SIMONS' BASIC

114 ADDITIONAL PROGRAMMING COMMANDS

commodore COMPUTER

SIMONS' BASIC
USER GUIDE

C64108

Commodore Italiana SPA
Via Fratelli Gracchi 48,
Cinisello Balsamo, Milano, Italy.

Commodore Computer BV
Marksingel, 2e4811 NV Breda, Postbus 720,
4803aS Breda, Netherlands.

Commodore Business Machines Ltd.,
3370, Pharmacy Avenue, Agincourt,
Ontario, M1W 2K4, Canada.

Commodore A.G. Schweiz,
Aeschenvorstadt 57, 4010,
Basel, Switzerland.

Commodore Business Machines Inc.,
1200, Wilson Drive, West Chester,
PA 19380, USA.

Commodore Buromaschinen GmbH,
Lyoner Str. 38, Postfach 710126,
6000 Frankfurt, West Germany.

Commodore Business Machines Pty. Ltd.,
5, Orion Road, Lane Cove,
New South Wales 2066, Australia.

Commodore Business Machines (UK) Ltd.,
675 Ajax Avenue, Slough Trading Estate,
Slough, Berks. SL1 4BG England.

ABOUT THE AUTHOR

When David Simons was thirteen, his father gave him a COMMODORE computer for his birthday. Since that time he has developed an understanding of computers far in advance of his years. The SIMONS' BASIC program is the product of that experience. David was motivated by the desire to have his new COMMODORE 64 include as many commands as possible. He surveyed the variations to BASIC offered by other micros and even some minis. From this list he put together 114 commands that now comprise SIMONS' BASIC. It is with pride that COMMODORE markets the work of this sixteen-year-old student.

This manual was prepared on a COMMMODORE 8000 series computer system using a word processor. The files were then electronically transmitted into a phototypesetter and typeset by

THE ELECTRONIC VILLAGE LTD., London W4

without compositor intervention.

Special thanks to Gail Wellington, Steve Beats and Keith Morris who helped in the preparation of this manual.

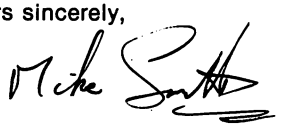
COMMENTS AND ERRATA REQUEST

TO THE READER

To the best of our knowledge, this manual is technically and typographically correct at the time of going to print. However, no matter how fine we make the sieve for catching errors, sometimes a few slip through.

If you notice any mistakes, we would be grateful if you would notify us of them. Comments, criticisms and suggestions are also earnestly solicited.

Yours sincerely,



Michael G. Smith.

Technical Author

COMMODORE BUSINESS MACHINES (UK), LTD.
675 Ajax Avenue
Trading Estate
Slough, Berkshire SL1 4BG
ENGLAND

COPYRIGHT—SOFTWARE PRODUCT

COPYRIGHT—MANUAL

DISCLAIMER

Although programs are tested by COMMODORE before release, no claim is made regarding the accuracy of this software. COMMODORE and its distributors cannot assume liability or responsibility for any loss or damage arising from the use of these programs. Programs are sold only on the basis of this understanding. Individual applications should be thoroughly tested before implementation. Should you require installation, maintenance or training, please consult your COMMODORE dealer.

TABLE OF CONTENTS

SECTION ONE—INTRODUCTION TO SIMONS' BASIC

SECTION TWO—PROGRAMMING AIDS

SECTION THREE—INPUT VALIDATION AND TEXT MANIPULATION

SECTION FOUR—EXTRA NUMERIC AIDS

SECTION FIVE—DISKETTE COMMANDS

SECTION SIX—GRAPHICS WITH SIMONS' BASIC

SECTION SEVEN—SCREEN MANIPULATION

SECTION EIGHT—SPRITE AND USER-DEFINED GRAPHICS

SECTION NINE—STRUCTURED PROGRAMMING

SECTION TEN—ERROR TRAPPING

SECTION ELEVEN—MAKING MUSIC WITH SIMONS' BASIC

SECTION TWELVE—READ FUNCTIONS

SECTION THIRTEEN—EXAMPLES OF SIMONS' BASIC PROGRAMS

APPENDIX—ERROR MESSAGES

GLOSSARY

INDEX

TABLE OF FIGURES

Figure

SECTION ONE
INTRODUCTION TO SIMONS' BASIC

1.1 INTRODUCTION

The SIMONS' BASIC cartridge has been designed to enable you to realize the full potential of your COMMODORE 64 computer. It does so by providing an additional 114 commands to complement the COMMODORE 64's standard BASIC. These extra commands fall into twelve broad areas as outlined below:

PROGRAMMING AIDS, such as KEY and TRACE, to facilitate speedier, more efficient BASIC programming.

CHARACTER STRING HANDLING commands, like INSERT and PLACE, to give you full control over string manipulation.

TEXT commands, such as CENTRE and PRINT AT, to facilitate screen formatting.

IMPROVED INPUT commands, like FETCH and INKEY, to give you full control over what is typed from the keyboard.

EXTRA ARITHMETIC OPERATORS, such as MOD and DIV, to provide a simpler method of integer division.

NUMERIC CONVERSION commands to enable you to change binary or hexadecimal numbers into the decimal equivalents.

STRUCTURED PROGRAMMING commands, such as PROC and IF..THEN..ELSE, to enable you to write more legible code.

SCREEN MANIPULATION aids, like SCRSV and COPY, to allow you to store/load screen data and/or produce a print-out of a high/low resolution screen.

GRAPHICS PLOTTING commands, such as CIRCLE and PAINT, to enable you to draw shapes on the screen.

SPRITE and USER-DEFINED GRAPHICS commands, like DESIGN, MOB SET, DETECT and CHECK to allow you to create and animate your own 'moveable object blocks' or design your own graphics characters.

MUSIC commands, such as WAVE and ENVELOPE, to enable you to create sound effects and compose and play music.

DISKETTE OPERATING commands, such as DIR, to simplify file handling.

The range of commands provided by the SIMONS' BASIC cartridge make it an essential tool for anyone interested in getting the most from his COMMODORE 64.

This manual has not been designed to teach BASIC programming on the COMMODORE 64. If you have no knowledge of BASIC programming, please refer to one of the following:

COMMODORE 64 User's Guide (supplied with your computer)

An Introduction to BASIC Parts 1 and 2, by Andrew Colin.

1.2 THE SIMONS' BASIC MANUAL

This manual is divided into thirteen sections as outlined below:

SECTION ONE—INTRODUCTION TO SIMONS' BASIC
This section outlines SIMONS' BASIC in broad terms. It also explains how to load the cartridge and how to enter a SIMONS' BASIC command. Included are the conventions used in this manual to describe each command. The compatability of SIMONS' BASIC with standard COMMODORE 64 BASIC is also discussed. Instructions on how to store, load and run SIMONS' BASIC programs are also given.

SECTION TWO—PROGRAMMING AIDS
Contained here are commands such as AUTO and TRACE to facilitate speedier, more efficient BASIC programming. Also included in this section is the KEY command which enables the COMMODORE 64's function keys to be programmed.

SECTION THREE—INPUT VALIDATION AND TEXT MANIPULATION
This section contains commands like INSERT and PLACE to improve character string handling. Also included are the commands FETCH and INKEY, both of which provide improved control over user input. In addition, screen text formatting commands, such as CENTRE and PRINT AT are also explained.

SECTION FOUR—EXTRA NUMERIC AIDS
Here three extra arithmetic operators, MOD, DIV and FRAC are described. The first two commands deal with integer division, whilst the third enables the fractional part of a number to be extracted. This section also contains a description of the commands % and $ which are used respectively for converting binary or hexadecimal numbers into decimal form and the EXOR command which performs an additional Boolean operation.

SECTION FIVE—DISKETTE COMMANDS
Two commands, DISK and DIR, are discussed here. DISK enables various disk operating commands such as formatting and file scratching to be done with one command, i.e. the disk channel is closed automatically when the task has been completed. DIR enables all, or a selected part, of a diskette directory to be displayed on the screen.

SECTION SIX—GRAPHICS WITH SIMONS' BASIC
In this section the wide range of SIMONS' BASIC graphics plotting commands are described. These commands allow you to draw shapes on the screen and paint them with any of the sixteen colours supplied by the COMMODORE 64.

SECTION SEVEN—SCREEN MANIPULATION
This section describes how to scroll an area of the screen in any direction. Also included are commands for moving an area of the screen to another location, changing the colour of screen characters and for storing and recalling screen data. Commands enabling high/low resolution screens to be printed are also described.

SECTION EIGHT—SPRITE AND USER-DEFINED GRAPHICS
Section Eight describes the SIMONS' BASIC commands concerned with the design and animation of COMMODORE 64 Sprite graphics. Also included are instructions to enable you to create your own graphics characters.

SECTION NINE—STRUCTURED PROGRAMMING
Here the various SIMONS' BASIC structured programming commands are explained.

SECTION TEN—ERROR TRAPPING
Section Ten contains commands which enable certain BASIC program errors to be trapped to prevent your programs from crashing.

SECTION ELEVEN—MAKING MUSIC WITH SIMONS' BASIC
Here the SIMONS' BASIC commands which allow you to play music on the COMMODORE 64 are described.

SECTION TWELVE—READ FUNCTIONS
This section describes those functions, such as PENX and POT, which allow you to incorporate control by a games device, such as a joystick, into a program.

SECTION THIRTEEN—EXAMPLES OF SIMONS' BASIC PROGRAMS
Section Thirteen contains listings of programs written using SIMONS' BASIC to demonstrate what may be achieved with the cartridge.

APPENDIX A—ERROR MESSAGES
A list of the error messages that you may encounter when using SIMONS' BASIC commands and their probable causes are given in this Appendix.

GLOSSARY
A list of terms that are used in this manual and their definitions are given in this section.

1.3 STARTING SIMONS' BASIC

The SIMONS' BASIC cartridge must always be inserted or removed from the COMMODORE 64 with the power OFF. The cartridge is inserted, label uppermost, into the cartridge slot at the rear of the computer. (See your COMMODORE 64 User's Guide.)

To begin using SIMONS' BASIC, simply turn the computer on with the cartridge in place. The following message is then displayed:

```
*** EXPANDED CBM V2 BASIC ***
30719 BYTES FREE
```

All the SIMONS' BASIC commands are now included in the operating system of your COMMODORE 64 and may be used at any time like any other BASIC command. Note that SIMONS' BASIC uses approximately 8K of the memory of the COMMODORE 64.

1.4 SIMONS' BASIC COMMANDS

The following is a list of commands which are added to your COMMODORE 64 operating system by the SIMONS' BASIC cartridge:

Commands for entering, debugging, listing and securing programs:

KEY, DISPLAY, AUTO, RENUMBER, PAUSE, MERGE, PAGE, OPTION, DELAY, FIND, TRACE, RETRACE, DUMP, COLD, OLD, RESET, CGOTO, DISAPA, SECURE.

Commands for text manipulation, screen formatting and input validation:

INSERT, INST, PLACE, DUP, USE, CENTRE, AT, LIN, FETCH, INKEY, ON KEY, DISABLE, RESUME.

Commands for integer division, numeric conversion and an additional Boolean operation.

MOD, DIV, FRAC, %, $, EXOR.

Commands for diskette handling:

DISK, DIR.

Commands for graphics plotting:

COLOUR, HIRES, MULTI, NRM, HICOL, LOW COL, PLOT, LINE, REC, CIRCLE, ARC, ANGL, BLOCK, PAINT, NRM, DRAW, ROT, CHAR, TEXT, TEST, CSET.

