by the assembler cannot be loaded directly into memory - it must be loaded using either the OLOAD command in the 64 MAC/MON or the special object code loader program supplied.

9.1.3 Two concordance listings are actually printed: the first has the labels in alphabetical order, and the second has them in numerical order. Each line of the concordance listing contains the label, its value, the line that it was defined in and the lines in which it was referred to. This is printed after the assembler listing.

9.1.4 If an error is found, a pointer will be printed pointing to the error, followed by an error message. A full list of error messages is given in Part 11 of this manual.

9.2 OFFSET (O.) command

This sets an offset which is added to the location counter when object code is being output using the O or M options. This offset also applies to the OLOAD, OSAVE and monitor-type commands.

Examples: OFFSET $9800
O.$E000

9.3 RUN command

This command assembles and executes an assembly language program, allocating memory to place the object code in.

The start address of the program itself should be defined with a .ORG directive (see Section 2.2.1.9), but when the location counter is set to reserve space, *= should be used.

10. THE ASSEMBLER IN DISK MODE

10.1 In DISK mode, the source program is held on disk as a series of linked files. The files are linked with .FILE directives (see Section 2.2.1.30). The last file should finish with an .END directive (Section 2.2.1.5).

10.2 All macros must be defined in the first file; this is kept in memory throughout the assembly. This file should be short, so as to leave as much space as possible for the symbol table. If there is a symbol table overflow, a '* FULL *' error message is printed, and assembly is aborted.

10.3 ASM (A.) command

In disk mode, the ASM command should not be followed by option letters as in resident mode. Instead, the computer asks if a listing, concordance listing or object code is to be generated. When prompted for the source filname, give the name of the first file.

If there is a source file in memory, you are given the option of saving it since it will be destroyed by assembly in disk mode; if you do not wish to save it, hit return when you are prompted for the filename.

11. ASSEMBLER ERROR MESSAGES

The assembler can generate the following error messages:

FWD REF OR UNDEFINED LABEL IN .BLOCK OR ORIGIN DIRECTIVE

.BYTE DIRECTIVE DATA TOO BIG
The .BYTE directive can only accept data less than 256.

BRANCH OUT OF RANGE
A relative branch must be to an address within the range *-126 to *-129.

BAD OP-CODE/OPERAND COMBINATION

LABEL ALREADY DEFINED

IMMEDIATE OPERAND TOO BIG
Immediate operands must be less than 256.

IRRESOLVABLE FWD REF OR UNDEFINED LABEL
The label has not been defined in the source file.

MACROS NESTED TOO DEEP
Macros can be nested up to a maximum of 32 deep.

TOO MANY .ENDMACS
An .ENDMAC was found without a corresponding .DEFMAC.

TOO MANY .DEFMACS
A .DEFMAC was found inside a macro definition. This message is also generated if the assembler runs off the end of the source text in the middle of a macro.

WRONG NUMBER OF PARAMETERS
The wrong number of parameters were used in a macro call.

UNDEFINED MACRO

CONCORDANCE TABLE OVERFLOW
This message is printed at the end of assembly.

CONTEXT ERROR IN DISK FILE
Disk mode only; a line was found in the source file which is not a valid line of 6502 assembly language. This probably means that the disk is corrupted.

ASSEMBLY TO RESERVED MEMORY
An attempt was made to assemble on top of zero page, the source text, the symbol table or the assembler. This also happens if there is not enough room for the object code in the RUN command. This message is only given if an assembly to memory is being carried out.

12. MACHINE CODE MONITOR COMMANDS

These commands allow you to inspect and modify memory directly; they include a symbolic disassembler. The offset as set by the OFFSET command is added to all locations when they are referred to.

A <byte string> is defined as a series of <string constant>s (see Section 2.1.4) and/or expressions (see Section 3.1) separated by commas.

Example: 1,2,'ABC',9

12.1 DECIMAL (DE.) command

This puts the monitor commands into DECIMAL mode: all numerical output is in base 10.

12.2 HEX (H.) command

This puts the monitor commands into HEX mode: All numerical output is in base 16.

12.3 CALC (?) command

This evaluates an expression and prints the result as an ASCII character, and in binary, octal, decimal and hex.

Examples:

```
CALC 'A'+1
'B' %01000010 @102 066 $42
READY.

? (9+3)/4
'.' %00000011 @003 003 $03
READY.
```

12.4 MLIST command

This prints the contents of memory in both numerical and ASCII form. It can take two forms:

MLIST <start address>
Lists memory 8 lines at a time. Hit <return> to continue, otherwise type in the next command.

MLIST <start address>,<end address>
Lists a block of memory continuously.

For a different output format, replace MLIST in the above with:

MLIST@ <no. bytes per line>,<no. spaces between bytes>,<no. linefeeds between lines>,

To change a byte in memory, simply move the cursor over it, change it and hit <return>. If you don't hit <return>, the byte in memory will not be changed in memory even if it is changed on the screen.

12.5 MDUMP (DUMP or MD.) command

This is similar to MLIST, the difference being that memory is not printed as ASCII characters.

12.6 MFIND (MF.) command

This prints the addresses of all occurrences of a sequence of bytes between two addresses:

MFIND <byte string> > <start address>,<end address>

Examples: MFIND $68,$20,$15,$DF>$E061,$F83C
MFIND 'BASIC'>$A000,$C000

12.7 COMPARE (CO.) command

This compares one block of memory with another, printing any differences:

COMPARE <start address 1>=<start address 2>,<end address 2>

Example: If the memory contents are:

```
$8000: $01 $02 $03 $04 $05 $06 $07 $08
$9000: $01 $03 $02 $04 $05 $66 $07 $08
```

then the following would be printed:

```
COMPARE $8000=$9000,$9008

$8001=$02,$9001=$03
$8002=$03,$9002=$02
$8005=$06,$9005=$66
READY.
```

12.8 MFILL (FILL) command

This fills a block of memory with a sequence of one or more bytes:

MFILL <start address>,<end address>=<byte string>

Examples: MFILL $D800,$DC00=0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
MFILL $8000,$9000=$EA

12.9 MMOVE (MM.) command

This moves a block of memory from one place to another:

MMOVE <new start address>=<old start address>,<old end address>

Example: MMOVE $B000=$9012,$9200

12.10 RELOC (REL.) command

This is similar to MMOVE, the difference being that all JMPs etc. are changed so that the program will run at the new address.

Example: RELOC $5000=$A000,$C000

It is also possible to place the relocated program in a different part of memory from where it will run:

RELOC <memory address>,<run address>=<old block start>,<old block end>

12.11 MCHANGE (MCH.) command

This finds all occurrences of a sequence of bytes and replaces it with another:

MCHANGE <old byte string> > <new byte string> > <start address>,<end address>

Example: MCHANGE$20,$91,$19>$20,$95,$19>$C900,$CA00

12.12 USR (U.) command

This performs a user-defined operation on a block of memory. A subroutine must be supplied which carries out the required operation on the accumulator:

USR <subroutine address>=<start of block>,<end of block>

Example: Add 1 to all locations between $8000 and $9000

1. Assemble into memory:

```
10 *=$9800
20        CLC
30        ADC #1
40        RTS
```

2. USR $9800=$8000,$9000

   The X and Y registers may be used in the subroutine

12.13 DUSR (DU.) command

This carries out a user-defined operation on double-byte quantities in a block of memory. A subroutine must be supplied which carries out the required operation on the accumulator (LOW) and X (HIGH).

Example: Subtract 1 from each entry in a table of addresses between $CB00 and $CB4C.

1. Assemble into memory:

```
10 *=$C000
20        SEC
30        SBC #1
40        BCS RETURN
50        DEX
60 RETURN RTS
```

2. DUSR $C000=$CB00,$CB4C

   The Y register may be used in the subroutine.

12.14 The Symbolic Disassembler

12.14.1 BYTE (B.) command

This is used to define blocks of memory which are printed as .BYTE directives when disassembling; the command should be followed by the lower and upper limits of the block, separated by commas. Up to 16 blocks can be held in memory at once, and more than one can be defined at a time by separating the blocks by semicolons.

Example: BYTE $9164,$9197;$9304,$9308

12.14.2 ASCII (ASC.) command

This is similar to .BYTE, but any ASCII characters are output as <string constant>s.

12.14.3 WORD (W.) command

This is similar to .BYTE, but it defines blocks which are printed as .WORD directives.

12.14.4 DBYTE (DB.) command

This is similar to .BYTE, but it defines blocks which are printed as .DBYTE directives.

12.14.5 TABLES (TA.) command

This prints out a table of the blocks of memory defined in the above four commands.

12.14.6 TABDEL command

This has the same syntax as the byte command, but it deletes an entry from the table.

12.14.7 TABCLR command

This removes all entries from the table defined by the BYTE, ASCII, WORD and DBYTE commands.

12.14.8 Defining Symbols for use by the Disassembler

If the symbol table has been destroyed by printing out the 'BYTES FREE' message, the disassembler output is non-symbolic, and the address and object code are also printed. After an error-free assembly, the disassembler will search the symbol table when printing out any constants. The address and object code is not printed out in this mode. Thus, an assembler source file consisting of '=' directives should be used to define symbols for use by the disassembler; the number of symbols is only limited by the memory size, about 1500 if the full 34k is being used for text storage.

12.14.9 DASM (DA.) command

This command is used to disassemble machine code in memory. It can take two forms:-

DASM <address>
This disassembles in blocks of 23 lines. Hit <return> to continue, or type in the next command.

DASM <start address>,<end address>
This disassembles a block of memory continuously.

12.14.10 FDASM command

This is used to disassemble to disk or tape:

FDASM <filename> <start address>,<end address>

Example: FDASM #BASIC# $A000,$C000

12.15 SYS command

This command calls a machine code subroutine.

Example: SYS $9804

12.16 Although all of the available zero page is used by the 64-MAC/MON, this memory can be used in your own routines as it is exchanged with a set of temporary storage locations when fetching or storing bytes in memory. Also, in the high memory version, the BASIC ROM is switched on when accessing memory, so that it is possible to disassemble it or use subroutines from it in your own programs.

13. MONITOR ERROR MESSAGES

The monitor commands can generate the following error messages:

WRONG LENGTH
The two byte strings in a MCHANGE command are of different length.

NOT IN TABLE
An attempt was made to delete a non existant table entry.

TABLE FULL
The tables can each hold only 16 entries.

OUT OF RANGE
An attempt was made to RELOCate a program to a location within itself.

14. THE DEBUGGER/TRACER

14.1 OPT command

This command is used to select the tracer's mode of operation. The command should be followed by zero or more of the following letters, each preceded by a comma:

J : JSR mode.
S : STEP mode.
A : ADDRESS mode.
R : REGISTER mode.

14.1.1 If JSR mode is enabled, all JSR instructions are executed rather than traced, and tracing stops when a RTS instruction is found. This mode should be used for debugging subroutines, where the lower level subroutines have already been similarly debugged.

14.1.2 In STEP mode, you must press return each time the registers are displayed. To terminate the trace, type in the next command as usual.

14.1.3 In ADDRESS mode, the address of each instruction executed is printed out.

14.1.4 In REGISTER mode, the register contents are printed at each instruction. If this mode is disabled, they are only displayed upon 'display-points' (see section 14.2).

14.2 DISP command

This is used to set the 'display points' at which the register contents are always printed out. Several display points may be specified by separating them with commas. A maximum of 16 display points may be specified. 'Display points' are not the same as conventional breakpoints ; they are detected by the software in the tracer rather than using a hardware BRK instruction, and they therefore are not detected if a machine-code program is run using the SYS command, but, unlike breakpoints, they may be set in ROM routines.

Example: DISP $8000,$8120

14.3 DISTAB command

This prints out a table of all the current 'display points'.

14.4 DISDEL command

This has the same syntax as DISP but it deletes items from the table.

14.5 DISCLR command

This clears all entries fromm the display point table.

14.6 REGS command

This prints out the contents of the CPU registers as used by TRACE. The register contents can be modified in the same way as memory locations with MLIST (see section 12.4). Only the contents of A, X, Y, S and P are relevant to the SYS instruction - SYS ignores the value of PC as given by REGS and the value of PC after SYS is invalid.

14.7 LOC command

This has the same syntax as the DISP command and is used to specify memory locations whose contents are always printed out along with the registers.

14.8 LOCDEL command

This has the same syntax as LOC but it deletes items from the table.

14.9 LOCCLR command

This deletes all entries from the table of locations specified by the LOC command.

(A LOCTAB command is not included since the REGS command will print out a list of the locations in the table).

14.10 TRACE command

This starts tracing at the memory location given by PC's value as it is printed out by the REGS command. If a BRK or an illegal instruction is found while tracing, a message is printed containing PC's current value and tracing is halted.

15. 6502 ARCHITECTURE

15.1 Byte-length registers:

A (accumulator)
P (processor status flag register)
S (stack pointer)
X (index register X)
Y (index register Y)

The general purpose user registers are A, X and Y. The stack pointer always contains the least significant byte of the next available stack location in page 1 ($0100 to $01FF). The P register consists of a set of seven status flags.

15.2 Word-length registers:

PC (computer counter)

Note: Pairs of memory locations in page zero may be used as word-length registers to hold indirect addresses. The lower address holds the least significent (or low) byte and the higher holds the most significant (or high) byte. Since the 6502 provides automatic wraparound, addresses $00FF and $0000 provide a rarely used pair.

15.3 Flags:

The flags are arranged in the P register as follows:

bit	flag	purpose
0	C	Carry
1	Z	Zero
2	I	IRQ interrupt disable
3	D	Decimal mode
4	B	BRK command
5	X	Unused (always set to 1)
6	V	Overflow
7	N	Negative (sign)

16. THE 6502 INSTRUCTION SET

ADC - Add memory to accumulator with carry.
flags affected: N,Z,C,V (Z is invalid if in decimal mode).

AND - Logical 'and' accumulator with memory.
flags affected: N,Z

ASL - Arithmetic shift left. (Bit 7 goes to C flag, a 0 is shifted into bit zero).
flags affected: N,Z,C

BCC - Branch to destination if C flag=0

BCS - Branch to destination if C flag=1

BEQ - Branch to destination if Z flag=1

BIT - Bit test. Logical 'and's ACC with memory and sets Z on the result but does not alter the contents of ACC. Bit 7 of memory goes to the N flag, and bit 6 goes to the V flag.
flags affected: N,Z,V

BMI - Branch to destination if N flag=1

BNE - Branch to destination if Z flag=0

BPL - Branch to destination if N flag=0

BRK - Force IRQ interrupt.

BVC - Branch to destination if V flag=0

BVS - Branch to destination if V flag=1

CLC - Clear the carry flag.
flags affected: C(=0)

CLD - Clear the decimal mode flag.
flags affected: D(=0)

CLI - Clear the interrupt disable flag.
flags affected: I(=0)

CLV - Clear the overflow flag.
flags affected: V(=0)

CMP - Compare accumulator with memory.
flags affected: N,Z,C

CPX - Compare X index register with memory.
flags affected: N,Z,C

CPY - Compare Y index register with memory.
flags affected: N,Z,C

DEC - Decrement memory.
flags affected: N,Z

DEX - Decrement X index register.
flags affected: N,Z

DEY - Decrement Y index register.
flags affected: N,Z

EOR - Exclusive-or memory with accumulator.
flags affected: N,Z

INC - Increment memory.
flags affected: N,Z

INX - Increment X index register.
flags affected: N,Z

INY - Increment Y index register.
flags affected: N,Z

JMP - Jump to new location.

JSR - Jump to a subroutine. Pushes the program counter+2 onto the stack and then jumps to the location.

LDA - Load accumulator from memory.
flags affected: N,Z

LDX - Load X index register from memory.
flags affected: N,Z

LDY - Load Y index register from memory.
flags affected: N,Z

LSR - Logical shift right. (Bit zero goes to carry, and a zero is shifted into bit 7.)
flags affected: N(=0),Z

NOP - No operation.

ORA - Logical 'or' accumulator with memory.
flags affected: N,Z

PHA - Push accumulator onto the stack.

PHP - Push processor status register onto the stack.

PLA - Pull accumulator from the stack.
flags affected: N,Z

PLP - Pull procesor status register from the stack.
flags affected: restored

ROL - Rotate left through carry. (Carry is shifted into bit 0 and bit 7 is shifted into carry.)
flags affected: N,Z,C

ROR - Rotate right through carry. (Carry is shifted into bit 7 and bit 0 is shifted into carry.)
flags affected: N,Z,C

RTI - Return from interrupt. Pulls status register and program counter off the stack.
flags affected: restored

RTS - Return from subroutine. Pulls an address off the stack, adds 1 and jumps to that location.

SBC - Subtracts memory from accumulator with carry. (Carry acts as an inverted borrow.)
flags affected: N,Z,C,V (Z is invalid if in decimal mode).

SEC - Set the carry flag.
flags affected: C(=1)

SED - Set the decimal mode flag.
flags affected: D(=1)

SEI - Set the interrupt disable flag.
flags affected: I(=1)

STA - Store accumulator in memory.

STX - Store the X-index register in memory.

STY - Store the Y-index register in memory.

TAX - Transfer the accumulator to the X index register.
flags affected: N,Z

TAY - Transfer the accumulator to the Y index register.
flags affected: N,Z

TSX - Transfer the stack pointer to the X index register.
flags affected: N,Z

TXA - Transfer the X index register to the accumulator.
flags affected: N,Z

TXS - Transfer the X index register to the stack pointer.

TYA - Transfer the Y index register to the accumulator.
flags affected: N,Z

17. THE 6502 ADDRESSING MODES

N.B. All 16-bit addresses are stored in memory with the least significant byte first.

17.1 IMMEDIATE ADDRESSING

The operand is contained in the second byte of the instruction.

Length:	2 bytes
Assembler Notation:	#<expression>
Example:	LDA #CR

17.2 ABSOLUTE ADDRESSING

The 2nd and 3rd bytes of the instruction form the effective address.

Length: 3 bytes
Assembler Notation: <expression>
Example: INC 34975

17.3 ABSOLUTE,X ADDRESSING

The effective address is formed by adding the X-register to the address in the 2nd and 3rd bytes of the instruction.

Length: 3 bytes
Assembler Notation: <expression>,X
Example: CMP TABLE,X

17.4 ABSOLUTE,Y ADDRESSING

The effective address is formed by adding the Y-register to the address in th 2nd and 3rd bytes of the instruction.

Length: 3 bytes
Assembler Notation: <expression>,Y
Example: INC $1000,Y

17.5 ZERO PAGE ADDRESSING

The second byte of the instruction is the low-order 8 bits of the effective address; the high-order byte is zero.

Length: 2 bytes
Assembler Notation: <expression>
Example: INC 100

17.6 ZERO PAGE,X ADDRESSING

The X register is added to the 2nd byte of the instruction to give the low-order 8 bits of the effective address; the high-order byte is always zero.

Length: 2 bytes
Assembler Notation: <expression>,X
Example: STA BUFFER,X

17.7 ZERO PAGE,Y ADDRESSING

The Y register is added to the 2nd byte of the instruction to give the low-order 8 bits of the effective address; the high-order byte is always zero.

Length: 2 bytes
Assembler Notation: <expression>,Y
Example: STX $33,Y

17.8 RELATIVE ADDRESSING

The 2nd byte of the instruction is a signed offset which is added to the program counter to give the effective address. The assembler automatically calculates the offset from the operand given.

Length: 2 bytes
Assembler Notation: <expression>
Example: BNE LOOP

17.9 ACCUMULATOR ADDRESSING

The accumulator is the operand of the instruction.

Length: 1 byte
Assembler Notation: A
Example: LSR A

17.10 IMPLIED ADDRESSING

This addressing mode is implied by the instruction; no operand exists.

Length: 1 byte
Example: DEY

17.11 INDIRECT ADDRESSING

The 2nd and 3rd bytes of the instruction contain a pointer to the 16-bit effective address of the instruction. Due to an error in the 6502's design, this will not work correctly if the 2nd and 3rd bytes of the instruction cross a page boundary.

Length: 3 bytes
Assembler Notation: (<expression>)
Example: JMP (VECTOR)

17.12 INDIRECT,Y ADDRESSING

The effective address is calculated by adding the Y register to a 16-bit address contained in page zero which is pointed to by the 2nd byte of the instruction.

Length: 2 bytes
Assembler Notation: (<expression>),Y
Example: LDA (PTR),Y

17.13 INDIRECT,X ADDRESSING

The 2nd byte of the instruction and the X register are added to give the address of two locations in page zero which hold the effective address.

Length: 2 bytes
Assembler Notation: (<expression>,X)
Example: LDA (BRKTAB,X)

The notations for zero page and absolute addressing modes are the same - the assembler decides which mode to use.