Commands for storing, printing and manipulating screen data:

LEFT/RIGHT/UP/DOWN scrolling, BCKGNDS, FLASH, OFF, BFLASH, FCHR, FCOL, FILL, MOVE, INV, SCRSV, SCRLD, COPY, HRDCPY.

Commands for generating/animating Sprites and creating your own characters:

DESIGN, @, CMOB, MOB SET, MMOB, RLOCMOB, DETECT, CHECK, MOB OFF, MEM.

Structured programming commands:

IF..THEN..ELSE, REPEAT..UNTIL, LOOP..EXIT IF..END LOOP, PROC, CALL, EXEC, END PROC, RCOMP, LOCAL, GLOBAL, NO ERROR, ON ERROR, OUT.

Commands for music synthesis:

WAVE, ENVELOPE, MUSIC, VOL, PLAY, SOUND.

Functions to use games devices with your programs:

PENX, PENY, POT, JOY.

1.5 ENTERING COMMANDS

All SIMONS' BASIC commands are entered in the same way as those in standard Commodore BASIC. Most SIMONS' BASIC commands can be used in direct mode or as part of a program. Any exceptions to this rule are indicated in the introduction to each section of the manual.

1.6 CONVENTIONS

The format of each SIMONS' BASIC command in this manual is presented using the following method of notation:

1. Brackets and items written in capital letters must be typed exactly as shown.

2. Items printed in lower case indicate a user-supplied or variable entry, e.g. coordinates or a plotting colour.

3. Other symbols, such as quotation marks and commas, must be typed exactly as shown.

4. Pressing the RETURN key is indicated by <RETURN>.

5. Keys other than alphabetic and numeric characters are indicated in the listing by the name on the key surface enclosed in < >, e.g. <CLR/HOME>. These appear on the screen as reversed characters. If two keys are enclosed, e.g. <CTRL RVS ON>, you must hold down the first key before pressing the second key.

6. With the exception of the FIND command (see Section 2.10), all SIMONS' BASIC keywords must be separated from the first parameter of the command with a space.

SECTION TWO
PROGRAMMING AIDS

2.1 INTRODUCTION

SIMONS' BASIC provides several commands which are useful when entering, debugging and listing your BASIC programs whether they include SIMONS' BASIC commands or not.

The KEY command enables the COMMODORE 64's function keys to be programmed. DISPLAY lists the values that have been assigned to these keys. The AUTO and RENUMBER commands create automatic program line numbering. MERGE combines a stored BASIC program with the program currently in the COMMODORE 64's memory.

The PAGE command permits you to specify how many screen lines you wish to use when listing programs on the screen. OPTION highlights all SIMONS' BASIC commands in a program listing. The DELAY command allows you to control the rate of scroll of program listings on the screen.

The TRACE and RETRACE commands display the numbers of program lines as they are executed. The DUMP command lists the values of all non-array variables. FIND locates all occurrences of a particular string of characters.

The PAUSE command is used to set a time delay in your program. CGOTO branches to a calculated line number. RESET instructs the COMMODORE 64 to read data from a defined program line. The SECURE and DISAPA commands 'blank' specified program lines to prevent unauthorised persons from examining your code. COLD returns the COMMODORE 64 to the SIMONS' BASIC start-up screen. The OLD command allows you to recover a program that has been NEWed.

Note that all the commands in this section can be used in direct mode or as part of a program.

2.2 ASSIGNING COMMANDS TO THE FUNCTION KEYS

2.2.1 KEY

FORMAT: KEY number,"code"

PURPOSE: To assign a command to a function key.

KEY enables you to assign your own commands to the COMMODORE 64 function keys and then change these commands if you wish. The number in the command format indicates the function key you wish to use from 1 to 16. The second parameter is the code you wish to assign to this key. A maximum of fifteen characters may be assigned to each key. Pressing the keys normally, you obtain functions F1, F3, F5 and F7. Holding down the SHIFT key and pressing these same keys, you get functions F2, F4, F6 and F8. By holding down the Commodore logo key and pressing the keys, you obtain functions F9, F10, F11 and F12. If you hold down the SHIFT key and the Commodore logo key, you get functions F13, F14, F15 and F16. Note that the code you assign to each key must be enclosed in quotation marks.

EXAMPLE: To assign the command MOB SET to function key F8:

COMMAND: KEY 8,"MOB SET" <RETURN>

RESULT: The SIMONS' BASIC code MOB SET is now assigned to the F8 function key and will be displayed every time this key is pressed.

2.2.2 ADDING CARRIAGE RETURNS

To eliminate the need to press RETURN following a function key command, you may add a carriage return to the key assignment as follows

a) Assign your command to the key (see Section 2.2.1). Type the end quote marks but do not press RETURN.

b) Type + CHR$(13) and press RETURN.

Now when you press the function key, you will automatically generate a RETURN following the assigned command.

EXAMPLE: To assign the BASIC command LIST and an automatic carriage return to the F7 function key:

COMMAND: KEY 7,"LIST" + CHR$(13) <RETURN>

RESULT: You may now list a program simply by pressing the F7 key.

2.2.3 DISPLAY

FORMAT: DISPLAY

PURPOSE: To list the commands assigned to the function keys.

DISPLAY enables you to review the current function key assignments.

EXAMPLE: To list the function key assignments after the assignment examples in the previous two sections:

COMMAND: DISPLAY <RETURN>

DISPLAY:
```
KEY 1 ""
KEY 2 ""
KEY 3 ""
KEY 4 ""
KEY 5 ""
KEY 6 ""
KEY 7 "LIST" + CHR$(13)
KEY 8 "MOB SET"
KEY 9 ""
KEY 10 ""
KEY 11 ""
KEY 12 ""
KEY 13 ""
KEY 14 ""
KEY 15 ""
KEY 16 ""
```

2.3 AUTO

FORMAT: AUTO start line number, increment

PURPOSE: To automatically generate program line numbers at a specified increment.

When the AUTO command is entered, the start program line number you have defined is displayed with the cursor following it waiting entry of a line of code. Thereafter, each time you type in a line of code and press RETURN, the increment you have specified will be added to the number of the previous line. The resulting figure will be displayed as the next program line number. To terminate this function, simply press RETURN when the line number is displayed.

EXAMPLE: To generate program line numbers automatically in intervals of 5 beginning at line 10:

COMMAND: AUTO 10,5 <RETURN>

DISPLAY: 10

TYPE: GET A$ <RETURN>

DISPLAY: 10 GET A$
15

TYPE: IF A$ = "" THEN 10 <RETURN>

DISPLAY: 10 GET A$
15 IF A$ = "" THEN 10
20

RESULT: Each time you enter a line of code and press RETURN, a line number 5 larger than the previous number is displayed.

EXAMPLE: To terminate automatic program line numbering in the program listed above:

TYPE: <RETURN>

COMMAND: LIST <RETURN>

DISPLAY: 10 GET A$
15 IF A$ = "" THEN 10
READY

RESULT: Automatic numbering is terminated.

2.4 RENUMBER

FORMAT: RENUMBER start line number,increment

PURPOSE: To automatically renumber all program lines.

RENUMBER automatically changes the numbers of all program lines. The program now begins at the start line number you have specified and all subsequent line numbers are displayed at the selected increment. This command is particularly useful if you need space in a program to insert more code.

NOTE

The RENUMBER command does not renumber GOTOs or GOSUBs. However, SIMONS' BASIC obviates the need for these instructions by replacing them with structured programming commands. See Section 9.

EXAMPLE: To renumber all the program lines of the following program:

ENTRY:
```
1 PRINT"<SHIFT CLR/HOME>"
2 FOR X = 1 TO 20
3 Z = RND(1) * 255
4 POKE 53280,Z
5 FOR Y = 1 TO 250: NEXTY,X
```

COMMAND: RENUMBER 100,10 <RETURN>

TYPE: LIST <RETURN>

DISPLAY:
```
100 PRINT"<SHIFT CLR/HOME>"
110 FOR X = 1 TO 20
120 Z = RND(1) * 255
130 POKE 53280,Z
140 FOR Y = 1 TO 250: NEXTY,X
```

2.5 PAUSE

FORMAT: PAUSE "message",number of seconds

or: PAUSE number of seconds

PURPOSE: To stop program execution for a specific interval.

PAUSE causes a program to wait before continuing to execute. The interval is a pre-specified length of time measured in seconds. Note that fractions of a second CANNOT be used. The PAUSE command can be used in two ways, either with or without a message.

If a message, enclosed in quotation marks, is included in the PAUSE command, the message is displayed for the specified period of time. Pressing the RETURN key interrupts the pause and continues the program execution.

If no message is included after the PAUSE, the program simply waits until the specified delay has elapsed.

EXAMPLE: To cause a program delay of 10 seconds:

ENTRY: 100 PAUSE 10

RESULT: When line 100 of the program is reached, a delay of 10 seconds occurs.

EXAMPLE: To display a message and wait for 1 minute:

ENTRY:
```
100 PAUSE "PRESS RETURN TO CONTINUE",60
```

RESULT: When this line is executed, PRESS RETURN TO CONTINUE is displayed and the program does not go on for one minute or until the RETURN key is pressed.

2.6 CGOTO

FORMAT: CGOTO expression

or: CGOTO operand operator variable

PURPOSE: To compute the line number to which the program should branch.

The CGOTO command allows you to branch to a variable line number determined by the result of a computation.

EXAMPLE: To branch to five different line numbers specified by a loop variable:

ENTRY:
```
10 REM"*** EXAMPLE OF CGOTO ***
20 FOR I = 1 TO 5
30 CGOTO I * 10 + 40
40 END
50 PRINT"I = 1":NEXT
60 PRINT"I = 2":NEXT
70 PRINT"I = 3":NEXT
80 PRINT"I = 4":NEXT
90 PRINT"I = 5":NEXT
```

RESULT: For each value of I, the line number is calculated and that line executed.

2.7 RESET

FORMAT: RESET line number

PURPOSE: To move data pointers to a specific line of data.

In standard BASIC, data is always read sequentially, i.e. the first item of data is used by the first READ statement, the second item by the next etc. RESET enables you to indicate the program line within a block of data from which reading is to begin i.e. you need not begin at the first item of data in the program or you may skip over some items to a specific point.

EXAMPLE: To select specific data depending on user input:

ENTRY:
```
10 REM"*** EXAMPLE OF RESET ***
20 PRINT"<SHIFT CLR/HOME>"
30 PRINT "WHICH CATEGORY?":PRINT:PRINT
40 PRINT"1) DOGS","2) CATS","3) BIRDS","4) FISH"
50 INPUT A: IF A < 0 OR A > 5 THEN PRINT"<SHIFT CURSOR
   UP>": GOTO 50
60 IF A = 1 THEN RESET 100
70 IF A = 2 THEN RESET 110
80 IF A = 3 THEN RESET 120
90 IF A = 4 THEN RESET 130
95 FOR I = 1 TO 5
97 READ A$:PRINT A$:NEXT I
99 PAUSE 10:GOTO 20
100 DATA ALSATIAN,CORGI,TERRIER,LABRADOR,SPANIEL
110 DATA PERSIAN,TABBY,ALLEY,SIAMESE,BURMESE
120 DATA SPARROW,STARLING,BUDGIE,CANARY,PIGEON
130 DATA TROUT,SALMON,CHUBB,BASS,ROACH
```

TYPE: RUN <RETURN>

ENTER: 3 <RETURN>

DISPLAY:
```
SPARROW
STARLING
BUDGIE
CANARY
PIGEON
```

RESULT: The program reads five items of data beginning at the line number relating to the user input value.