%00000000	@000	000	$00	BRK IMPLIED
%00000001	@001	001	$01	ORA INDIRECT,X
%00000010	@002	002	$02	UNUSED
%00000011	@003	003	$03	UNUSED
%00000100	@004	004	$04	UNUSED
%00000101	@005	005	$05	ORA ZERO PAGE
%00000110	@006	006	$06	ASL ZERO PAGE
%00000111	@007	007	$07	UNUSED
%00001000	@010	008	$08	PHP IMPLIED
%00001001	@011	009	$09	ORA IMMEDIATE
%00001010	@012	010	$0A	ASL ACCUMULATOR
%00001011	@013	011	$0B	UNUSED
%00001100	@014	012	$0C	UNUSED
%00001101	@015	013	$0D	ORA ABSOLUTE
%00001110	@016	014	$0E	ASL ABSOLUTE
%00001111	@017	015	$0F	UNUSED
%00010000	@020	016	$10	BPL RELATIVE
%00010001	@021	017	$11	ORA INDIRECT,Y
%00010010	@022	018	$12	UNUSED
%00010011	@023	019	$13	UNUSED
%00010100	@024	020	$14	UNUSED
%00010101	@025	021	$15	ORA ZERO PAGE,X
%00010110	@026	022	$16	ASL ZERO PAGE,X
%00010111	@027	023	$17	UNUSED
%00011000	@030	024	$18	CLC IMPLIED
%00011001	@031	025	$19	ORA ABSOLUTE,Y
%00011010	@032	026	$1A	UNUSED
%00011011	@033	027	$1B	UNUSED
%00011100	@034	028	$1C	UNUSED
%00011101	@035	029	$1D	ORA ABSOLUTE,X
%00011110	@036	030	$1E	ASL ABSOLUTE,X
%00011111	@037	031	$1F	UNUSED
%00100000	@040	032	$20	JSR ABSOLUTE
%00100001	@041	033	$21	AND INDIRECT,X
%00100010	@042	034	$22	UNUSED
%00100011	@043	035	$23	UNUSED
%00100100	@044	036	$24	BIT ZERO PAGE
%00100101	@045	037	$25	AND ZERO PAGE
%00100110	@046	038	$26	ROL ZERO PAGE
%00100111	@047	039	$27	UNUSED
%00101000	@050	040	$28	PLP IMPLIED
%00101001	@051	041	$29	AND IMMEDIATE
%00101010	@052	042	$2A	ROL ACCUMULATOR
%00101011	@053	043	$2B	UNUSED
%00101100	@054	044	$2C	BIT ABSOLUTE
%00101101	@055	045	$2D	AND ABSOLUTE
%00101110	@056	046	$2E	ROL ABSOLUTE
%00101111	@057	047	$2F	UNUSED
%00110000	@060	048	$30	BMI RELATIVE
%00110001	@061	049	$31	AND INDIRECT,X
%00110010	@062	050	$32	UNUSED
%00110011	@063	051	$33	UNUSED
%00110100	@064	052	$34	UNUSED
%00110101	@065	053	$35	AND ZERO PAGE,X
%00110110	@066	054	$36	ROL ZERO PAGE,X

```
%00110111 @067 055 $37 UNUSED
%00111000 @070 056 $38 SEC IMPLIED
%00111001 @071 057 $39 AND ABSOLUTE,Y
%00111010 @072 058 $3A UNUSED
%00111011 @073 059 $3B UNUSED
%00111100 @074 060 $3C UNUSED
%00111101 @075 061 $3D AND ABSOLUTE,X
%00111110 @076 062 $3E ROL ABSOLUTE,X
%00111111 @077 063 $3F UNUSED
%01000000 @100 064 $40 RTI IMPLIED
%01000001 @101 065 $41 EOR INDIRECT
%01000010 @102 066 $42 UNUSED
%01000011 @103 067 $43 UNUSED
%01000100 @104 068 $44 UNUSED
%01000101 @105 069 $45 EOR ZERO PAGE
%01000110 @106 070 $46 LSR ZERO PAGE
%01000111 @107 071 $47 UNUSED
%01001000 @110 072 $48 PHA IMPLIED
%01001001 @111 073 $49 EOR IMMEDIATE
%01001010 @112 074 $4A LSR ACCUMULATOR
%01001011 @113 075 $4B UNUSED
%01001100 @114 076 $4C JMP ABSOLUTE
%01001101 @115 077 $4D EOR ABSOLUTE
%01001110 @116 078 $4E LSR ABSOLUTE
%01001111 @117 079 $4F UNUSED
%01010000 @120 080 $50 BVC RELATIVE
%01010001 @121 081 $51 EOR INDIRECT,Y
%01010010 @122 082 $52 UNUSED
%01010011 @123 083 $53 UNUSED
%01010100 @124 084 $54 UNUSED
%01010101 @125 085 $55 EOR ZERO PAGE,X
%01010110 @126 086 $56 LSR ZERO PAGE,X
%01010111 @127 087 $57 UNUSED
%01011000 @130 088 $58 CLI IMPLIED
%01011001 @131 089 $59 EOR ABSOLUTE,Y
%01011010 @132 090 $5A UNUSED
%01011011 @133 091 $5B UNUSED
%01011100 @134 092 $5C UNUSED
%01011101 @135 093 $5D EOR ABSOLUTE,X
%01011110 @136 094 $5E LSR ABSOLUTE,X
%01011111 @137 095 $5F UNUSED
%01100000 @140 096 $60 RTS IMPLIED
%01100001 @141 097 $61 ADC INDIRECT,X
%01100010 @142 098 $62 UNUSED
%01100011 @143 099 $63 UNUSED
%01100100 @144 100 $64 UNUSED
%01100101 @145 101 $65 ADC ZERO PAGE
%01100110 @146 102 $66 ROR ZERO PAGE
%01100111 @147 103 $67 UNUSED
%01101000 @150 104 $68 PLA IMPLIED
%01101001 @151 105 $69 ADC IMMEDIATE
%01101010 @152 106 $6A ROR ACCUMULATOR
%01101011 @153 107 $6B UNUSED
%01101100 @154 108 $6C JMP INDIRECT
%01101101 @155 109 $6D ADC ABSOLUTE
%01101110 @156 110 $6E ROR ABSOLUTE
%01101111 @157 111 $6F UNUSED
%01110000 @160 112 $70 BVS RELATIVE
%01110001 @161 113 $71 ADC INDIRECT,Y
```

```
%01110010 @162 114 $72 UNUSED
%01110011 @163 115 $73 UNUSED
%01110100 @164 116 $74 UNUSED
%01110101 @165 117 $75 ADC ZERO PAGE,X
%01110110 @166 118 $76 ROR ZERO PAGE,X
%01110111 @167 119 $77 UNUSED
%01111000 @170 120 $78 SEI IMPLIED
%01111001 @171 121 $79 ADC ABSOLUTE,Y
%01111010 @172 122 $7A UNUSED
%01111011 @173 123 $7B UNUSED
%01111100 @174 124 $7C UNUSED
%01111101 @175 125 $7D ADC ABSOLUTE,X
%01111110 @176 126 $7E ROR ABSOLUTE,X
%01111111 @177 127 $7F UNUSED
%10000000 @200 128 $80 UNUSED
%10000001 @201 129 $81 STA INDIRECT,X
%10000010 @202 130 $82 UNUSED
%10000011 @203 131 $83 UNUSED
%10000100 @204 132 $84 STY ZERO PAGE
%10000101 @205 133 $85 STA ZERO PAGE
%10000110 @206 134 $86 STX ZERO PAGE
%10000111 @207 135 $87 UNUSED
%10001000 @210 136 $88 DEY IMPLIED
%10001001 @211 137 $89 UNUSED
%10001010 @212 138 $8A TXA IMPLIED
%10001011 @213 139 $8B UNUSED
%10001100 @214 140 $8C STY ABSOLUTE
%10001101 @215 141 $8D STA ABSOLUTE
%10001110 @216 142 $8E STX ABSOLUTE
%10001111 @217 143 $8F UNUSED
%10010000 @220 144 $90 BCC RELATIVE
%10010001 @221 145 $91 STA INDIRECT,Y
%10010010 @222 146 $92 UNUSED
%10010011 @223 147 $93 UNUSED
%10010100 @224 148 $94 STY ZERO PAGE,X
%10010101 @225 149 $95 STA ZERO PAGE,X
%10010110 @226 150 $96 STX ZERO PAGE,Y
%10010111 @227 151 $97 UNUSED
%10011000 @230 152 $98 TYA IMPLIED
%10011001 @231 153 $99 STA ABSOLUTE,Y
%10011010 @232 154 $9A TXS IMPLIED
%10011011 @233 155 $9B STA ABSOLUTE,X
%10011100 @234 156 $9C UNUSED
%10011101 @235 157 $9D UNUSED
%10011110 @236 158 $9E UNUSED
%10011111 @237 159 $9F UNUSED
%10100000 @240 160 $A0 LDY IMMEDIATE
%10100001 @241 161 $A1 LDA INDIRECT,X
%10100010 @242 162 $A2 LDX IMMEDIATE
%10100011 @243 163 $A3 UNUSED
%10100100 @244 164 $A4 LDY ZERO PAGE
%10100101 @245 165 $A5 LDA ZERO PAGE
%10100110 @246 166 $A6 LDX ZERO PAGE
%10100111 @247 167 $A7 UNUSED
%10101000 @250 168 $A8 TAY IMPLIED
%10101001 @251 169 $A9 LDA IMMEDIATE
%10101010 @252 170 $AA TAX IMPLIED
%10101011 @253 171 $AB UNUSED
%10101100 @254 172 $AC LDY ABSOLUTE
```

```
%10101101 @255 173 $AD LDA ABSOLUTE
%10101110 @256 174 $AE LDX ABSOLUTE
%10101111 @257 175 $AF UNUSED
%10110000 @260 176 $B0 BCS RELATIVE
%10110001 @261 177 $B1 LDA INDIRECT,Y
%10110010 @262 178 $B2 UNUSED
%10110011 @263 179 $B3 UNUSED
%10110100 @264 180 $B4 LDY ZERO PAGE,X
%10110101 @265 181 $B5 LDA ZERO PAGE,X
%10110110 @266 182 $B6 LDX ZERO PAGE,Y
%10110111 @267 183 $B7 UNUSED
%10111000 @270 184 $B8 CLV IMPLIED
%10111001 @271 185 $B9 LDA ABSOLUTE,Y
%10111010 @272 186 $BA TSX IMPLIED
%10111011 @273 187 $BB UNUSED
%10111100 @274 188 $BC LDY ABSOLUTE,X
%10111101 @275 189 $BD LDA ABSOLUTE,X
%10111110 @276 190 $BE LDX ABSOLUTE,Y
%10111111 @277 191 $BF UNUSED
%11000000 @300 192 $C0 CPY IMMEDIATE
%11000001 @301 193 $C1 CMP INDIRECT,X
%11000010 @302 194 $C2 UNUSED
%11000011 @303 195 $C3 UNUSED
%11000100 @304 196 $C4 CPY ZERO PAGE
%11000101 @305 197 $C5 CMP ZERO PAGE
%11000110 @306 198 $C6 DEC ZERO PAGE
%11000111 @307 199 $C7 UNUSED
%11001000 @310 200 $C8 INY IMPLIED
%11001001 @311 201 $C9 CMP IMMEDIATE
%11001010 @312 202 $CA DEX IMPLIED
%11001011 @313 203 $CB UNUSED
%11001100 @314 204 $CC CPY ABSOLUTE
%11001101 @315 205 $CD CMP ABSOLUTE
%11001110 @316 206 $CE DEC ABSOLUTE
%11001111 @317 207 $CF UNUSED
%11010000 @320 208 $D0 BNE RELATIVE
%11010001 @321 209 $D1 CMP INDIRECT,Y
%11010010 @322 210 $D2 UNUSED
%11010011 @323 211 $D3 UNUSED
%11010100 @324 212 $D4 UNUSED
%11010101 @325 213 $D5 CMP ZERO PAGE,X
%11010110 @326 214 $D6 DEC ZERO PAGE,X
%11010111 @327 215 $D7 UNUSED
%11011000 @330 216 $D8 CLD IMPLIED
%11011001 @331 217 $D9 CMP ABSOLUTE,Y
%11011010 @332 218 $DA UNUSED
%11011011 @333 219 $DB UNUSED
%11011100 @334 220 $DC UNUSED
%11011101 @335 221 $DD CMP ABSOLUTE,X
%11011110 @336 222 $DE DEC ABSOLUTE,X
%11011111 @337 223 $DF UNUSED
%11100000 @340 224 $E0 CPX IMMEDIATE
%11100001 @341 225 $E1 SBC INDIRECT,X
%11100010 @342 226 $E2 UNUSED
%11100011 @343 227 $E3 UNUSED
%11100100 @344 228 $E4 CPX ZERO PAGE
%11100101 @345 229 $E5 SBC ZERO PAGE
%11100110 @346 230 $E6 INC ZERO PAGE
```

```
%11100111 @347 231 $E7 UNUSED
%11101000 @340 232 $E8 INX IMPLIED
%11101001 @351 233 $E9 SBC IMMEDIATE
%11101010 @352 234 $EA NOP IMPLIED
%11101011 @353 235 $EB UNUSED
%11101100 @354 236 $EC CPX ABSOLUTE
%11101101 @355 237 $ED SBC ABSOLUTE
%11101110 @356 238 $EE INC ABSOLUTE
%11101111 @357 239 $EF UNUSED
%11110000 @360 240 $F0 BEQ RELATIVE
%11110001 @361 241 $F1 SBC INDIRECT,Y
%11110010 @362 242 $F2 UNUSED
%11110011 @363 243 $F3 UNUSED
%11110100 @364 244 $F4 UNUSED
%11110101 @365 245 $F5 SBC ZERO PAGE,X
%11110110 @366 246 $F6 INC ZERO PAGE,X
%11110111 @367 247 $F7 UNUSED
%11111000 @370 248 $F8 SED IMPLIED
%11111001 @371 249 $F9 SBC ABSOLUTE,Y
%11111010 @372 250 $FA UNUSED
%11111011 @373 251 $FB UNUSED
%11111100 @374 252 $FC UNUSED
%11111101 @375 253 $FD SBC ABSOLUTE,X
%11111110 @376 254 $FE INC ABSOLUTE,X
%11111111 @377 255 $FF UNUSED
```

19. THE GRAPHICS ROUTINES

These routines are designed to facilitate the manipulation of sprites and screen data. There are 138 different routines in 10K of code; these are the same routines that are used in BASIC LIGHTNING and WHITE LIGHTNING. The sound routines are not included since they simply store values in the SID's registers, but they are listed as assembly language source code in section 20 for completeness.

If you have the disk version of MACHINE LIGHTNING, you will find the graphics routines saved under the filename "IDEAL". They can be loaded using the MLOAD command in 64-MAC/MON. Remember that the graphics routines cannot be used with the high memory version of 64 MAC/MON.

At this point, we assume that you are familiar with the graphics commands in BASIC LIGHTNING, and that you understand programming in assembly language.

19.1 MEMORY ORGANISATION

The graphics routines use the following locations in zero page:

$02-$05, $07-$08, $0A-$12, $49-$4E, $5C-$5F, $65-$66, $69-$6E, $70-$8A.

Therefore none of these locations should be used in your own routines because the graphics routines will corrupt them.

The memory is organised as follows:

$0000 to $00FF	zero page
$0100 to $01FF	6502 stack.
$0200 to $03FF	KERNAL system variables.
$0400 to $04FF	8 sets of sprite variables.
$0500 to $05FF	reserved for interrupt scrolling buffer.
$0600 to $06FF	reserved for foreground scrolling buffer.
$0700 to $07FF	graphics routines sprite pointers.
$0800 to $97FF	free for your own program and sprites.
$0800 to $47FF	64-MAC/MON, if present.
$9800 to $BFD6	IDEAL graphics routines.
$BFD7 to $BFFF	IDEAL system variables.
$C000 to $C7FF	Character set and/or hardware sprite data.
$C800 to $CBFF	Text screen.
$CC00 to $CFFF	Hires attribute screen.
$D000 to $DFFF	I/O devices.
$D800 to $DBFF	Colour memory for text or secondary attribute memory for hi-res.
$E000 to $FFFF	KERNAL ROM
$E000 to $FFFF	Hires screen pixel data.

You may notice that the KERNAL ROM and the hi-res screen pixel data share the same memory - the graphics routines switch out the KERNAL ROM whenever the pixel data underneath has to be used.

19.2 MEMORY LOCATIONS

The following definitions should be put at the top of your programs:

```
IDEAL = $9800            ;ENTRY POINT
INT10SAVE = $BFD7        ;10 LOCATIONS WHICH MUST BE SAVED ON INTERRUPT
SETBASE = $BFE1          ;OFFSET FOR ACCESSING SPRITE VARIABLES
LOMEM = $BFE2            ;LOWER LIMIT (PAGE) FOR SPRITES
HIMEM = $BFE3            ;UPPER LIMIT (PAGE) FOR SPRITES
SOUNDCONTROL = $BFE4     ;3 BYTES FOR SOUND REG. VALUES
SOUNDLENGTH = $BFE7      ;LENGTH OF SOUNDS IN 60THS OF SECONDS
INTFLAG = $BFEA          ;TOP BIT SET=ENABLE USER INTERRUPT ROUTINE
BUFFBASE = $BFEC         ;BASE (PAGE) ADDR. OF WRAP BUFFER
ERRORADDR = $BFF6        ;ADDRESS OF ERROR ROUTINE
INTADDR = $BFFA          ;ADDRESS OF USER INTERRUPT ROUTINE
SPST = $BFFC             ;ADDRESS OF START OF SPRITES
SPND = $BFFE             ;ADDRESS OF END OF SPRITES
VAL1 = 73                ;3 LOCATIONS FOR PASSING PARAMETERS
VAL2 = 75
VAL3 = 77
SPN = $0400              ;SPRITE VARIABLES
COL = $0402
ROW = $0404
WID = $0406
HGT = $0408
SPN2 = $040A
COL2 = $040C
ROW2 = $040E
NUM = $0410
ICL = $0412
ATR = $0414
CCOL = $0416
CROW = $0418
```

19.3 PASSING PARAMETERS

Values are passed to and from the routines using the sprite variables SPN, COL, ROW, WID, HGT, SPN2, COL2, ROW2, NUM, ICL, ATR, CCOL and CROW as well as the zero page locations VAL1, VAL2 and VAL3. ICL is equivalant to INC in BASIC LIGHTNING - INC cannot be used because it is a 6502 op-code numemonic. There are actually 8 different sets of variables that can be used:

Set 0 is stored from $0400 to $041F
Set 1 is stored from $0420 to $043F
Set 2 is stored from $0440 to $045F
Set 3 is stored from $0460 to $047F
Set 4 is stored from $0480 to $049F
Set 5 is stored from $04A0 to $04BF
Set 6 is stored from $04C0 to $04DF
Set 7 is stored from $04E0 to $04FF

To use an alternative set, the set number must be multiplied by 32 and stored in SETBASE:

```
        LDA #5*32
        STA SETBASE     ;SELECT VARIABLE SET 5
```

If you use different sets of variables in this way, and you want to write routines which will work with all different variable sets, the sprite variables cannot be accessed directly but have to be referenced using the offset in SETBASE:

```
        LDY SETBASE
        LDA #0
        STA HGT+1,Y      ;HGT=16
        LDA #16
        STA HGT,Y
```

19.4 ACCESSING THE ROUTINES

There is only one entry point for the graphics routines - the label IDEAL. The number of the routine should be placed in the accumulator, the parameters having been set up in the sprite variables or in VAL1, VAL2 and VAL3 as appropriate beforehand. For example, to call routine no. 42, use:

```
        LDA #42
        JSR IDEAL
```

The A, X and Y registers are preserved by the routines. When VAL2 and VAL3 are used to pass parameters to a routine, VAL2 is the first parameter in BASIC Lightning syntax, and VAL3 is the second.

Here is a full list:

ACC	NAME	PARAMETERS	RETURNS VALUE
0	INIT		
1	SCLR	SPN,ATR	
2	SPRITE	SPN,WID,HGT	
3	WIPE	SPN	
4	RESET		
5	H38COL		

6	LORES		
7	HIRES		
8	PLOT	SPN,COL,ROW	
9	BOX	SPN,COL,ROW,WID,HGT	
10	POLY	SPN,COL,ROW,WID,GHT,NUM, ICL	
11	DRAW	SPN,COL,ROW,COL2,ROW2	
12	MODE	VAL2	
13	S2COL		
14	S4COL		
15	H40COL		
16	SCRLX	VAL2	
17	WRR1	SPN,COL,ROW,WID,HGT	
18	WRL1	SPN,COL,ROW,WID,HGT	
19	SCR1	SPN,COL,ROW,WID,HGT	
20	SCL1	SPN,COL,ROW,WID,HGT	
21	WRR2	SPN,COL,ROW,WID,HGT	
22	WRL2	SPN,COL,ROW,WID,HGT	
23	SCR2	SPN,COL,ROW,WID,HGT	
24	SCL2	SPN,COL,ROW,WID,HGT	
25	WRR8	SPN,COL,ROW,WID,HGT	
26	WRL8	SPN,COL,ROW,WID,HGT	
27	SCR8	SPN,COL,ROW,WID,HGT	
28	SCL8	SPN,COL,ROW,WID,HGT	
29	ATTR	SPN,COL,ROW,WID,HGT	
30	ATTL	SPN,COL,ROW,WID,HGT	
31	ATTUP	SPN,COL,ROW,WID,HGT	
32	ATTDN	SPN,COL,ROW,WID,HGT	
33	CHAR	SPN,COL,ROW,NUM	
34	WINDOW	VAL2	
35	MULTI		
36	MONO		
37	TBORDER	VAL2	
38	HBORDER	VAL2	
39	TPAPER	VAL2	
40	HPAPER	VAL2	
41	WRAP	SPN,COL,ROW,WID,HGT,NUM	
42	SCROLL	SPN,COL,ROW,WID,HGT,NUM	
43	INK	VAL2	
44	SETA	SPN,COL,ROW,WID,HGT,ATR	
45	ATTGET	SPN,COL,ROW	ATR
46	ATT2ON		
47	ATTON		
48	ATTOFF		
49	MIR	SPN,COL,ROW,WID,HGT	
50	MAR	SPN,COL,ROW,WID,HGT	
51	WCLR	SPN,COL,ROW,WID,HGT,ATR	
52	INV	SPN,COL,ROW,WID,HGT	
53	SPIN	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2	
54	MOVBLK	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2	
55	MOVXOR	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2	
56	MOVAND	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2	
57	MOVOR	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2	
58	MOVATT	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2	

59	XPANDX	SPN,COL,ROW,WID,HGT;SPN2,COL2,ROW2
60	XPANDY	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
61	GETBLK	SPN,COL,ROW
62	PUTBLK	SPN,COL,ROW
63	CPYBLK	SPN,COL,ROW
64	GETXOR	SPN,COL,ROW
65	PUTXOR	SPN,COL,ROW
66	CPYXOR	SPN,SPN2
67	GETOR	SPN,COL,ROW
68	PUTOR	SPN,COL,ROW
69	CPYOR	SPN,SPN2
70	GETAND	SPN,COL,ROW
71	PUTAND	SPN,COL,ROW
72	CPYAND	SPN,SPN2
73	DBLANK	
74	DSHOW	
75	PUTCHR	SPN,COL,ROW,NUM
76	LCASE	
77	UCASE	
78	SPRCONV	SPN,COL,ROW,SPN2
79	.ON	VAL2
80	.OFF	VAL2
81	.SET	VAL2,VAL3
82	FLIPA	SPN,COL,ROW,WID,HGT
83	.4COL	VAL2
84	.2COL	VAL2
85	.COL0	VAL2
86	.COL1	VAL2
87	.XPANDX	VAL2
88	.SHRINK	VAL2
89	.XPANDY	VAL2
90	.SHRINKY	VAL2
91	.XPOS	VAL2,VAL3
92	.YPOS	VAL2,VAL3
93	.COL	VAL2,VAL3
94	.OVER	VAL2
95	.UNDER	VAL2
96	SWAPATT	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
97	DTCTON	
98	DTCTOFF	
99	BLK%BLK	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
100	OR%BLK	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
101	AND%BLK	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
102	XOR%BLK	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
103	BLK%OR	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
104	OR%OR	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
105	AND%OR	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
106	XOR%OR	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
107	BLK%AND	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
108	OR%AND	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
109	AND%AND	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
110	XOR%AND	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2
111	BLK%XOR	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2

112	OR&XOR	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2	
113	AND&XOR	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2	
114	XOR&XOR	SPN,COL,ROW,WID,HGT,SPN2,COL2,ROW2	
115	RESEQ		
116	FLIP	SPN,COL,ROW,WID,HGT	
117	.HIT	VAL2	VAL2
118	SCAN	SPN,COL,ROW,WID,HGT	VAL1
119	POINT	SPN,COL,ROW	VAL1
120	DFA	SPN	VAL1,WID,HGT
121	AFA2	SPN	VAL1,WID,HGT
122	AFA	SPN	VAL1,WID,HGT
123	KB	VAL2	VAL2
124	FIRE1		VAL1
125	FIRE2		VAL1
126	JS1		VAL1
127	JS2		VAL1
128	SCRLY	VAL2	
129	RELOCATE	NUM	
130	SCRSET		
131	ICOLD	LOMEM,HIMEM,SPST	
132	TRACK	SPN,SPN2	
133	MOVE	SPN,COL,ROW	
134	PLAY	SPN,COL,ROW	
135	RPLAY	SPN,COL,ROW	
136	R.XPOS	VAL2	VAL2
137	R.YPOS	VAL2	VAL2

All but six of these routines are BASIC LIGHTNING commands. The six which aren't are explained here:

19.5 INIT, ICOLD, SCRSET and RELOCATE.

19.5.1 INIT (ACC = 0)

This is used to set up the screen display and the graphics routines, and it should be called at the start of your program:

```
START   LDA #0     ;INIT
        JSR IDEAL
```

Note that after using this routine you will not be able to use the function keys from 64 MAC/MON.