ACTION: Hold down the RUN/STOP key and press the RESTORE key.

RESULT: The program stops.

2.8 MERGE

FORMAT: MERGE "program name",device number

PURPOSE: To load a previously saved program and incorporate it into the program currently in the COMMODORE 64's memory.

The device number refers to the number of the peripheral on which the program to be MERGEd is stored. This number is 1 for a cassette unit and 8 for a disk unit. If no device number is specified, 1, i.e. cassette is assumed. The program name is specified in the same way as with the BASIC command LOAD.

CAUTION
THE MERGED PROGRAM WILL FOLLOW THE PROGRAM CURRENTLY IN MEMORY, i.e. THE MERGED PROGRAM LINES WILL BE APPENDED RATHER THAN INTERSPERSED. USE THE RENUMBER COMMAND (see Section 2.4) TO RENUMBER THE MERGED PROGRAM BEFORE EXECUTION.

EXAMPLE: To MERGE the cassette program named "SIMONS' BASIC1" with the program currently in memory:

ACTION: Write a small program and save it on cassette under the name "SIMONS' BASIC1".

COMMAND: Type NEW <RETURN>

ACTION: Write another small program.

COMMAND: MERGE "SIMONS' BASIC1",1 <RETURN>

DISPLAY: PRESS PLAY ON TAPE

ACTION: Press the PLAY button on the cassette unit.

DISPLAY: LOADING SIMONS' BASIC1
READY

RESULT: The two programs are now merged.

2.9 PROGRAM LISTING AIDS

2.9.1 PAGE

FORMAT: PAGE n

PURPOSE: To divide a program listing into 'pages' of n lines.

PAGE permits you to specify the number of screen lines you wish to use when listing a program. When the command is executed, a LIST will display the first line number of the program. Each section of the listing can then be displayed by pressing the RETURN key. A parameter of zero will terminate the paging enabling the program to be listed normally. Note that the parameter in this command refers to the number of screen lines and not to program lines which may occupy more than one screen line. If a program line overflows the screen limits you have defined, that entire line will appear on the next screen.

EXAMPLE: To list a program using only 5 screen lines:

ACTION: Load or create a program containing more than ten lines of code.

COMMAND: PAGE 5 <RETURN>

TYPE: LIST <RETURN>

RESULT: The first program line number is displayed.

COMMAND: Press the RETURN key.

RESULT: The first 5 lines of your program are displayed.

COMMAND: Press the RETURN key.

RESULT: The second 5 lines of your program are displayed.

COMMAND: PAGE 0 <RETURN>

TYPE: LIST <RETURN>

RESULT: Your program lists normally.

2.9.2 OPTION

FORMAT: OPTION n

PURPOSE: To highlight all SIMONS' BASIC commands when a program is listed.

The OPTION command with a parameter of 10 causes all SIMONS' BASIC commands to be highlighted in reverse-field when the program is listed either on the screen or on the printer. A parameter other than 10 (between 0 and 255) turns off the highlighting.

CAUTION

LISTINGS PRINTED AFTER THE OPTION COMMAND HAS BEEN USED WILL CAUSE YOUR PRINTER RIBBON TO WEAR OUT VERY QUICKLY. IT IS THEREFORE RECOMMENDED THAT LISTINGS OF THIS SORT ARE NOT PRINTED FREQUENTLY.

Note that some of the commands used in the example program below have not yet been covered. They are included merely to illustrate the use of the OPTION command.

EXAMPLE: To highlight all SIMONS' BASIC commands in the following program:

```
10 HIRES 0,1
20 CIRCLE 160,120,0,28,100
30 REC 160,120,160,120,0
40 PAUSE 10
50 CSET 0 : END
```

COMMAND: OPTION 10 <RETURN>

TYPE: LIST <RETURN>

RESULT: The SIMONS' BASIC commands are highlighted in reverse field in the screen listing.

COMMAND: OPTION 0 <RETURN>

TYPE: LIST <RETURN>

RESULT: The program lists normally.

NOTE

When listing to the printer enter all the commands on one line, e.g.:

OPEN 4,4:CMD4:LIST:PRINT#4:CLOSE4<RETURN>

Before listing any subsequent program, switch the printer off and then back on.

2.9.3 DELAY

FORMAT: DELAY n

PURPOSE: To vary the rate of scrolling of a program listing.

When the SHIFT key is held down during a program listing, the rate of screen scroll slows down. The DELAY command varies the speed of this slowed listing. The parameter following the command determines the duration of the delay. This number must be in the range 1 to 255. A larger value in the command causes a proportionately slower program listing scroll rate.

EXAMPLE: To list a program at the slowest speed available:

COMMAND: DELAY 255 <RETURN>

TYPE: LIST <RETURN>

ACTION: Hold down the SHIFT key.

RESULT: The program listing is displayed character by character.

COMMAND: Release the SHIFT key.

RESULT: The program lists normally.

NOTE
When listing any BASIC program on the COMMODORE 64, the CTRL key slows down the rate of screen scroll until the key is released.

2.10 FIND

FORMAT: FINDcode

or: FINDcharacter string

PURPOSE: To search a BASIC program for a given code or character string and display the numbers of the program lines where it appears.

FIND is used to locate specific code or character string occurrences in a BASIC program. The command displays all line numbers that contain the string or code. Note that any spaces between FIND and the specified characters or between the final character and RETURN are considered part of the character string for which the search is being made. Therefore program keywords must be entered WITHOUT a preceding space.

EXAMPLE: To find the character string ABCD in the following program:

```
10 REM FIND ABCD
20 REM PRINT "ABCD" VERTICALLY
30 PRINT "ABCD VERTICALLY"
40 A$ = "ABCD"
50 FOR C = 1 TO LEN (A$)
60 PRINT MID$(A$,C,1):NEXT
70 REM ABCD DONE
```

COMMAND: FIND"ABCD" <RETURN>

DISPLAY: 20 40

RESULT: Every program line number containing the character string ABCD enclosed within quotation marks is displayed. Note that line numbers 10 and 70 are not displayed because ABCD is not within quotation marks in those lines.

2.11 PROGRAM DEBUGGING AIDS

2.11.1 TRACE

FORMAT: TRACE n

PURPOSE: To display the number of the program line being executed.

The TRACE command is entered before a program is run. If a value of 10 is used as the command parameter, when you execute the program, a "window" appears in the top right corner of the screen. As the program lines are executed, the numbers are displayed in the window. A maximum of six numbers are shown at any one time. The format is: # (line number). The lines in the window scroll automatically so that the last but one program line number executed appears at the bottom of the window. The Commodore logo key, if held down, enables you to step through the program line by line. A parameter of 0 will turn TRACE off. Note that the TRACE command CANNOT be used on a high-resolution screen or if the MEM command (see Section 8.3.2) has been used.

CAUTION
THE TRACE WINDOW OVERWRITES ANYTHING DISPLAYED IN ITS POSITION ON THE SCREEN. THEREFORE, TAKE CARE THAT ANY TEXT YOU WISH TO OBSERVE IS PRINTED OUTSIDE THIS AREA.

EXAMPLE: To display the program line numbers one at a time when the following program is RUN:

```
10 PRINT "<SHIFT CLR/HOME>"
20 FOR X = 65 TO 96
30 PRINT "<CLR/HOME>";CHR$(X)
40 FOR Z=1 TO 250: NEXT Z
50 NEXT X
60 GOTO 20
```

COMMAND: TRACE 10 <RETURN>

TYPE: RUN <RETURN>

RESULT: Each line of the program, as it is executed, appears in the window.

TYPE: TRACE 0 RUN < RETURN >

RESULT: The window disappears and the program executes normally.

2.11.2 RETRACE

FORMAT: RETRACE

PURPOSE: To resume TRACING after editing a program.

When using the TRACE command, if you stop program execution and clear the screen, the TRACE window disappears. The RETRACE command turns TRACE back on and displays the last set of line numbers that were executed before the program was stopped. When the program is re-run, the normal TRACE display appears. Execution does not continue from where the program was stopped but from its start. Note that RETRACE cannot be used if the TRACE command has been turned off.

EXAMPLE: Using the program from the previous section, to stop the program execution, clear the screen and re-run with TRACE:

COMMAND: TRACE 10 <RETURN>

TYPE: RUN <RETURN>

RESULT: Each line of the program as it is executed appears in the TRACE window.

ACTION: Press the RUN/STOP key.

TYPE: LIST <RETURN>

ENTER: 30 PRINT "<CLR/HOME>";CHR$(X),X <RETURN>

ACTION: Hold down the SHIFT key and press the CLR/HOME key.

RESULT: The screen clears.

COMMAND: RETRACE <RETURN>

RESULT: The window at the top right of the screen re-appears and displays the line numbers that were showing when the program was stopped.

ACTION: Move the cursor below the window.

TYPE: RUN <RETURN>

RESULT: The line numbers are again displayed in the TRACE window as the program runs.

2.12 DUMP

FORMAT: DUMP

PURPOSE: To display the values of all non-array variables.

The DUMP command display the values of all variables except those contained in arrays. The values shown are those contained in the variables when the program was stopped either by pressing the RUN/STOP key or by reaching a program terminator. The variables are listed in the order in which they were defined in the program and are displayed in the format:

variable name = value

NOTE

If your program contains more than 25 variables, to prevent the list from scrolling off the screen, hold down the CTRL key. To view the remainder of the list release the key.

EXAMPLE: To display the variables from the following program:

```
10 A$ = "RANDOM COLOURS"
20 PRINT "<SHIFT CLR/HOME>",A$
30 X = INT(RND(8) * 15)
60 POKE 53281,X
70 FOR C = 1 TO 100:NEXT C
80 GOTO 30
```

TYPE: RUN <RETURN>

ACTION: After the screen has changed colour a few times, hold down the RUN/STOP key and press the RESTORE key.

COMMAND: DUMP <RETURN>

DISPLAY:
```
A$ = "RANDOM COLOURS"
X = 9
C = 80
```

(Note that, because the values of X and C are generated randomly, the numbers displayed for these two variables will depend on when the program is stopped.)

2.13 COLD

FORMAT: COLD

PURPOSE: To reset the COMMODORE 64 to the start of SIMONS' BASIC:

COLD will clear any program held in the memory of the COMMODORE 64 and display the screen that appeared when you switched on the computer with the SIMONS' BASIC cartridge in place.

WARNING
ANY PROGRAM THAT IS IN THE COMPUTER'S MEMORY WHEN THE COLD COMMAND IS USED IS CLEARED. IF YOU WISH, YOU MAY RECALL IT BY USING THE OLD COMMAND (See Section 2.15). IF ANY PART OF A NEW PROGRAM HAS BEEN ENTERED THERE IS NO WAY TO RESTORE THE PREVIOUS PROGRAM.

EXAMPLE: To reset the COMMODORE 64 to the start of SIMONS' BASIC.

COMMAND: COLD <RETURN>

RESULT: The initial SIMONS' BASIC screen is displayed.

2.14 PROGRAM SECURITY AIDS

2.14.1 INTRODUCTION

SIMONS' BASIC provides two commands which can be used to hide specified lines of program code in order to prevent unauthorised persons from examining them. The DISAPA command indicates which lines of code you wish to hide. The SECURE command blanks the code in these lines. These commands are useful for hiding passwords, serial numbers, etc.

WARNING
THERE IS NO WAY TO REVERSE THESE COMMANDS OTHER THAN RE-TYPING THE HIDDEN LINES. THEREFORE, BEFORE THEY ARE USED, IT IS WISE TO STORE AN UN-SECURED COPY OF THE PROGRAM FOR YOUR OWN USE.