19.5.2 ICOLD (ACC = 131)

This clears the sprite storage and sets up all the sprite pointers. LOMEM and HIMEM should be set to the lower and upper memory limits (/256) over which sprites cannot go, and SPST should be set to the start of memory to be used for sprites; normally this is the same as LOMEM.

Example: Initialise sprite storage to go from $5000 to $7800:

```
        LDA    #0
        STA    SPST
        LDA    #$50
        STA    SPST+1
        STA    LOMEM
        LDA    #$78
        STA    HIMEM
        LDA    #131
        JSR    IDEAL
```

19.5.3 SCRSET (ACC = 130)

This routine will put the computer into LORES mode if in HIRES mode without a window set up. It is used in BASIC LIGHTNING when printing the "READY" message:

```
        LDA    #130
        JSR    IDEAL
```

19.5.4 RELOCATE (ACC = 129)

This moves the sprites up or down in memory by a signed offset in NUM. SPST and SPND are altered, but LOMEM and HIMEM are not.

If you have used WHITE LIGHTNING you will notice that ISPRITE, DSPRITE and CLR are not defined here. That is because they are written in FORTH using SPRITE, WIPE, RELOCATE and RESET:

```
FORTH DEFINITIONS HEX
: DSPRITE SPND @ WIPE SPND @ - NUM ! RELOCATE ;
: ISPRITE WID C@ HGT C@ OVER OVER
  -0A * * 7 - NUM ! DFA -1 =
  IF SPN C@ IF RELOCATE THEN THEN
  HGT ! WID ! SPRITE ;
: CLR SPND @ 1 - SPST ! RESET ;
```

Using RELOCATE with NUM=0 will set up all the sprite pointers and should be used after loading sprites from tape or disk.

To load new sprites in above what already exists:

```
SETLFS = $FFBA
SETNAM = $FFBD
LOAD = $FFD5

TOP     LDX #1                  ;device 1=tape
                                ;(use 8 for disk)
        LDY #0                  ;relocated load
        JSR SETLFS
        LDA #NAME2-NAME
        LDX #NAME&255
        LDY #NAME/256
        JSR SETNAM              ;set filename
        LDA SPND                ;load sprites at SPND-1
        SEC
        SBC #1
        TAX
        LDA SPND+1
        SBC #0
        TAY
        LDA #0
```

```
        JSR LOAD
        LDY SETBASE
        LDA #0
        STA NUM,Y
        STA NUM+1,Y
        LDA #129
        JSR RELOCATE
        JMP START
NAME    .BYTE     'SPRITES'
NAME2
```

Note: sprites should not usually be loaded in by a program in this way; it is more convenient to link the sprites with the program before saving it. This is described later, in section 19.8.

19.5.5 R.XPOS and R.YPOS (ACC = 136 and 137)

These are used when reading the position of a hardware sprite. .XPOS and .YPOS should be used only to set a sprite's position. For example, to read the position of hardware sprite no. 3 use:

```
        LDA #3
        STA VAL2
        LDA #0
        STA VAL2+1
        LDA #136
        JSR IDEAL
        LDA VAL2
        LDX VAL2+1
```

This would put the X-position of sprite no. 3 into the A and X registers.

19.6 ERRORS

When a graphics routine encounters nonsensical parameters it does an indirect jump on ERRORADDR. This normally points to a BRK instruction; however it can be altered to point to your own error routine:

```
        LDA     #ERROR&255
        STA     ERRORADDR
        LDA     #ERROR/256
        STA     ERRORADDR+1
```

The error number is in the accumulator after an error:

ACC	ERROR
0	NO ROOM
1	CORRUPTED SPRITE
2	REDEFINED SPRITE
3	NO SUCH SPRITE
4	DELETE SPRITE ZERO
5	OUT OF RANGE

The stack is not reset; this has to be done by your own routine:

```
CHROUT          =$FFD2

ERROR           LDX #$FF
                TXS
                PHA
                LDA #130            ;SCRSET
                JSR IDEAL
                LDX #0
ERRORLOOP       LDA ERRORMSG,X
                BEQ ENDERROR
                JSR CHROUT
                INX
                JMP ERRORLOOP
ENDERROR        PLA
                ORA #'0'
                JSR CHROUT
                JMP MAIN

ERRORMSG        .BYTE   13, 'ERROR#',0
```

19.7 USING INTERRUPTS

Mastering machine code gives most programmers access to the speed of commercial games, but often the smoothness and continuity are lacking. The problem is that some parts of the program need to execute at regular intervals, and trying to achieve this can involve a lot of calculation and wasted processor time. The solution is to use interrupts to execute particular sections of code. Machine Lightning does this for you, using the vector INTADDR and the flag INTFLAG. The 64's interrupt occurs 60 times a second, so that a background subroutine can be executed at this frequency, or by counting interrupts, at lower frequencies.

The top bit of INTFLAG, when set, enables keyboard scanning and the execution of the user's background subroutine. Clearing the top bit disables execution. To use a routine in background mode, you must point the vector at INTADDR to it:

```
LDA #%00000000
STA INTFLAG             ;DISABLE INTERRUPTS
LDA #<BACKGROUND        ;CHANGE VECTOR
STA INTADDR
LDA #>BACKGROUND
STA INTADDR+1
LDA #%10000000          ;ENABLE INTERRUPTS
STA INTFLAG
```

Note that interrupts have to be disabled while changing the vector since an interrupt might occur when only one byte of the vector has been changed.

The first thing that the background routine must do is to save all temporary locations which are used in zero page, and the ten locations from INT10SAVE onwards. Also, the wrap buffer pointer at BUFFBASE and the variable pointer at SETBASE must be altered. After execution of your routine, all the locations saved must be restored:

```
BACKGROUND      LDA #%00000000          ;DISABLE INTERRUPTS
                STA INTFLAG
                LDY #0
BACK1           LDX ZPSAVE,Y            ;SAVE LOCATIONS IN ZERO PAGE.
                LDA 0,X
                STA ZPSTORE,Y
                INY
                CPY #ENDZPSAVE-ZPSAVE
                BNE BACK1
                LDY #256-10
BACK2           LDA INT10SAVE+10-256,Y  ;SAVE LOCATIONS AT INT10SAVE.
                PHA
                INY
                BNE BACK2
                LDA BUFFBASE
                PHA
                LDA SETBASE
                PHA
                LDA #7*32
                STA SETBASE             ;SET 7
                LDA #11
                SEC
                SBC BUFFBASE            ;SWAP TO OTHER SCROLLING BUFFER.
                STA BUFFBASE
                JSR INTSUB              ;YOUR OWN ROUTINE
                PLA
                STA SETBASE
                PLA
                STA BUFFBASE
                LDY #10
BACK3A          PLA
                STA INT10SAVE-1,Y       ;RESTORE LOCATIONS AT INT10SAVE.
                DEY
                BNE BACK3A
                LDY #ENDZPSAVE-ZPSAVE
BACK3           LDA ZPSTORE-1,Y         ;RESTORE ZERO PAGE.
                LDX ZPSAVE-1,Y
                STA 0,X
                DEY
                BNE BACK3
                LDA #%10000000          ;RE-ENABLE INTERRUPTS.
                STA INTFLAG
                RTS
ZPSAVE          .BYTE  $02,$03,$04,$05,$07,$08,$0A,$0B
                .BYTE  $0C,$0D,$0E,$0F,$10,$11,$12,$49
                .BYTE  $4A,$4B,$4C,$4D,$4E,$5C,$5D,$5E
                .BYTE  $5F,$65,$66,$69,$6A,$6B,$6C,$6D
                .BYTE  $6E,$70,$71,$72,$73,$74,$75,$76
                .BYTE  $77,$78,$79,$7A,$7B,$7C,$7D,$7E
                .BYTE  $7F,$80,$81,$82,$83,$84,$85,$86
                .BYTE  $87,$88,$89,$8A
ENDZPSAVE
ZPSTORE         .BLOCK ENDZPSAVE-ZPSAVE
```

If the execution time of the background routine exceeds one 60th of a second, it is not possible to execute it more than 30 times a second; if it exceeds one 30th of a second it cannot be executed more than 20 times a second, and so on.

Also, as the execution time approaches one 60th of a second, or some multiple of it, less and less processor time will be available for execution of the foreground program. Sometimes it will be necessary to reduce the frequency of execution of the background routine to give more time to execute the foreground program. This can be done by counting interrupts:

```
INTCOUNT        .BYTE 2
INTSUB          DEC  INTCOUNT
                BEQ  DOINT
                RTS
DOINT           LDA  #2
                STA  INTCOUNT
( rest of routine.........)
```

Remember that when an interrupt occurs, the foreground program will stop whatever it is doing, execute the background routine and continue with the foreground program. Suppose that the background program is a sideways scroll of a user defined screen window and the foreground program PUTs a character into the window. A problem arises if an interrupt occurs halfway throught the PUT because the top half of the character will be scrolled before the second half of the character is PUT to the screen. To circumvent this problem, where an operation is carried out on the same screen or sprite data by both the foreground and background programs, the background program should be temporarily disabled by clearing the top bit of INTFLAG, the foreground operation carried out, and the background program re-enabled by setting the top bit of INTFLG again.

19.8 CREATING A STAND-ALONE PROGRAM

Once a program has been written using 64-MAC/MON it must be tested using the monitor commands. First, load in the graphics routines and the sprites created with the Sprite Generator using MLOAD. When the sprites are loaded in, you must specify the start address, otherwise the sprites will load in where the sprite generator saved them- $A000. The object code created by the assembler can now be loaded in using OLOAD if it has not been assembled directly to memory. The first thing that your program must do after INITialising everything (using a call to IDEAL with A=0) is to set up the sprite pointers using a RELOCATE of zero. For example, if your sprites were at $6000, you would use:

```
LDA #0
LDY SETBASE
STA NUM,Y
STA NUM+1,Y             ;NUM=0
STA SPST                ;SPST=$6000
LDA #$60
STA SPST+1
STA LOMEM               ;LOMEM=$60(00)
LDA #$98
STA HIMEM               ;HIMEM=$98(00)
LDA #129
JSR IDEAL               ;RELOCATE
```

You can now test your program using the monitor and tracer commands.