2.14.2 DISAPA

FORMAT: DISAPA:

PURPOSE: To indicate that the code in a program line is to be hidden.

The DISAPA command is used as the first command on a program line and specifies that the code in this line is to be hidden. The SECURE command (see the following section) is then used to hide the code. The DISAPA command automatically places three colons (:) before the code in each line in which it appears.

Note that it is necessary to allow space on the line for these characters, i.e. the maximum length of a line to be hidden (excluding DISAPA and colons) is 30 characters.

EXAMPLE: To indicate that the code in lines 10 to 40 of the following program is to be hidden:

```
10 PRINT "HELLO"
20 PRINT SA
30 SA=SA+1
40 GOTO 10
```

ENTER:

```
10 DISAPA: PRINT "HELLO"
40 DISAPA: GOTO 10
```

COMMAND: LIST <RETURN>

DISPLAY:

```
10 DISAPA ::::: PRINT "HELLO"
20 PRINT SA
30 SA = SA + 1
40 DISAPA ::::: GOTO 10
```

RESULT: When you use the SECURE command, (see the following section) the program lines containing DISAPA will be hidden.

2.14.3 SECURE

FORMAT: SECURE 0

PURPOSE: To hide all program lines beginning with the DISAPA command.

The SECURE command prevents listing of the code in all program lines containing DISAPA (see the previous section) as the first command on that line. The code will execute as normal.

EXAMPLE: To hide lines 10 and 40 in the example program from the previous section:

COMMAND: SECURE 0 <RETURN>

TYPE: LIST <RETURN>

DISPLAY:
```
10
20 PRINT SA
30 SA = SA + 1
40
```

RESULT: When the program is listed, lines 10 and 40 appear to contain no code though the line numbers are displayed and the program runs normally.

2.15 OLD

FORMAT: OLD

PURPOSE: To reverse the NEW command.

OLD enables a program that has apparently been cleared from memory with the NEW command to be recalled and executed again. The command requires no parameters. (In more technical terms the OLD command resets the zero-page pointers to the start and end of BASIC.)

EXAMPLE: To NEW the following program and then recall it:

```
10 REM OLD COMMAND
20 A$ = "COMMODORE 64"
30 FOR C = 1 TO LEN(A$)
40 PRINT"<CLR/HOME>        ",LEFT$(A$,C)
50 FOR X = 1 TO 100 :NEXT X,C
```

COMMAND: NEW <RETURN>

TYPE: LIST <RETURN>

DISPLAY: READY

COMMAND: OLD <RETURN>

TYPE: LIST <RETURN>

DISPLAY:
```
10 REM OLD COMMAND
20 A$ = "COMMODORE 64"
30 FOR C = 1 TO LEN(A$)
40 PRINT"<CLR/HOME>        ",LEFT$(A$,C)
50 FOR X = 1 TO 100 :NEXT X,C
```

SECTION THREE
INPUT VALIDATION AND TEXT MANIPULATION

3.1 INTRODUCTION

Section Three contains those SIMONS' BASIC commands concerned with character string handling, screen formatting and input validation.

The INSERT command enables you to create a larger character string by inserting one string into another. INST enables one character string to be overwritten, from a specified position within it, by another string. The PLACE command allows you to determine the position of a group of characters within a string. DUP permits you to produce a larger character string by duplicating a smaller one a defined number of times.

The LIN command returns the number of the row on which the cursor is positioned. CENTRE allows you to centre text on a screen line. The PRINT AT command permits you to specify where text is to be printed on the screen. USE permits you to align columns of numeric data.

The FETCH command enables you to set parameters for user input. INKEY allows you to check which function key has been pressed. The ON KEY command causes a program to branch to a specific point depending on what has been typed. DISABLE terminates this command while RESUME causes it to be re-enabled.

Used in conjunction with the standard COMMODORE 64 BASIC character string commands, these features provide you with full manipulative control over text strings.

Note that the commands in this section may be used in direct mode or as part of a program.

3.2 CHARACTER STRING HANDLING

3.2.1 INSERT

FORMAT: INSERT ("sub string","main string",p)

PURPOSE: To insert one character string into another.

INSERT allows a group of characters to be placed in the midst of a character string thereby creating a longer string. The parameter p indicates the position in the main string AFTER which the sub-string is inserted. The sub-string and main-string can be any expressions enclosed within quotation marks or string variables, i.e. "aaaaa" or a$. The maximum length of the new string is 255 characters.

The INSERT command may also be used to compare two character strings using 'true/false' logic, i.e. compared in a statement of logic where a value of -1 is returned if the statement is true and 0 if it is false.

Two possible errors can be generated if this command is used incorrectly. They are:

? INSERT PARAMETER TOO LARGE

This occurs when the position specified as the insert point within the main string is a value larger than the string length.

? CREATED STRING TOO LONG

This error message is displayed if the string you have created with the INSERT command is greater than 255 characters, i.e. greater than BASIC can support.

EXAMPLE: To insert the word "BYE" into the character string "GOOD HE SAID":

ENTER: 100 PRINT INSERT ("BYE ","GOOD HE SAID",5)

TYPE: RUN <RETURN>

DISPLAY: GOOD BYE HE SAID

RESULT: The sub string "BYE" has been inserted within the main string "GOOD HE SAID" beginning at the sixth character position.

EXAMPLE: To create a longer string variable:

ENTER:
```
100 B$= "BYE "
105 A$=INSERT (B$,"GOOD HE SAID",5)
110 PRINT A$
```

TYPE: RUN <RETURN>

DISPLAY: GOOD BYE HE SAID

TYPE: DUMP <RETURN>

DISPLAY:
```
B$ = "BYE "
A$="GOOD BYE HE SAID"
```

EXAMPLE: To compare two character strings:

ENTER:
```
100 A = (INSERT("BYE ","GOOD HE SAID",5) = " GOOD BYE
      HE SAID")
110 PRINT A
```

TYPE: RUN <RETURN>

DISPLAY: -1

RESULT: Because the two strings are the same, i.e. the condition is true, a value of -1 is returned. If the condition had been false, a value of zero would have been returned.

3.2.2 INST

FORMAT: INST ("sub string","main string",p)

PURPOSE: To overwrite a string beginning at a specified position.

INST replaces a string of characters with another string overwriting the main string starting from the position specified. The sub string or main string can be any expression provided they are character string variables i.e. "aaaaa" or XX$. The value of p indicates the position AFTER which the sub string overwrites the main string.

There is one possible error message that could occur with this command:

? CREATED STRING TOO LONG

This happens if the new string you have created is longer than 255 characters.

EXAMPLE: To replace the word "GOOD" with "BETTER" in the sentence "HE WAS GOOD":

ENTER:
```
 5 A$ = "HE WAS GOOD"
10 A$ = INST("BETTER",A$,7)
20 PRINT A$
```

TYPE: RUN <RETURN>

DISPLAY: HE WAS BETTER

COMMAND: DUMP <RETURN>

DISPLAY: A$ = "HE WAS BETTER"

3.2.3 PLACE

FORMAT: PRINT PLACE ("sub string","main string")

PURPOSE: To determine the position of a sub string within a main string.

PLACE searches for a specified group of characters (sub string) within a character string. If the group is found, the position of the first character of the group is returned. If a match is not found a value of zero is returned. The length of the sub string must always be shorter than that of the main string being searched. This command may also be used to compare two numeric variables.

EXAMPLE: To determine the position of the sub string "BETTER" within the main string "HE WAS BETTER":

ENTER:
```
10 A$ = INST("BETTER","HE WAS GOOD",7)
20 PRINT PLACE ("BETTER",A$)
```

TYPE: RUN <RETURN>

DISPLAY: 8

EXAMPLE: A simple English Language test:

ENTER:
```
10 PRINT"ENTER THE POSITION OF THE FIRST CHARACTER
   OF THE ADVERB";
15 PRINT"IN THE FOLLOWING SENTENCE:":PRINT
20 PAUSE 1
30 A$ = "HE CALLED OUT FOR HER LOUDLY"
40 B = PLACE("LOUDLY",A$):B$ = "LOUDLY":PRINT A$
50 INPUT A
60 IF A = B THEN 80
70 PRINT"INCORRECT":PRINT"THE CORRECT ANSWER IS" B
75 PRINT"THE ADVERB IS  ";B$:END
80 PRINT "WELL DONE":END
```

TYPE: RUN <RETURN>

DISPLAY: ENTER THE POSITION OF THE FIRST CHARACTER OF THE ADVERB IN THE FOLLOWING SENTENCE:

HE CALLED OUT FOR HER LOUDLY

TYPE: 8 <RETURN>

DISPLAY: INCORRECT
THE CORRECT ANSWER IS 23
THE ADVERB IS LOUDLY

3.2.4 DUP

FORMAT: DUP ("string",n)

PURPOSE: To duplicate a character string n times.

DUP enables a new character string to be produced from multiples of a string. The n indicates the number of times the old string is reproduced.

Note that, if the new string you have created is longer than 255 characters, the following error message is displayed:

? CREATED STRING TOO LONG

EXAMPLE: To duplicate a character string three times and then add another string:

ENTER:
```
10 A$ = DUP ("HELLO-",3)
20 B$ = "WHAT'S GOING ON HERE?"
30 C$ = A$+B$:PRINT C$
```

TYPE: RUN <RETURN>

DISPLAY: HELLO-HELLO-HELLO-WHAT'S GOING ON HERE?

3.2.5 CENTRE

FORMAT: CENTRE "character string"

PURPOSE: To centre a character string on a screen line.

CENTRE enables text to be displayed in the middle of a screen line. You need not know the length of the text to use this command.

EXAMPLE: To centre the character string "COMMODORE 64":

COMMAND: CENTRE "COMMODORE 64" <RETURN>

DISPLAY: COMMODORE 64

3.2.6 AT

FORMAT: PRINT AT (c,r) "character string"

or: PRINT "1st.string"AT(c,r)"2nd.string"

PURPOSE: To print a character string at a specified screen location.

The AT command enables you to specify the screen location where the printing of a character string will begin. This replaces the use of cursor control characters to position the text. The parameters c and r define the column and row coordinates of the location on the screen where you wish the character string following the parameter to begin. More than one AT command may be combined in a single statement.

EXAMPLE: To position the character string "COMMODORE 64" at column 13, row 8:

COMMAND: PRINT AT(13,8)"COMMODORE 64" <RETURN>

DISPLAY: As shown in Figure 3-1.

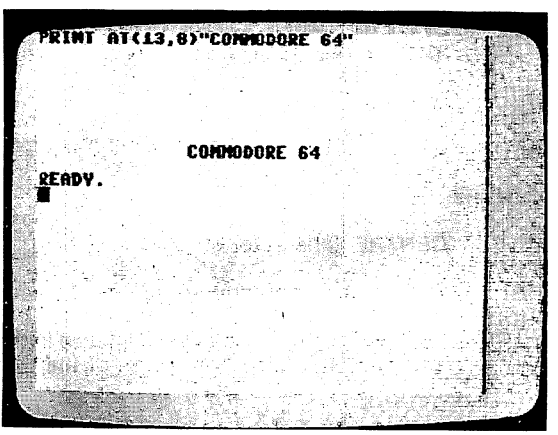


FIGURE 3-1 A SINGLE 'AT' COMMAND

EXAMPLE: To print the character string "CBM 64" starting at column 13, row 8 and the string "SIMONS' BASIC" three lines below and two characters to the right:

COMMAND: PRINT AT(13,8)"CBM 64"AT(15,11)"SIMONS' BASIC" <RETURN>

DISPLAY: As shown in Figure 3-2.

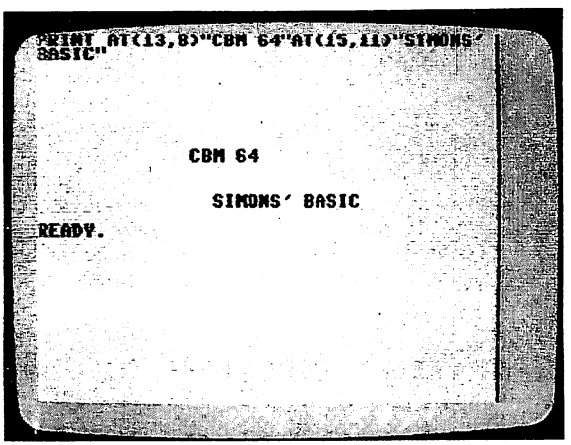


FIGURE 3-2 A COMPOUNDED 'AT' COMMAND

3.2.7 USE

FORMAT: USE "# # # .# # # # #",vs:PRINT

or: USE "# #text.# # #text",vs:PRINT

PURPOSE: To format numeric data.

The USE command allows you to format lists of numbers, i.e. to align the decimal points. The amount of hash signs (#) either side of the decimal point instructs the COMMODORE 64 to display the corresponding number of figures from the string relative to this position. If you wish, you may also insert text between the hash signs. The parameter vs is the string representation of the number you wish to USE. Note that PRINT must follow the string as the USE command does not force a carriage return.

EXAMPLE: To print a tabulated list of randomly generated prices:

ENTRY:
```
10 REM"*** EXAMPLE OF USE ***
20 A$ = STR$(RND(1) * 199)
30 USE "$###.##C",A$:PRINT
40 GET A$:IF A$ = "" THEN 40
50 GOTO 20
```

TYPE: RUN <RETURN>

ACTION: Press any key

DISPLAY: $126.45C

ACTION: Press any key

DISPLAY: $ 35.36C

RESULT: Each time you press a key, a value is displayed. The decimal points of all figures in the list appear in the same position on each screen line. Note that, as the values are generated randomly, the figures that are shown above are examples only.

3.3 INPUT VALIDATION COMMANDS

3.3.1 FETCH

FORMAT: FETCH "control character",l,designated string.

PURPOSE: To limit the type and number of characters for user input.

FETCH enables you to control what is accepted as input from the keyboard. The control character within quotation marks determines the types of characters allowed. The types of valid characters and the associated control character are shown below:

CONTROL CHARACTER	VALID CHARACTERS
CLR/HOME	Un-shifted alphabetic characters only.
CURSOR DOWN	Numeric characters only.
CURSOR RIGHT	Alphanumeric and shifted characters.

The parameter l in the FETCH command is a number which specifies the maximum amount of characters that the user may enter. The third parameter in the command specifies the string variable into which the input will be placed.

EXAMPLE: To restrict user input to a maximum of eight unshifted alphabetical characters and place this input into the string variable A$:

ENTER:
```
10 PRINT:PRINT"WHAT'S YOUR NAME?"
20 FETCH " <CLR HOME> ",8,A$
30 PRINT"HELLO "A$
```

TYPE: RUN <RETURN>

DISPLAY: WHAT'S YOUR NAME?
(cursor)

ACTION: Hold down the SHIFT key and press any letter.

RESULT: Nothing happens.

ACTION: Press a numeric key.

RESULT: Again, nothing happens.

TYPE: MIKE <RETURN>

DISPLAY: HELLO MIKE

COMMAND: DUMP <RETURN>

DISPLAY: A$ = "MIKE"

RESULT: Only a string of eight or fewer unshifted alphabetic characters is accepted as input into A$.

3.3.2 INKEY

FORMAT: INKEY

PURPOSE: To test for a function key input.

INKEY enables you to determine which function key has been pressed. INKEY represents the number of the function key which is pressed (1 through 8). This command is especially useful in menu driven programs where the functions keys can be used to select specific options or operations.

EXAMPLE: To test for function keys F1 and F2:

ENTRY:
```
  10 A = INKEY
  20 ON A GOSUB 1000,2000
  30 GOTO10
1000 PRINT "YOU PRESSED F1":RETURN
2000 PRINT "YOU PRESSED F2":RETURN
```

TYPE: RUN <RETURN>

ACTION: Press the F1 function key.

DISPLAY: YOU PRESSED F1

ACTION: Hold down the SHIFT key and press the F1 function key.

DISPLAY: YOU PRESSED F2

3.3.3 ON KEY

FORMAT: ON KEY "character(s)",:GOTO line number

PURPOSE: To branch to a specific point in a program.

The ON KEY command causes the COMMODORE 64 to scan the keyboard for input of one of the characters defined in the command. Any key not specified is ignored. On receipt of a valid character, program execution continues from the line specified by GOTO. The reserved variable ST holds the CHR$ value of the key that has been pressed. (A full list of CHR$ codes can be found in your COMMODORE 64 User Guide.) This command is especially useful in menu-driven programs.

NOTE

When an ON key command is executed the COMMODORE 64 will still scan the keyboard even after a character within the specified range has been entered. You must therefore use the DISABLE command (see the following section) to turn ON KEY off.

EXAMPLE: To define a range of valid input characters:

ENTRY:
```
10 PRINT"<SHIFT CLR/HOME>PRESS A KEY (E TO END)"
20 B$ = "DGHNVMLPOE"
30 ON KEY B$,: GOTO 50
40 GOTO 20
```

RESULT: When this section of the program is run, the program halts until one of the characters in the range defined is entered.

3.3.4 DISABLE

FORMAT: DISABLE

PURPOSE: To terminate the ON KEY command.

DISABLE causes the keyboard scan generated by the ON KEY command (see the previous section) to be turned off. This command must ALWAYS be used if the ON KEY command has been used. Failure to do so will result in 'recursive jumps', i.e. the program will always return to the line specified by ON KEY each time one of the specified characters is typed.

EXAMPLE: To disable the ON KEY command:

ENTRY:
```
10 PRINT"<SHIFT CLR/HOME>PRESS A KEY (E TO END)"
20 B$ = "DGHNVMLPOE"
30 ON KEY B$,: GOTO 50
40 GOTO 20
50 DISABLE
```

RESULT: When this section of the program is executed, the ON KEY command is turned off after a valid character is typed.

3.3.5 RESUME

FORMAT: RESUME

PURPOSE: To reinstate the previous ON KEY command.

The RESUME command causes the last ON KEY command that was defined to be turned back on. This causes the program to halt again until one of the characters in the range specified by ON KEY is typed.

EXAMPLE: Expanding the program from the previous section, to turn the ON KEY command back on:

ENTRY:
```
10 PRINT "<SHIFT CLR/HOME>PRESS A KEY (E TO END)"
20 B$ = "DGHNVMLPOE"
30 ON KEY B$,: GOTO 50
40 GOTO 30
50 DISABLE
60 A$ = CHR$(ST): X = PLACE(A$,B$)
70 ON X GOTO 80,90,100,110,120,130,140,150,160,170
80 PRINT "IT WAS D": RESUME
90 PRINT "IT WAS G": RESUME
100 PRINT "IT WAS H": RESUME
110 PRINT "IT WAS N": RESUME
120 PRINT "IT WAS V": RESUME
130 PRINT "IT WAS M": RESUME
140 PRINT "IT WAS L": RESUME
150 PRINT "IT WAS P": RESUME
160 PRINT "IT WAS O": RESUME
170 PRINT "IT WAS E": END
```

TYPE: RUN <RETURN>

TYPE: V <RETURN>

DISPLAY: IT WAS V

TYPE: X <RETURN>

RESULT: Nothing is displayed.

TYPE: E <RETURN>

DISPLAY:
```
IT WAS E
READY
```

RESULT: A character within the range defined in the ON KEY command causes that character message to be displayed. Any other character is ignored.

SECTION FOUR
EXTRA NUMERIC AIDS

4.1 INTRODUCTION

This section contains various commands to assist you when handling numeric data. The commands MOD and DIV enable integer division to be performed on positive numbers. All results are returned rounded. FRAC allows you to extract the fractional part of a number. Also included in this section are commands to convert hexadecimal or binary numbers into decimal. An addition Boolean operator, exclusive or (EXOR), completes the commands in this section.

Note that the commands in this section may be used in direct mode or as part of a program.

4.2 ADDITIONAL ARITHMETIC OPERATORS

4.2.1 MOD

FORMAT: MOD(x,y)

PURPOSE: To return the remainder when one integer is divided by another.

The MOD command displays the remainder when one integer, i.e. whole number, is divided by another integer. The MOD command can be used directly or within a program.

EXAMPLE: To divide 15 by 4 and produce the remainder:

TYPE: PRINT MOD(15,4) <RETURN>

DISPLAY: 3

4.2.2 DIV

FORMAT: DIV(x,y)

PURPOSE: To return the largest integer which, when multiplied by y is equal to or less than x.

The DIV command enables you to divide one floating-point number by another and produce the result in integer format, i.e. the fractional part of the result is ignored.

EXAMPLE: To divide 10 by 3 and produce the result in integer form:

ENTRY:
```
10 A = DIV(10,3)
20 PRINT A
```

TYPE: RUN <RETURN>

DISPLAY: 3

4.2.3 FRAC

FORMAT: FRAC(n)

PURPOSE: To return the fractional part of a number.

FRAC allows you to extract that part of a floating-point, i.e. non-integer, number that follows the decimal point up to a maximum of nine decimal places.

EXAMPLE: To divide 22 by 6 and produce the fractional part of the result:

COMMAND: PRINT FRAC(22/6) <RETURN>

DISPLAY: .666666667

EXAMPLE: To return the fractional part of π:

ENTRY:
```
10 PRINT π
20 PRINT FRAC(π)
```

TYPE: RUN <RETURN>

DISPLAY:
```
3.14159265
.141592653
```

4.3 NUMERIC CONVERSION

4.3.1 % - BINARY TO DECIMAL CONVERSION

FORMAT: PRINT %binary number

PURPOSE: To convert from binary into decimal.

The % command converts a binary number into its decimal equivalent.

If a non-binary number is used as the argument in the command, the message:

? NOT BINARY CHARACTER

is displayed.

EXAMPLE: To convert the binary number 10110101 into decimal form:

COMMAND: PRINT %10110101 <RETURN>

DISPLAY: 181

4.3.2 $ - HEXADECIMAL TO DECIMAL CONVERSION

FORMAT: PRINT $hexadecimal number

PURPOSE: To convert from hexadecimal into decimal

The $ command converts a hexadecimal number into its decimal equivalent.

If a non-hexadecimal number is used as the argument in the command, the message:

? NOT HEX CHARACTER

is displayed.

EXAMPLE: To convert the hexadecimal number EB38 into decimal form:

COMMAND: PRINT $EB38 <RETURN>

DISPLAY: 60216

4.3.3 COMBINING THE CONVERSION COMMANDS

The two commands above can be used together.

EXAMPLE: To add together a binary and hexadecimal number and return the result in decimal form:

COMMAND: PRINT %10110101 + $EB38 <RETURN>

DISPLAY: 60397

4.4. EXOR

FORMAT: EXOR(n,n1)

PURPOSE: To perform an exclusive or between two numbers

The EXOR command allows you to perform an exclusive or between two number. The command first converts both numbers into binary form. It then compares these binary numbers bit by bit. If both bits are the same, the corresponding result bit is cleared, i.e. a 0. If the bits are different, the corresponding result bit is set, i.e. a 1.