If you want to run a subroutine under interrupt when 64-MAC/MON is loaded, you must be extremely careful not to corrupt the zero page. 64-MAC/MON uses all the available zero page, although it appears to leave it free because the zero page is exchanged with an alternate set of locations whenever memory is accessed by one of the monitor commands or you call a machine-code routine using SYS. Thus, your interrupt routine must not disturb the zero page if you run it while tracing a program or using 64-MAC/MON.

Once you have tested your program, it must be put into a form where it can easily be loaded and run from BASIC. In the following example, we assume that the program to be saved starts at $4800 (and ends at $BFFF, at the end of the graphics routines), and that the entry point also is $4800.

Type NEW and reserve memory from $4800 to $5000 for text. Type in the following program:

```
 10              PROGSTART=$4800          ;START OF PROGRAM
 20              PTR1=$FB
 30              PTR2=$FD
 40              *=$0801
 50              .BYTE   $0D,$08,$0A,$00,$9E,$28,$32,$30
 60              .BYTE   $36,$33,$29,$00,$00,$00
 70 MEM2063      LDY #0
 80              STY PTR1
 90              STY PTR2
100              LDA #$C0
110              STA PTR2+1
120              LDA #$C000-PROGSTART+$0C00/256
130              STA PTR1+1
140              LDX #$C000-PROGSTART/256
150 PROGSHIFT1   DEC PTR2+1
160              DEC PTR1+1
170 PROGSHIFT2   LDA (PTR1),Y
180              STA (PTR2),Y
190              INY
200              BNE PROGSHIFT2
210              DEX
220              BNE PROGSHIFT1
230              LDA #$36
240              STA 1
250              JMP PROGSTART
```

Type "OFFSET $4800" followed by ASM,M. Then save the program object code on tape or disk using MSAVE "BOOT"$5001,$5100.

Now load in your program with the sprites and graphics routines, ready to run, type OFFSET 0 and OSAVE it:

```
OFFSET 0
OSAVE "PRG"$4800,$C000
```

Switch off the computer, and load the LOADER program from BASIC. We want to load the program at $0C00. This implies an offset of $0C00-$4800 = -$3C00 or $C400. After the program has been loaded in using this offset, type NEW. Load in the BOOT program that was MSAVEd earlier:

```
LOAD "BOOT"
```

At this point, the end of the machine-code program will be at $C000-$4800+$0C00 which is $8400. To set the BASIC end-of-text pointer to this value, type:

```
POKE 45,0:POKE 46,132:CLR
```

You can now save your program off to tape or disk just like a normal BASIC program. When it is loaded in and RUN, the computer executes the SYS 2063 which was created by the two .BYTE directives in BOOT, block moves your program back up to its original location ($4800), and executes it.

```
LINE# LOC. OBJECT LABELS         LINE

   60 0000                       ;* * * SOUND ROUTINES * * *

   80 0000                       ;THESE ROUTINES PROVIDE ALL OF THE
   90 0000                       ;SOUND COMMANDS THAT ARE AVAILABLE
  100 0000                       ;FROM BASIC LIGHTNING.

  120 0000                       ;ALL PARAMETERS ARE PASSED IN THE 6502'S
  130 0000                       ;REGISTERS. WHEN A TWO-BYTE VALUE IS
  140 0000                       ;PASSED IN THE A,X REGISTER PAIR,
  150 0000                       ;A ALWAYS HOLDS THE HIGH BYTE AND
  160 0000                       ;X HOLDS THE LOW BYTE.

  180 0000        SOUNDCONTROL  =$BFE4
  190 0000        SOUNDLENGTH   =$BFE7
  200 0000        SID           =$D400

  220 6100                       *=$6100

  240 6100 00     MUL7           .BYTE   0,7,14
  240 6101 07
  240 6102 0E

  260 6103                       ;FRQ: Y=VOICE; A,X=FREQUENCY

  280 6103 48     FRQ            PHA
  290 6104 8A                    TXA
  300 6105 BEFF60                LDX MUL7-1,Y
  310 6108 9D00D4                STA SID,X
  320 610B 68                    PLA
  330 610C 9D01D4                STA SID+1,X
  340 610F 60                    RTS

  360 6110                       ;NOISE: Y=VOICE

  380 6110 B9E3BF NOISE          LDA SOUNDCONTROL-1,Y
  390 6113 2906                  AND #%00000110
  400 6115 0980                  ORA #%10000000
  410 6117 99E3BF                STA SOUNDCONTROL-1,Y
  420 611A 60                    RTS
```

```
LOC. OBJECT LABELS              LINE

6118                            ;PULSE: Y=VOICE; A,X=WIDTH

611B 48     PULSE               PHA
611C B9E3BF                     LDA SOUNDCONTROL-1,Y
611F 2906                       AND #%00000110
6121 0940                       ORA #%01000000
6123 99E3BF                     STA SOUNDCONTROL-1,Y
6126 8A                         TXA
6127 BEFF60                     LDX MUL7-1,Y
612A 9D02D4                     STA SID+2,X
612D 68                         PLA
612E 9D03D4                     STA SID+3,X
6131 60                         RTS

6132                            ;SAW: Y=VOICE

6132 B9E3BF SAW                 LDA SOUNDCONTROL-1,Y
6135 2906                       AND #%00000110
6137 0920                       ORA #%00100000
6139 99E3BF                     STA SOUNDCONTROL-1,Y
613C 60                         RTS

613D                            ;TRI: Y=VOICE

613D B9E3BF TRI                 LDA SOUNDCONTROL-1,Y
6140 2906                       AND #%00000110
6142 0910                       ORA #%00010000
6144 99E3BF                     STA SOUNDCONTROL-1,Y
6147 60                         RTS

6148                            ;MUSIC: Y=VOICE, A=LENGTH

6148 08     MUSIC               PHP
6149 78                         SEI
614A 99E6BF                     STA SOUNDLENGTH-1,Y
614D BEFF60                     LDX MUL7-1,Y
6150 B9E3BF                     LDA SOUNDCONTROL-1,Y
6153 0901                       ORA #%00000001
6155 9D04D4                     STA SID+4,X
6158 28                         PLP
6159 60                         RTS

615A                            ;ADSR: Y=VOICE, A=ATTACK*16+DECAY,
615A                            ;X=SUSTAIN*16+RELEASE

615A 48     ADSR                PHA
615B 8A                         TXA
615C BEFF60                     LDX MUL7-1,Y
615F 9D06D4                     STA SID+6,X
6162 68                         PLA
6163 9D05D4                     STA SID+5,X
6166 60                         RTS
```

```
LINE# LOC. OBJECT LABELS          LINE

  970 6167                        ;RING: Y=VOICE, CARRY SET/CLEAR=ON/OFF

  990 6167 B9E3BF RING            LDA SOUNDCONTROL-1,Y
 1000 616A 29F0                   AND #%11110000
 1010 616C 9002                   BCC RINGEXIT
 1020 616E 0904                   ORA #%00000100
 1030 6170 99E3BF RINGEXIT        STA SOUNDCONTROL-1,Y
 1040 6173 60                     RTS

 1060 6174                        ;SYNC: Y=VOICE, CARRY SET/CLEAR=ON/OFF

 1080 6174 B9E3BF SYNC            LDA SOUNDCONTROL-1,Y
 1090 6177 29F0                   AND #%11110000
 1100 6179 9002                   BCC SYNCEXIT
 1110 617B 0902                   ORA #%00000010
 1120 617D 99E3BF SYNCEXIT        STA SOUNDCONTROL-1,Y
 1130 6180 60                     RTS

 1150 6181                        ;FILTER: Y=VOICE, CARRY SET/CLEAR=ON/OFF

 1170 6181 AD9961 FILTER          LDA FILTCONTROL
 1180 6184 399261                 AND FILTTAB1-1,Y
 1190 6187 9003                   BCC FILTEXIT
 1200 6189 199561                 ORA FILTTAB2-1,Y
 1210 618C 8D9961 FILTEXIT        STA FILTCONTROL
 1220 618F 8D17D4                 STA SID+23
 1230 6192 60                     RTS

 1250 6193 FE     FILTTAB1        .BYTE   %11111110,%11111101,%11111011
 1250 6194 FD
 1250 6195 FB
 1260 6196 01     FILTTAB2        .BYTE   %00000001,%00000010,%00000100
 1260 6197 02
 1260 6198 04

 1280 6199 00     FILTCONTROL     .BYTE   0
 1290 619A 00     FILTCONTROL2    .BYTE   0

 1310 619B                        ;RESONANCE: A=RESONANCE

 1330 619B 0A     RESONANCE       ASL A
 1340 619C 0A                     ASL A
 1350 619D 0A                     ASL A
 1360 619E 0A                     ASL A
 1370 619F 48                     PHA
 1380 61A0 AD9961                 LDA FILTCONTROL
 1390 61A3 290F                   AND #%00001111
 1400 61A5 8D9961                 STA FILTCONTROL
 1410 61A8 68                     PLA
 1420 61A9 0D9961                 ORA FILTCONTROL
 1430 61AC 8D9961                 STA FILTCONTROL
 1440 61AF 8D17D4                 STA SID+23
 1450 61B2 60                     RTS
```

```
LINE# LOC. OBJECT LABELS          LINE

1470 61B3                         ;MUTE: CARRY SET/CLEAR=ON/OFF

1490 61B3 08     MUTE             PHP
1500 61B4 AD9A61                  LDA FILTCONTROL2
1510 61B7 0A                      ASL A
1520 61B8 28                      PLP
1530 61B9 6A                      ROR A
1540 61BA 8D9A61                  STA FILTCONTROL2
1550 61BD 8D18D4                  STA SID+24
1560 61C0 60                      RTS

1580 61C1                         ;VOLUME: A=VOLUME

1600 61C1 48     VOLUME           PHA
1610 61C2 AD9A61                  LDA FILTCONTROL2
1620 61C5 29F0                    AND #%11110000
1630 61C7 8D9A61                  STA FILTCONTROL2
1640 61CA 68                      PLA
1650 61CB 0D9A61                  ORA FILTCONTROL2
1660 61CE 8D9A61                  STA FILTCONTROL2
1670 61D1 8D18D4                  STA SID+24
1680 61D4 60                      RTS

1700 61D5                         ;PASS: X=RANGE; 0,1,2 OR 3.

1720 61D5 AD9A61 PASS             LDA FILTCONTROL2
1730 61D8 298F                    AND #%10001111
1740 61DA 1DE461                  ORA PASSMODES,X
1750 61DD 8D9A61                  STA FILTCONTROL2
1760 61E0 8D18D4                  STA SID+24
1770 61E3 60                      RTS

1790 61E4 10     PASSMODES        .BYTE   %00010000,%01000000
1790 61E5 40
1800 61E6 20                      .BYTE   %00100000,%01010000
1800 61E7 50

1820 61E8                         ;CUTOFF: A,X=FREQUENCY

1840 61E8 48     CUTOFF           PHA
1850 61E9 8A                      TXA
1860 61EA 8D0662                  STA CUTOFFTEMP
1870 61ED 2907                    AND #%00000111
1880 61EF 8D15D4                  STA SID+21
1890 61F2 68                      PLA
1900 61F3 4A                      LSR A
1910 61F4 6E0662                  ROR CUTOFFTEMP
1920 61F7 4A                      LSR A
1930 61F8 6E0662                  ROR CUTOFFTEMP
1940 61FB 4A                      LSR A
1950 61FC 6E0662                  ROR CUTOFFTEMP
1960 61FF AD0662                  LDA CUTOFFTEMP
1970 6202 8D16D4                  STA SID+22
1980 6205 60                      RTS

2000 6206 00     CUTOFFTEMP       .BYTE   0
```

```
=====================================================================
SOUND ROUTINES, PAGE #5
=====================================================================

LINE# LOC. OBJECT LABELS              LINE

 2020 6207                            ;SIDCLR

 2040 6207 A203   SIDCLR              LDX #3
 2050 6209 A900                       LDA #0
 2060 620B 8D9961                     STA FILTCONTROL
 2070 620E 8D9A61                     STA FILTCONTROL2
 2080 6211 9DE3BF SIDLOOP1            STA SOUNDCONTROL-1,X
 2090 6214 CA                         DEX
 2100 6215 D0FA                       BNE SIDLOOP1
 2110 6217 9D00D4 SIDCLRLOOP          STA SID,X
 2120 621A E8                         INX
 2130 621B E019                       CPX #25
 2140 621D D0F8                       BNE SIDCLRLOOP
 2150 621F 60                         RTS

 2170 6220                            ;ENV: RESULT IN A

 2190 6220 AD1CD4 ENV                 LDA SID+28
 2200 6223 60                         RTS

 2220 6224                            ;OSC: RESULT IN A

 2240 6224 AD1BD4 OSC                 LDA SID+27
 2250 6227 60                         RTS

SUCCESSFUL ASSEMBLY; NO ERRORS.
```