EXAMPLE: To exclusive or 87 and 45:

COMMAND: PRINT EXOR(87,45)

DISPLAY: 122

The routine used to arrive at this answer is shown below:

First Number = 87 = 01010111
Second Number = 45 = 00101101

Result = 01111010 = 122

EXAMPLE: To print characters on the screen in reverse field:

ENTRY:
```
 5 PRINT"<SHIFT CLR/HOME>"
 7 FOR C = 1 TO 10
 8 PRINT "SIMONS' BASIC":PRINT:NEXT
10 FOR X = 0 TO 999
20 A = PEEK (1024 + X)
30 IF A = 32 THEN 60
40 K = EXOR (A,128)
50 POKE 1024 + X,K
60 NEXT :GOTO 10
```

TYPE: RUN <RETURN>

RESULT: Every character on the screen is changed into reverse field and then back to normal.

SECTION FIVE
DISKETTE COMMANDS

5.1 INTRODUCTION

SIMONS' BASIC contains two simplified disk-handling commands. DISK eliminates the need to specify a logical file number, device number and secondary address when opening a channel to a disk drive unit. The command also automatically closes the channel when the operation specified has been completed. The DIR command replaces the BASIC code LOAD "$",8 allowing you to list some or all of a diskette directory with a single command.

5.2 DISK

FORMAT: DISK,"operation"

PURPOSE: To open a diskette channel and then close it when the operation is executed.

DISK replaces the following BASIC code:

OPEN logical file,device number,secondary address:PRINT # logical file

The command opens a channel to the diskette unit and then automatically closes this channel when the specified operation has been completed.

EXAMPLE: To format a new diskette heading it "TEST":

ACTION: Place a new diskette in the diskette unit.

COMMAND: DISK "N0:TEST, 01" <RETURN>

RESULT: After a few minutes, the new diskette is formatted with the header "TEST" and the drive light goes off to indicate that the diskette channel has been closed.

EXAMPLE: To scratch a program from a diskette:

ACTION: Type in the following short program and then save it under the name of "ORANGE" on diskette:

```
10 REM"*** EXAMPLE OF DISK ***
20 REM"*** DELETING A PROGRAM ***
```

COMMAND: DISK "S0:ORANGE" <RETURN>

RESULT: The program "ORANGE" is deleted from the diskette and the diskette channel is closed.

5.3 DIR

FORMAT: DIR "$

or: DIR"$:character string*

or: DIR"$:?character string

or: DIR"$:?character string*

PURPOSE: To list some or all of a diskette directory.

The DIR command replaces the BASIC code: LOAD "$",8. The command enables you to display some or all of a diskette directory. You may display only those files whose names begin with a particular character or string of characters by entering this character or character string followed by an asterisk. If you wish, you may display only those files where a specific character or character string is in a particular position within the filename by replacing the leading characters with question marks (?).

EXAMPLE: To list a complete directory:

COMMAND: DIR"$ <RETURN>

RESULT: The directory of the diskette in device number 8 is displayed.

EXAMPLE: To list only those files where the third character of the filename is "S":

COMMAND: DIR"$:??S* <RETURN>

RESULT: The display shows the directory listing of the names of files in which the third character is S.

SECTION SIX
GRAPHICS WITH SIMONS' BASIC

6.1 INTRODUCTION

This section describes the comprehensive SIMONS' BASIC graphics plotting commands. These commands enable you to plot points, draw shapes, enter text and paint on the screen in any one of sixteen colours without having to access any memory locations.

The COLOUR command sets up the colour of the screen and the border surrounding it. The HIRES command puts the COMMODORE 64 into high-resolution graphics plotting mode. In this mode, all points are plotted pixel by pixel. MULTI initializes the multi-colour mode. Here, each point plotted is two pixels wide. Both the HIRES and MULTI commands allow you to specify in which colour you wish to plot your graphics shapes. The LOW COL command changes these colours while HI COL reverts back to those plotting colours that were originally selected.

The PLOT command enables single dots to be plotted on the screen. TEST allows you to check the status of a defined screen location, i.e. whether a dot has been plotted in that position and in which colour the dot has been plotted. The REC command allows you to draw rectangles and CIRCLE enables circular shapes to be drawn. The ARC command plots a specified section of the circumference of a circular shape while the ANGL command draws its radius. Line draws a solid line.

The PAINT command fills a graphics shape with a specified colour. BLOCK displays fully shaded blocks of colour. The DRAW and ROT commands permit you to design a freehand shape and then display it at a specific size and angle of rotation.

The CSET command selects either the Upper/Lower case or Upper Case/Graphics COMMODORE 64 character set. This command also allows you to recall and display the last graphics screen that was shown. The CHAR and TEXT commands print single characters and character strings respectively on a graphics screen. NRM returns to a normal screen from a graphics screen.

The first half of Section Six discusses the configuration of the screen and the differences between high-resolution and multi-colour graphics. The sixteen COMMODORE 64 colours are listed and you are shown how to select a colour when plotting. The second half of this section describes the format and use of each graphics plotting command. The commands are listed in the order in which they might be used to write programs such as those contained in Section Thirteen of this manual.

Note that, with the exception of the COLOUR command, the commands in Section Six can only be used as part of a program.

6.2 SCREEN CONFIGURATION

For the purposes of graphics plotting, the COMMODORE 64 screen is divided into a matrix or grid. Each point on the grid is specified by its x and y coordinates much as you would indicate a point on a graph. For example, location 0,0 refers to the top left corner of the screen. The size of the grid varies according to whether you are using the high-resolution or multi-colour graphics mode. In high-resolution mode, the screen is divided into a 320 by 200 dot matrix. In multi-colour mode, this matrix is 160 by 200 dots. This means that each dot plotted in high-resolution mode is one pixel wide, i.e. the smallest addressable point on the screen. In multi-colour mode, each point plotted is two pixels wide.

6.3 COMMODORE 64 COLOURS

Whether in high-resolution or multi-colour mode, only THREE colours can be used in any one 8 by 8 pixel area of the screen. The COMMODORE 64 provides sixteen different plotting colours. These colours and their associated values are listed below:

0 Black
1 White
2 Red
3 Cyan
4 Purple
5 Green
6 Blue
7 Yellow
8 Orange
9 Brown
10 Light Red
11 Gray 1
12 Gray 2
13 Light Green
14 Light Blue
15 Gray 3

When plotting graphics, the colour you wish to use for the shape is specified in terms of its associated colour number.

6.4 PLOT TYPES

All SIMONS' BASIC graphics-plotting commands have one common feature. They each require you to specify a 'plot type'. This simply tells the COMMODORE 64 how to plot each point. The plot types for both high-resolution and multi-colour modes are listed below:

HIGH-RESOLUTION MODE

PLOT TYPE	FUNCTION PERFORMED
0	Clears a dot.
1	Plots a dot on the screen.
2	Inverses a dot, i.e. turns a dot OFF if it is ON, or ON if it is OFF.

MULTI-COLOUR MODE

PLOT TYPE	FUNCTION PERFORMED
0	Clears a dot.
1	Plots a dot in colour 1 of the MULTI/LOW COL command.
2	Plots a dot in colour 2 of the MULTI/LOW COL command.
3	Plots a dot in colour 3 of the MULTI/LOW COL command.
4	Inverses the dot colour, i.e.: A dot plotted in colour 0 changes to colour 3 A dot plotted in colour 1 changes to colour 2 A dot plotted in colour 2 changes to colour 1 A dot plotted in colour 3 changes to colour 0

6.5 GRAPHICS PLOTTING COMMANDS

6.5.1 COLOUR

FORMAT: COLOUR bo,sc

PURPOSE: To set up the screen background and border colours.

The COLOUR command allows you to specify the colour of the screen background for a low-resolution screen and the the colour of the border surrounding both low and high-resolution screens. The parameter sc refers to the screen background colour and the parameter bo to the border colour. Colours are selected by specifying their associated colour numbers (see Section 6.3).

Note that the screen background for a low-resolution screen will remain at the colour selected until the COLOUR command is executed again using a different colour value. The background colour of a screen containing high-resolution or multi-colour graphics is selected using the HIRES command (see the following section).

EXAMPLE: To specify a screen background colour of cyan and a border colour of blue:

COMMAND: COLOUR 3,6 <RETURN>

RESULT: A cyan screen is displayed surrounded by a blue border.

6.5.2 HIRES

FORMAT: HIRES pc,sb

PURPOSE: To initialize the high-resolution graphics mode and select a plotting colour and screen background colour.

The HIRES command sets the screen into high-resolution graphics mode, i.e all points are plotted pixel by pixel. The first parameter, pc, is the colour number of the plotting colour you wish to use (see Section 6.3). The second parameter, sb, specifies the background colour of each 8 by 8 pixel square through which plotting takes place. Note that there must be a space between the HIRES command and its first parameter.

EXAMPLE: To select a plot colour of black on a white background:

ENTER:
```
10 HIRES 0,1
20 GOTO 20
```

TYPE: RUN <RETURN>

RESULT: A blank white screen is displayed.

ACTION: Press the RUN/STOP key.

RESULT: The normal screen appears.

6.5.3 REC

FORMAT: REC x,y, A , B,plot type

PURPOSE: To draw a rectangle.

The REC command draws a rectangular shape on a graphics screen. The first parameters of the command (x,y) specify the coordinates of the top left corner of the rectangle. The x indicates the distance from the left edge and y from the top of the screen. The parameter A indicates the distance from the top left to the top right corner of the rectangle and B from the top left to the bottom left corner. Plot type is as described in Section 6.4.

EXAMPLE: To draw a rectangle in high-resolution graphics at the top left corner of the screen:

ENTER:
```
10 HIRES 0,1
20 REC 0,0,40,20,1
30 GOTO 30
```

TYPE: RUN <RETURN>

RESULT: A black rectangle is displayed.

ACTION: Press the RUN/STOP key.

RESULT: The normal screen is displayed.

6.5.4 MULTI

FORMAT: HIRES pc,sb: MULTI c1,c2,c3

PURPOSE: To initialize the multi-colour graphics mode and select three plotting colours.

MULTI, when used following the HIRES command, will cause all plotting to take place in multi-colour graphics mode, i.e. each point plotted will be two pixels wide. The three parameters following MULTI define the plot colours you wish to use. Each plot colour is selected by referring to its position within the MULTI command as the 'plot type' in a plotting command (see Section 6.4).

EXAMPLE: To enter the multi-colour graphics mode specifying black, red and blue as the plotting clours and then draw three rectangles:

ENTER:
```
10 HIRES 0,1: MULTI 0,2,6
30 REC 0,0,40,20,1
40 REC 20,20,40,20,2
50 REC 40,40,40,20,3
60 GOTO 60
```

TYPE: RUN <RETURN>

RESULT: One black, one red and one blue rectangle are drawn.

ACTION: Press the RUN/STOP key.

RESULT: The normal screen is displayed.

6.5.5 NRM

FORMAT: NRM

PURPOSE: To return to a low-resolution screen from a graphics screen.

The NRM command allows you to clear a high-resolution or multi-colour graphics display and return to a low-resolution screen.

EXAMPLE: Using the program from the previous section, to return to the normal screen after the graphics screen has been displayed for five seconds:

ENTER:
```
10 HIRES 0,1: MULTI 0,2,6
30 REC 0,0,40,20,1
40 REC 20,20,40,20,2
50 REC 40,40,40,20,3
60 PAUSE 5
70 NRM
```

TYPE: RUN <RETURN>

RESULT: Three rectangles are displayed for five seconds. The normal screen then appears displaying READY and a flashing cursor.