CONCORDANCE TABLE IN ALPHABETICAL ORDER:

LABEL	VALUE	DEFN.	REFERENCES
ADSR	$615A	890	
CUTOFF	$61E8	1840	
CUTOFFTEMP	$6206	2000	1860 1910 1930 1950 1960
ENV	$6220	2190	
FILTCONTROL	$6199	1280	1170 1210 1380 1400 1420 1430 2060
FILTCONTROL2	$619A	1290	1500 1540 1610 1630 1650 1660 1720 1750 2070
FILTER	$6181	1170	
FILTEXIT	$618C	1210	1190
FILTTAB1	$6193	1250	1180
FILTTAB2	$6196	1260	1200
IRQ	$6103	280	
MUL7	$6100	240	300 520 790 910
MUSIC	$614B	760	
MUTE	$61B3	1490	
NOISE	$6110	380	
OSC	$6224	2240	
PASS	$61D5	1720	
PASSMODES	$61E4	1790	1740
PULSE	$611B	460	
RESONANCE	$619B	1330	
RING	$6167	990	
RINGEXIT	$6170	1030	1010
SAW	$6132	600	
SID	$D400	200	310 330 530 550 820 920 940 1220 1440 1550 1670 1760 1880 1970 2110 2190 2240
SIDCLR	$6207	2040	
SIDCLRLOOP	$6217	2110	2140
SIDLOOP1	$6211	2080	2100
SOUNDCONTROL	$BFE4	180	380 410 470 500 600 630 680 710 800 990 1030 1080 1120 2080
SOUNDLENGTH	$BFE7	190	780
SYNC	$6174	1080	
SYNCEXIT	$617D	1120	1100
TRI	$613D	680	
VOLUME	$61C1	1600	

CONCORDANCE TABLE IN NUMERICAL ORDER:

LABEL	VALUE	DEFN.	REFERENCES							
MUL7	$6100	240	300	520	790	910				
FRQ	$6103	280								
NOISE	$6110	380								
PULSE	$611B	460								
SAW	$6132	600								
TRI	$613D	680								
MUSIC	$614B	760								
ADSR	$615A	890								
RING	$6167	990								
RINGEXIT	$6170	1030	1010							
SYNC	$6174	1080								
SYNCEXIT	$617D	1120	1100							
FILTER	$6181	1170								
FILTEXIT	$618C	1210	1190							
FILTTAB1	$6193	1250	1180							
FILTTAB2	$6196	1260	1200							
FILTCONTROL	$6199	1280	1170	1210	1380	1400	1420	1430	2060	
FILTCONTROL2	$619A	1290	1500	1540	1610	1630	1650	1660	1720	1750
			2070							
RESONANCE	$619B	1330								
MUTE	$61B3	1490								
VOLUME	$61C1	1600								
PASS	$61D5	1720								
PASSMODES	$61E4	1790	1740							
CUTOFF	$61E8	1840								
CUTOFFTEMP	$6206	2000	1860	1910	1930	1950	1960			
SIDCLR	$6207	2040								
SIDLOOP1	$6211	2080	2100							
SIDCLRLOOP	$6217	2110	2140							
ENV	$6220	2190								
OSC	$6224	2240								
SOUNDCONTROL	$BFE4	180	380	410	470	500	600	630	680	710
			800	990	1030	1080	1120	2080		
SOUNDLENGTH	$BFE7	190	780							
SID	$D400	200	310	330	530	550	820	920	940	1220
			1440	1550	1670	1760	1880	1970	2110	2190
			2240							

```
=====================================================================
EXAMPLE SUBROUTINES, PAGE #1
=====================================================================

LINE# LOC. OBJECT LABELS      LINE

  60 0000                     ;****************************************
  70 0000                     ;DEFINITIONS....
  80 0000                     ;****************************************

 100 6000                     *=$6000

 120 6000         IDEAL       =$9800          ;ENTRY POINT
 130 6000         SETBASE     =$BFE1          ;OFFSET FOR SPRITE VARIABLES
 140 6000         SPN         =$0400          ;SPRITE VARIABLES
 150 6000         COL         =$0402
 160 6000         ROW         =$0404
 170 6000         WID         =$0406
 180 6000         HGT         =$0408
 190 6000         SPN2        =$040A
 200 6000         COL2        =$040C
 210 6000         ROW2        =$040E
 220 6000         NUM         =$0410
 230 6000         ICL         =$0412
 240 6000         ATR         =$0414
 250 6000         CCOL        =$0416
 260 6000         CROW        =$0418

 280 6000                     ;KERNAL ROUTINES....

 300 6000         CHROUT      =$FFD2          ;OUTPUT CHARACTER
 310 6000         PLOT        =$FFF0          ;POSITION CURSOR

 340 6000         FREKZP      =$FB            ;TEMP. LOCATION
```

```
=========================================================================
EXAMPLE SUBROUTINES, PAGE #2
=========================================================================

LINE# LOC. OBJECT LABELS            LINE

  350 6000                          ;****************************************
  360 6000                          ;ROUTINE: PRINTSTRING
  370 6000                          ;PRINTS A STRING POINTED TO BY X(LOW),
  380 6000                          ;AND Y(HIGH) ON TEXT SCREEN,
  390 6000                          ;TERMINATED BY NULL.
  400 6000                          ;REGISTERS ALTERED: A,Y
  410 6000                          ;LOCATIONS USED: FREKZP,FREKZP+1
  420 6000                          ;****************************************
  ..
  440 6000 86FB   PRINTSTRING       STX FREKZP          ;FREKZP IS POINTER
  450 6002 84FC                     STY FREKZP+1        ;TO TEXT.
  460 6004 A000                     LDY #0
  470 6006 B1FB   PRINTSTRINGLOOP   LDA (FREKZP),Y
  480 6008 F00C                     BEQ PRINTSTRINGEXIT  ;EXIT IF DONE
  490 600A 20D2FF                   JSR CHROUT          ;PRINT CHAR
  500 600D E6FB                     INC FREKZP          ;INCREMENT TEXT POINTER
  510 600F D0F5                     BNE PRINTSTRINGLOOP
  520 6011 E6FC                     INC FREKZP+1
  530 6013 4C0660                   JMP PRINTSTRINGLOOP
  540 6016 60     PRINTSTRINGEXIT   RTS
```

```
LOC.  OBJECT LABELS          LINE

6017                         ;****************************************
6017                         ;ROUTINE TO DIVIDE TWO NUMBERS.
6017                         ;ENTRY: DIVIDEND IN DIVIDEND AND DIVIDEND+1
6017                         ;DIVISOR IN DIVISOR AND DIVISOR+1
6017                         ;EXIT: QUOTIENT IN DIVIDEND AND DIVIDEND+1
6017                         ;REMAINDER IN REMAINDER AND REMAINDER+1
6017                         ;CARRY SET IF ATTEMPT TO DIVIDE BY ZERO.
6017                         ;****************************************

6017 AD5260 DIVIDE           LDA DIVISOR
601A 0D5360                  ORA DIVISOR+1      ;DIVISION BY ZERO?
601D 38                      SEC
601E F02D                    BEQ DIVIDEEXIT     ;YES- EXIT, CARRY SET
6020 A211                    LDX #17            ;16 BITS+1
6022 A900                    LDA #0
6024 A8                      TAY                ;ZERO REMAINDER...
6025 F016                    BEQ UPDATEDIVIDE   ;JMP

6027 2E5060 DIVIDELOOP       ROL REMAINDER
602A 2E5160                  ROL REMAINDER+1
602D 38                      SEC
602E AD5060                  LDA REMAINDER
6031 ED5260                  SBC DIVISOR
6034 A8                      TAY
6035 AD5160                  LDA REMAINDER+1
6038 ED5360                  SBC DIVISOR+1
603B 9006                    BCC DECDIVCOUNT
603D 8C5060 UPDATEDIVIDE     STY REMAINDER
6040 8D5160                  STA REMAINDER+1
6043 2E4E60 DECDIVCOUNT      ROL DIVIDEND       ;SHIFT CARRY INTO DIVIDEND
6046 2E4F60                  ROL DIVIDEND+1     ;WHICH WILL BE QUOTIENT.
6049 CA                      DEX
604A D0DB                    BNE DIVIDELOOP
604C 18                      CLC
604D 60     DIVIDEEXIT       RTS

6050        DIVIDEND         .BLOCK  2
6052        REMAINDER        .BLOCK  2
6054        DIVISOR          .BLOCK  2
```

```
LOC. OBJECT LABELS          LINE

6054                        ;*****************************************
6054                        ;CONVERT NUMBER TO STRING AT BUFF.
6054                        ;ENTRY: NUMBER IN X(LOW),Y(HIGH)
6054                        ;EXIT: 5-CHARACTER STRING AT
6054                        ;BUFF, TERMINATED WITH NULL.
6054                        ;REGISTERS USED: ALL
6054                        ;LOCATIONS USED: DIVIDEND,REMAINDER,DIVISOR
6054                        ;*****************************************

6054 8E4E60 NUM2STRING      STX DIVIDEND        ;STORE NUMBER.
6057 8C4F60                 STY DIVIDEND+1
605A A90A                   LDA #10             ;BASE 10
605C 8D5260                 STA DIVISOR
605F A900                   LDA #0
6061 8D5360                 STA DIVISOR+1
6064 A204                   LDX #4              ;5 DIGITS
6066 8A     NUM2STRLOOP     TXA
6067 48                     PHA                 ;SAVE X
6068 201760                 JSR DIVIDE          ;DIVIDE TO GET NEXT DIGIT
606B 68                     PLA                 ;RESTORE X
606C AA                     TAX
606D AD5060                 LDA REMAINDER       ;REMAINDER IS NEXT DIGIT
6070 0930                   ORA #'0'            ;CONVERT TO ASCII
6072 9D8960                 STA BUFF,X          ;STORE
6075 CA                     DEX                 ;LOOP FOR 5 DIGITS
6076 10EE                   BPL NUM2STRLOOP
6078 A900                   LDA #0              ;STORE NULL ON END.
607A 8D8E60                 STA BUFF+5
607D 60                     RTS

607E                        ;*****************************************
607E                        ;PRINT NUMBER IN X(LOW),Y(HIGH) AS
607E                        ;DECIMAL ON TEXT SCREEN.
607E                        ;REGISTERS USED: ALL
607E                        ;LOCATIONS USED: DIVIDEND,REMAINDER,
607E                        ;DIVISOR,BUFF TO BUFF+5,FREKZP.
607E                        ;*****************************************

607E 205460 PRINTDECIMAL    JSR NUM2STRING      ;CONVERT TO STRING
6081 A289                   LDX #BUFF&255       ;AND PRINT
6083 A060                   LDY #BUFF/256
6085 200060                 JSR PRINTSTRING
6088 60                     RTS

608E        BUFF            .BLOCK  5
```

```
==========================================================================
EXAMPLE SUBROUTINES, PAGE #5
==========================================================================

LOC. OBJECT LABELS              LINE

608E                            ;****************************************
608E                            ;PUT STRING POINTED TO BY X(LOW),Y(HIGH)
608E                            ;INTO A SPRITE AT COL,ROW.
608E                            ;UPON ENTRY, ACC=0 FOR NORMAL CHARS,
608E                            ;1 FOR REVERSE, 4 FOR DOUBLE WIDTH,
608E                            ;AND 5 FOR REVERSE DOUBLE WIDTH.
608E                            ;REGISTERS USED: ALL
608E                            ;LOCATIONS USED: FREKZP,FREKZP+1,COL,NUM
608E                            ;****************************************

608E 86FB   STRPLOT             STX FREKZP          ;FREKZP IS POINTER TO STRING
6090 84FC                       STY FREKZP+1
6092 AEE1BF                     LDX SETBASE
6095 9D1104                     STA NUM+1,X         ;OFFSET IS TOP BYTE OF NUM.
6098 A000   STRPLOTLOOP         LDY #0
609A B1FB                       LDA (FREKZP),Y      ;GET CHAR
609C F021                       BEQ STRPLOTEXIT     ;EXIT IF DONE
609E AEE1BF                     LDX SETBASE
60A1 9D1004                     STA NUM,X           ;STORE IT IN NUM
60A4 A921                       LDA #33             ;PUT CHAR IN SPRITE
60A6 200098                     JSR IDEAL
60A9 BD1104                     LDA NUM+1,X         ;SET CARRY IF
60AC C904                       CMP #4              ;DOUBLE-WIDTH.
60AE BD0204                     LDA COL,X
60B1 6901                       ADC #1              ;ADD 1 OR 2 TO COL.
60B3 9D0204                     STA COL,X
60B6 F6FB                       INC FREKZP,X        ;INCREMENT STRING POINTER.
60B8 D0DE                       BNE STRPLOTLOOP
60BA F6FC                       INC FREKZP+1,X
60BC 4C9860                     JMP STRPLOTLOOP
60BF 60     STRPLOTEXIT         RTS
```

```
LINE# LOC. OBJECT LABELS          LINE

 1740 60C0                         ;**************************************
 1750 60C0                         ;MULTIPLY TWO NUMBERS:
 1760 60C0                         ;ENTRY: MULTIPLIER IN MULTIPLIER, MULTIPLIER
 1770 60C0                         ;MULTIPLICAND IN MULTIPLICAND, MULTIPLICAND+
 1780 60C0                         ;EXIT: PRODUCT IN MULTIPLIER, MULTIPLIER+1,
 1790 60C0                         ;MULTIPLIER+2 AND MULTIPLIER+3.
 1800 60C0                         ;**************************************

 1820 60C0 A900   MULTIPLY         LDA #0
 1830 60C2 8DF260                  STA MULTIPLIER+2  ;ZERO HIGH WORD
 1840 60C5 8DF360                  STA MULTIPLIER+3  ;OF PRODUCT.
 1850 60C8 A211                    LDX #17           ;16 BITS+1.
 1860 60CA 18                      CLC
 1870 60CB 6EF360 MULTLOOP         ROR MULTIPLIER+3
 1880 60CE 6EF260                  ROR MULTIPLIER+2
 1890 60D1 6EF160                  ROR MULTIPLIER+1
 1900 60D4 6EF060                  ROR MULTIPLIER    ;IF NEXT BIT=1...
 1910 60D7 9013                    BCC DECMULCOUNT
 1920 60D9 18                      CLC
 1930 60DA ADF460                  LDA MULTIPLICAND  ;ADD MULTIPLICAND
 1940 60DD 6DF260                  ADC MULTIPLIER+2  ;TO PRODUCT.
 1950 60E0 8DF260                  STA MULTIPLIER+2
 1960 60E3 ADF560                  LDA MULTIPLICAND+1
 1970 60E6 6DF360                  ADC MULTIPLIER+3
 1980 60E9 8DF360                  STA MULTIPLIER+3
 1990 60EC CA     DECMULCOUNT      DEX               ;CONTINUE UNTIL DONE.
 2000 60ED D0DC                    BNE MULTLOOP
 2010 60EF 60                      RTS

 2030 60F4        MULTIPLIER       .BLOCK  4
 2040 60F6        MULTIPLICAND     .BLOCK  2

 2060 60F6                         ;**************************************
 2070 60F6                         ;POSITIONS TEXT CURSOR AT
 2080 60F6                         ;COLUMN Y, ROW X.
 2090 60F6                         ;**************************************

 2110 60F6 18     CURSORPOS        CLC
 2120 60F7 20F0FF                  JSR PLOT
 2130 60FA 60                      RTS

SUCCESSFUL ASSEMBLY; NO ERRORS.
```