6.5.6 LOW COL

FORMAT: LOW COL c1,c2,c3

PURPOSE: To change plotting colours.

LOW COL enables you to specify a different set of graphics plotting colours from those originally selected with the HIRES or MULTI commands.

> NOTE
> Because only two colours are used in high-resolution graphics plotting (see Section 6.5.2), the third colour in the LOW COL command has no effect in hi-res. However, three numbers must be used in the LOW COL command irrespective of what graphics mode has been initialized.

EXAMPLE: To draw a black rectangle in high-resolution graphics mode and shade the lines yellow:

ENTER:
```
10 HIRES 0,1
20 LOW COL 0,7,0
30 REC 20,20,60,60,1
40 PAUSE 5
50 NRM
```

TYPE: RUN <RETURN>

RESULT: A black rectangle is drawn on a white screen. Every 8 by 8 pixel rectangle over which plotting has occurred is coloured yellow. After five seconds, the normal screen is displayed.

EXAMPLE: To draw three rectangles in multi-colour graphics mode using each of the original plot colours, then changing these colours and drawing three rectangles in each of the new colours:

ENTER:
```
10 HIRES 0,1: MULTI 2,3,6: Z = 10
 20 FOR X = 1 TO 3
 30 REC 10,Z,30,30,X
 40 Z = Z + 40: NEXT
 50 LOW COL 4,5,7: Z = 10
 60 FOR X = 1 TO 3
 70 REC 50,Z,30,30,X
 80 Z = Z + 40: NEXT
 90 PAUSE 5
100 NRM
```

TYPE: RUN <RETURN>

RESULT: Six rectangles are drawn, each a different colour. After five seconds, the normal screen is displayed.

6.5.7 HI COL

FORMAT: HI COL

PURPOSE: To revert to the originally selected plotting colours.

The HI COL command allows you to restore your original plotting colours, i.e. those originally set up with the HIRES or MULTI command, if LOW COL (see the previous section) has been used.

EXAMPLE: To draw nine rectangles in different colours:

ENTER:
```
10 HIRES 0,1: MULTI 2,3,6: Z = 10
20 FOR Y = 10 TO 50 STEP 40
30 FOR X = 1 TO 3
40 REC Y,Z,30,30,X
50 Z = Z + 40:NEXT X: Z = 10:LOW COL 4,5,7:NEXT Y
60 HI COL
70 FOR X = 1 TO 3
80 REC Y,Z,30,30,X
90 Z = Z + 40:NEXT
100 PAUSE 5
110 NRM
```

TYPE: RUN <RETURN>

RESULT: Three rectangles are displayed in the original plot colours, three in the colours assigned with the LOW COL command and three more, again using the original plot colours. After five seconds, the normal screen is displayed.

6.5.8 PLOT

FORMAT: PLOT x,y,plot type

PURPOSE: To plot one dot.

PLOT plots a single dot on a graphics screen. The parameters x and y specify the horizontal and vertical screen coordinates respectively of the point to be plotted. Plot type is as described in Section 6.4.

EXAMPLE: To plot a black dot in multi-colour graphics mode:

ENTRY:
```
10 HIRES 0,1:MULTI 0,1,2
20 PLOT 80,100,1
30 PAUSE 5
40 NRM
```

TYPE: RUN <RETURN>

RESULT: A black dot is plotted at the centre of the screen. After five seconds, the normal screen is displayed.

EXAMPLE: To plot a dotted curve:

ENTRY:
```
10 HIRES 0,1
20 FOR X = 0 TO 320 STEP .5
30 Y = 100 + SIN(X/30)*90
40 PLOT X,Y,1
50 NEXT
1000 GOTO 1000
```

TYPE: RUN <RETURN>

RESULT: A black sine wave is drawn.

6.5.9 TEST

FORMAT: variable = TEST (x,y)

PURPOSE: To determine if something has been drawn at a screen location.

TEST allows you to examine the status of a location on a graphics screen. The parameters x and y are the screen coordinates of the point being tested. If a dot has been plotted at that point, the plot type of the dot is returned (see Section 6.4). A value of 0 is returned if no dot is present. The dot may be any part of a graphics shape.

EXAMPLE: To generate a line that terminates when it touches another line:

ENTRY:
```
10 REM"*** EXAMPLE OF TEST ***
20 HIRES 0,1
25 FOR X = 0 TO 200
30 PLOT 200,X,1:NEXT
40 FOR I = 1 TO 320
50 IF TEST(I,100) = 1 THEN 70
60 PLOT I,100,1:NEXT
70 PAUSE 5
80 NRM
```

TYPE: RUN <RETURN>

RESULT: The horizontal line stops when it touches the vertical line. After five seconds, the normal screen is displayed.

6.5.10 LINE

FORMAT: LINE x,y,x1,y1,plot type

PURPOSE: To plot a line.

LINE draws a line from one point on the screen to another. The parameters x and y are the screen coordinates of the start of the line. The parameters x1 and y1 are the coordinates of the end of the line. Plot type is as described in Section 6.4.

EXAMPLE: To draw a diagonal line across the screen:

ENTRY:
```
10 HIRES 0,1
20 LINE 0,0,320,200,1
30 PAUSE 5
40 NRM
```

TYPE: RUN <RETURN>

RESULT: A black line is drawn from the top left corner to the bottom right corner of the screen. After five seconds, the normal screen is displayed.

6.5.11 CIRCLE

FORMAT: CIRCLE x,y,xr,yr,plot type

PURPOSE: To plot a circular shape.

CIRCLE enables you to draw a circular shape on a graphics screen. The parameters x and y specify the screen coordinates of the centre of the shape you wish to draw. The parameters xr and yr indicate the horizontal and vertical radii of the shape respectively. By varying these radii, circles and ellipses of different sizes can be drawn. Plot type is as described in Section 6.4.

NOTE

Because the screen is rectangular rather than square, x and y radii of the same length will not enable you to draw a perfect circle on the screen. In order to do this in high-resolution mode, the x radius must equal the y radius multiplied by 1.4 In multi-colour mode, the x radius must equal the y radius multiplied by 1.6. However, if you wish to dump a Multi-Colour or High-Resolution screen containing circles on the printer, to obtain printed round shapes, the values of the x and y radii must be equal. See Section 7.13.2 for details of printing.

EXAMPLE: To draw a circular shape in high-resolution mode:

ENTRY:
```
10 HIRES 0,1
20 CIRCLE 160,100,52,40,1
30 PAUSE 5
40 NRM
```

TYPE: RUN <RETURN>

RESULT: A black circle is drawn in the centre of the screen. After five seconds, the normal screen is displayed.

EXAMPLE: To draw an ellipse in multi-colour graphics mode:

ENTRY:
```
10 HIRES 0,1:MULTI 2,3,4
20 CIRCLE 80,100,60,30,1
30 PAUSE 5
40 NRM
```

TYPE: RUN <RETURN>

RESULT: A red ellipse is drawn. After five seconds, the normal screen is displayed.

6.5.12 ARC

FORMAT: ARC x,y,sa,ea,i,xr,yr,plot type

PURPOSE: To draw an arc of a circular shape.

The ARC command enables you to draw part of the circumference of a circular shape. The parameters x and y are the screen coordinates of the centre of the circular shape from which the arc is drawn. Parameters sa and ea define the start and end angles of the arc respectively. The parameter i specifies the plotting increment, i.e. the interval in degrees between each point on the arc. To obtain a solid arc, this value is 1. A larger value separates the dots that make up the arc. Parameters xr and yr indicate the vertical and horizontal radii respectively of the circular shape of which the arc is part. Plot type is as described in Section 6.4.

EXAMPLE: To draw two arcs of the same circular shape:

ENTRY:
```
10 HIRES 0,1
20 ARC 160,100,30,150,1,40,40,1
30 ARC 160,100,210,330,1,40,40,1
40 PAUSE 5
50 NRM
```

TYPE: RUN <RETURN>

RESULT: A pair of black 'brackets' is drawn. After five seconds, the normal screen is displayed.

6.5.13 ANGL

FORMAT: ANGL x,y,angle,xr,yr,plot type

PURPOSE: To draw the radius of a circle.

The ANGL command allows you to draw the radius of a circle without having to display its circumference. The parameters x and y are the screen coordinates of the centre of the circle. 'angle' is the angle, in degrees, at which the radius is drawn relative to the perpendicular, e.g. a radius drawn at an angle of 45 degrees would be at the 3 o'clock position on a clock-face. Parameters xr and yr are the horizontal and vertical radii respectively of the circular shape of which the radius is part. Plot type is as described in Section 6.4.

EXAMPLE: To draw a wheel:

ENTRY:
```
10 HIRES 0,1
20 CIRCLE 160,100,40 * 1.4,40,1
30 CIRCLE 160,100,45 * 1.4,45,1
40 FOR X = 0 TO 360 STEP 22.5
50 ANGL 160,100,X,40 * 1.4,40,1
60 NEXT
70 PAUSE 10
80 NRM
```

TYPE: RUN <RETURN>

RESULT: A black 'spoked' wheel is drawn. After ten seconds, the normal screen is displayed.

EXAMPLE: To draw a fan:

ENTRY:
```
10 HIRES 0,1
20 FOR X = 0 TO 170 STEP 5
30 ANGL 160,100,X,40,40,1
40 NEXT
50 FOR X = 170 TO 0 STEP-5
60 ANGL 160,100,X,40,40,0
70 NEXT
80 PAUSE 10
90 NRM
```

TYPE: RUN <RETURN>

RESULT: A fan is opened and then closed. After ten seconds, the normal screen is displayed.

6.5.14 PAINT

FORMAT: PAINT x,y,plot type (0,1,2,3 only)

PURPOSE: To fill an enclosed area with colour

PAINT fills in a graphics shape with the colour defined by plot type (see Section 6.4). The area to be painted MUST be completely enclosed or painting will take place over the whole screen. The area to be painted is specified by the x and y coordinates of ANY point within its boundaries. In high-resolution mode, the same area may only be painted once. This can be overcome by clearing the screen, changing the plotting colours with the LOW COL command (see Section 6.5.6) re-drawing the shape and then painting it again. In multi-colour mode, the same area may be painted with a different colour as often as you wish.

EXAMPLE: To draw a black rectangle and paint it in yellow:

ENTRY:
```
10 HIRES0,1
20 REC 120,60,40,40,1
30 LOW COL 7,1,0
40 PAINT 130,70,1
50 PAUSE 5
60 NRM
```

TYPE: RUN <RETURN>

RESULT: A black square is drawn in the centre of the screen and filled in with yellow. After five seconds, the normal screen is displayed.

EXAMPLE: To draw a coloured pie chart:

ENTRY:
```
 10 HIRES 0,1:MULTI 5,4,6
 20 CIRCLE 80,100,48,78,1
 30 ANGL 80,100,120,48,78,1
 40 ANGL 80,100,160,48,78,1
 50 ANGL 80,100,220,48,78,1
 60 ANGL 80,100,330,48,78,1
 70 PAINT 90,35,1
 80 PAINT 60,60,3
 90 PAINT 90,120,2
105 LOW COL 7,4,6
110 PAINT 80,110,1
120 PAUSE 5
130 NRM
```

TYPE: RUN <RETURN>

RESULT: A four-segment pie chart is drawn in the centre of the screen and each segment is painted a different colour. After five seconds, the normal screen is displayed.