CONCORDANCE TABLE IN ALPHABETICAL ORDER:

LABEL	VALUE	DEFN.	REFERENCES
ATR	$0414	240	
BUFF	$6089	1400	1190 1230 1350 1360
CCOL	$0416	250	
CHROUT	$FFD2	300	490
COL	$0402	150	1650 1670
COL2	$040C	200	
CROW	$0418	260	
CURSORPOS	$60F6	2110	
DECDIVCOUNT	$6043	850	820
DECMULCOUNT	$60EC	1990	1910
DIVIDE	$6017	650	1140
DIVIDEEXIT	$604D	900	680
DIVIDELOOP	$6027	740	880
DIVIDEND	$604E	920	850 860 1050 1060
DIVISOR	$6052	940	650 660 780 810 1080 1100
FREEZP	$00FB	330	440 450 470 500 520 1520 1530 1570 1680 1700
HGT	$0408	180	
HT	$0412	230	
IDEAL	$9800	120	1620
MULTIPLICAND	$60F4	2040	1930 1960
MULTIPLIER	$60F0	2030	1830 1840 1870 1880 1890 1900 1940 1950 1970 1980
MULTIPLY	$60C0	1820	
MULTLOOP	$60CB	1870	2000
NUM	$0410	220	1550 1600 1630
NUM2STRING	$6054	1050	1340
NUM2STRLOOP	$6066	1120	1210
PLOT	$FFF0	310	2120
PRINTDECIMAL	$607E	1340	
PRINTSTRING	$6000	440	1370
PRINTSTRINGEXIT	$6016	540	480
PRINTSTRINGLOOP	$6006	470	510 530
REMAINDER	$6050	930	740 750 770 800 830 840 1170
ROW	$0404	160	
ROW2	$040E	210	
SCRBASE	$BFE1	130	1540 1590
SEN	$0400	140	
SEN2	$040A	190	
UNPLOT	$608E	1520	
UNPLOTEXIT	$60BF	1720	1580
UNPLOTLOOP	$6098	1560	1690 1710
UPDATEDIVIDE	$603D	830	720
WID	$0406	170	

CONCORDANCE TABLE IN NUMERICAL ORDER:

LABEL	VALUE	DEFN.	REFERENCES
FREKZP	$00FB	330	440 450 470 500 520 1520 1530 1570 1680 1700
SPN	$0400	140	
COL	$0402	150	1650 1670
ROW	$0404	160	
WID	$0406	170	
HGT	$0408	180	
SPN2	$040A	190	
COL2	$040C	200	
ROW2	$040E	210	
NUM	$0410	220	1550 1600 1630
ICL	$0412	230	
ATR	$0414	240	
CCOL	$0416	250	
CROW	$0418	260	
PRINTSTRING	$6000	440	1370
PRINTSTRINGLOOP	$6006	470	510 530
PRINTSTRINGEXIT	$6016	540	480
DIVIDE	$6017	650	1140
DIVIDELOOP	$6027	740	880
UPDATEDIVIDE	$603D	830	720
DECDIVCOUNT	$6043	850	820
DIVIDEEXIT	$604D	900	680
DIVIDEND	$604E	920	850 860 1050 1060
REMAINDER	$6050	930	740 750 770 800 830 840 1170
DIVISOR	$6052	940	650 660 780 810 1080 1100
NUM2STRING	$6054	1050	1340
NUM2STRLOOP	$6066	1120	1210
PRINTDECIMAL	$607E	1340	
BUFF	$6089	1400	1190 1230 1350 1360
STRPLOT	$608E	1520	
STRPLOTLOOP	$6098	1560	1690 1710
STRPLOTEXIT	$60BF	1720	1580
MULTIPLY	$60C0	1820	
MULTLOOP	$60CB	1870	2000
DECMULCOUNT	$60EC	1990	1910
MULTIPLIER	$60F0	2030	1830 1840 1870 1880 1890 1900 1940 1950 1970 1980
MULTIPLICAND	$60F4	2040	1930 1960
CURSORPOS	$60F6	2110	
IDEAL	$9800	120	1620
SETBASE	$BFE1	130	1540 1590
CHROUT	$FFD2	300	490
PLOT	$FFF0	310	2120

There are a number of very good Commodore 64 Assemblers and Monitors available so why is Machine Lightning different from the rest?

Machine Lightning is a complete machine code development system that is chiefly aimed at the perfectionist games writer. It has an extremely powerful syntax checking editor.

The Assembler has the following advanced features:

- ▶ Definable macros, as well as conditional assembly and all the standard assembler directives are supported.
- ▶ Source is tokenised to save memory and assembles at a phenomenal 30,000 lines a minute.
- ▶ Full printer support, Commodore and normal Centronics parallel.
- ▶ The disk version assembles from disk.

MACHINE LIGHTNING

The Monitor has the following features:

- ▶ Complete 6502 symbolic disassembly to screen or printer.
- ▶ Comprehensive set of monitor commands plus very powerful single stepping trace facilities.
- ▶ Break/Display points can be set in ROM or RAM.

Special Features

- ▶ More than 10K (object) of graphics/sound routines with more than 100 entry points, each tabulated. Routines cover scrolling, spinning, mirroring, enlarging and much, much more.
- ▶ An extremely powerful sprite generator program to design the characters utilised in the graphics object library.
- ▶ A full multi-tasking extension to Commodore Basic with more than 120 extended graphics commands. This will enable quick simple experimentation before writing the finished machine code program utilising the graphics library.
- ▶ Finished machine code programs can be sold with the object library with no marketing restrictions whatsoever.
- ▶ Fully comprehensive manual explains how to utilise 6502 interrupts to produce interrupt driven routines.

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