6.5.15 BLOCK

FORMAT: BLOCK x,y,x1,y1,plot type

PURPOSE: To draw a fully shaded block of colour.

The BLOCK command draws a rectangle and fills it with colour all at the same time. This single command performs the same function as drawing a rectangle with the REC command and colouring it with the PAINT command. The BLOCK command carries out both operations at once. Note, however, that with BLOCK, the colour of the sides of the rectangle are the same as that of the inside of the shape.

The BLOCK command is useful if you wish to create several adjacent blocks of different colours without separating them with lines. The parameters x and y specify the top left-hand corner of the block of colour you wish to display. Parameters x1 and y1 are the coordinates of the bottom right-hand corner of the block. Plot type is as described in Section 6.4.

EXAMPLE: To draw two blocks in different colours:

ENTRY:
```
10 HIRES 0,1: MULTI 2,6,1
20 BLOCK 10,50,50,90,1
30 BLOCK 51,50,90,90,2
40 PAUSE 5
50 NRM
```

TYPE: RUN <RETURN>

RESULT: A red block is displayed adjacent to a blue block. After five seconds, the normal screen is displayed.

6.5.16 DRAW

FORMAT: DRAW "nnnnnnn....9",x,y,plot type

PURPOSE: To design a shape.

The DRAW command allows you to design a shape and then display it on the screen. The shape is designed in the same way as drawing a picture on a piece of paper without removing the pencil. There is, however, one important difference - you can instruct the COMMODORE 64 to move the pencil and not make a mark. (See Section 6.5.17 for an example.)

The x and y parameters in the DRAW command are the coordinates of the point on the screen where the drawing of the shape begins, i.e. its origin. Each n within the quotation marks is an instruction telling the COMMODORE 64 how to move the pencil when drawing the shape. A maximum of 74 instructions can be placed within the quotation marks on any one program line. You may, however, add strings of instructions together up to a maximum of 255. To continue the shape thereafter, a new origin must be specified beginning where the previous string ended. Each instruction and its corresponding number is shown below:

NUMBER	INSTRUCTION
0	Move one pixel to the right.
1	Move one pixel up.
2	Move one pixel down.
3	Move one pixel to the left.
5	Move one pixel to the right and plot a dot.
6	Move one pixel up and plot a dot.
7	Move one pixel down and plot a dot.
8	Move one pixel to the left and plot a dot.
9	Stop drawing.

Note that the instructions above merely design a shape. The shape cannot be drawn until the DRAW and ROT commands are both incorporated into the program (see the following section), i.e the DRAW command specifies the shape and the ROT command generates it. Plot type is as described in Section 6.4.

EXAMPLE: To design a bell:

ENTRY:
```
10 A$ = "57575757878787575757575
   77777777777775757575757575788888888888888"
20 A$ = A$ + "88888888888888865656565656565
   6666666666666"
30 A$ = A$ + "56565656565686886565656"
```

RESULT: When this section of the program is run, the design instructions for a bell are stored in the variable A$.

6.5.17 ROT

FORMAT: ROT r,s

PURPOSE: To display a shape in a specified orientation and size.

The ROT command allows you to display a shape created by the DRAW command (see the section above) at a specified angle of rotation in a defined size. The parameter r specifies by how much the shape is to be rotated relative to the perpendicular about its origin, i.e. the point on the screen from which the shape was drawn. This value of r (range 0 thru 7) defines the angle of rotation as shown in the table below:

ROTATION NUMBER	DEGREES OF ROTATION
0	0
1	45
2	90
3	135
4	180
5	225
6	270
7	315

The second parameter in the ROT command defines the displayed size of the shape you have designed. A "1" in this position indicates that the shape is to be displayed at normal size, i.e. each parameter in the draw command represents one pixel. Any increase in this figure causes a corresponding increase in size.

NOTE

If you specify too large a size for the shape you have designed, it will disappear from the screen when it is displayed. Always ensure therefore that this figure is kept at a realistic level.

EXAMPLE: To display, in normal and enlarged size, the shape designed in the previous section:

ENTRY:
```
10 A$ = "575757578787875757575757
   777777777777575757575757578888888888888"
20 A$ = A$ + "88888888888888865656565656565
   6666666666666"
30 A$ = A$ + "56565656565686868656565656"
40 HIRES 0,1
45 FOR Y = 0 TO 7
50 FOR X = 1 TO 3
60 ROT Y,X
70 DRAW A$,160,80,1
80 PAUSE 1
90 DRAW A$,160,80,0
100 NEXT:NEXT
110 FOR X = 3 TO 1 STEP -1
115 FOR Y = 7 TO 0 STEP -1
120 ROT Y,X
130 DRAW A$,160,80,1
140 PAUSE 1:DRAW A$,160,80,0:NEXT:NEXT
150 GOTO 45
```

TYPE: RUN <RETURN>

RESULT: A bell is displayed at seven different angles of rotation in three sizes

6.5.18 CSET

FORMAT: CSET n

PURPOSE: To select either of the COMMODORE 64 character sets or recall and display the last graphics screen.

The CSET command serves one of three functions depending on the value of the parameter n. CSET 0 allows you to select the COMMODORE 64 Upper Case/Graphics character set. CSET 1 enables you to select the Upper/Lower Case character set. CSET 2 re-displays the last graphics screen that was shown.

> NOTE
>
> When recalling a multi-colour graphics screen, you must always follow CSET 2 with the command MULTI (see Section 6.5.3) using the same parameters that were originally assigned to this command.

EXAMPLE: To print a string using alternate character sets:

ENTRY:
```
10 PRINT"<SHIFT CLR/HOME>"
20 PRINT AT (12,14)"SIMONS' BASIC"
30 CSET 1:PAUSE 1:CSET 0:PAUSE 1
40 GOTO 10
```

TYPE: RUN <RETURN>

RESULT: The character string "SIMONS' BASIC" is displayed at the centre of the screen first in upper case and then lower case letters.

EXAMPLE: To re-display a previously created high-resolution screen:

ENTRY:
```
10 HIRES 0,1:MULTI 0,4,6
15 FOR I = 1 TO 20
20 A = INT(90 * RND(1)) + 2: B = INT(90 * RND(1)) + 2
25 C = INT(90 * RND(1)) + 2: D = INT(60 * RND(1)) + 2
27 P = INT(3 * RND(1)) + 1
30 REC A,B,C,D,1
35 PAINT A + 1,B + 1,P
37 NEXT I
40 PAUSE 2:CSET 0
50 PRINT"<SHIFT CLR/HOME>PRESS ANY KEY TO RE-DISPLAY"
60 PRINT"(<CURSOR DOWN>)THE LAST SCREEN"
70 GET A$: IF A$ ="" THEN 70
80 CSET 2: MULTI 0,4,6: PAUSE 2:CSET 0
90 GOTO 50
```

TYPE: RUN <RETURN>

RESULT: 20 coloured squares are drawn on a multi-colour graphics screen. After a short delay, the normal screen appears with the message PRESS ANY KEY TO RE-DISPLAY THE LAST SCREEN.

ACTION: Press any key.

RESULT: The graphics screen is re-displayed.

6.6 PRINTING TEXT ON A GRAPHICS SCREEN

6.6.1 CHAR

FORMAT: CHAR x,y,poke code,plot type,size

PURPOSE: To print a character on a graphics screen.

The CHAR command allows you to display text character by character on a high-resolution or multi-colour graphics screen. The parameters x and y specify the location of the character on the screen. The next parameter in the command is the 'poke' code of the character you wish to display. (A full list of poke codes is contained in your COMMODORE 64 User's Guide.) Plot type is as described in Section 6.4. The last parameter in this command specifies the height of the character in the range 1 thru 8. A "1" in this position indicates that the character is to be displayed at its normal size, i.e. eight pixels high. Any increase in this figure causes a corresponding increase in character height, e.g. a value of 3 would display the character at a height of 24 pixels. The width of characters CANNOT be varied.

NOTE

User-defined graphics characters CANNOT be used on a high-resolution or multi-colour graphics screen.

EXAMPLE: To display characters at twice their normal size:

ENTRY:

```
10 REM"*** EXAMPLE OF CHAR ***
20 HIRES 0,1
30 FOR J = 1 TO 12
40 FOR I = 0 TO 40
50 CHAR I * 8, A,I + J * 40,1,2
60 NEXT:A = A + 15: NEXT
70 PAUSE 5
80 NRM
```

TYPE: RUN <RETURN>

RESULT: The entire COMMODORE 64 character set is displayed at double its normal height. After a five second delay, the normal screen is displayed.

6.6.2 TEXT

FORMAT: TEXT x,y,"(CTRL a) character string",plot type,s,i

or: TEXT x,y,"(CTRL b) character string",plot type,s,i

PURPOSE: To print a character string on a graphics screen.

TEXT enables you to print character strings on a graphics screen. The parameters x and y specify the screen coordinates of the first letter of the string. The next parameter is the string itself. The control character preceding the string indicates whether the text is to be displayed in upper or lower case letters. To display text in upper case:

1. Type the first set of quotation marks.
2. Hold down the CTRL key and press the 'a' key. (A reverse-field 'A' is displayed.)
3. Enter the character string.
4. Type the last set of quotation marks.

To display text in lower case:

1. Type the first set of quotation marks.
2. Hold down the CTRL key and press the 'b' key. (A reverse-field 'B' is displayed.)
3. Enter the character string.
4. Type the last set of quotation marks.

You may also mix upper and lower case letters in the same string. To do this, hold down the CTRL key and press the 'a' key before the characters you wish to display in upper case, and hold down the CTRL key and press the 'b' key before the characters you wish to display in lower case.

Plot type is as described in Section 6.4. The parameter s specifies the height of each character in the string in the range 1 thru 8. A "1" in this position specifies normal-sized characters. Any increase in this figure causes a corresponding increase in character size, e.g. if you specified a character size of 8, the text would be displayed at eight times its normal height. The width of characters CANNOT be changed. The last parameter i, defines the number of pixels between each character in the string. For normal spacing, this figure is 8. Any increase in this figure creates a correspondingly larger space between characters.

EXAMPLE: To display two character strings on a high-resolution screen:

ENTRY:

```
10 REM"*** EXAMPLE OF TEXT ***
20 HIRES 0,1
30 FOR I = 1 TO 30
40 X = INT(320 * RND(1)):Y = INT(200 * RND(1))
50 LINE 160,100,X,Y,1:NEXT
60 TEXT 60,20,"<CTRL B>TEXT ON THE HIRES SCREEN",1,2,8
70 TEXT 20,180,"<CTRL A>ANYWHERE <CTRL B>YOU LIKE
!",1,1,16
80 PAUSE 5
90 NRM
```

RESULT: When the program is run, a series of random lines are drawn. Two messages are then displayed, the first in lower case using characters at eight times their normal size, the second in upper and lower case using double spaces between letters.

SECTION SEVEN
SCREEN MANIPULATION

7.1 INTRODUCTION

Several comprehensive low-resolution graphics and screen data-handling features are included in SIMONS' BASIC.

The BCKGNDS command allows you to define the colour of the background of a character. The FLASH command switches characters in a defined colour into reverse field and then back again at a specified interval. BFLASH flashes the border surrounding the screen in a similar fashion. OFF terminates the FLASH command.

The FCHR command enables a defined area of the screen to be filled with a selected character. The FCOL command fills a specified section of the screen with a designated colour. FILL combines these commands by enabling you to fill a defined area of the screen with a specific character in a particular colour.

The MOVE command duplicates a specified section of the screen at another screen location. The INV command enables you to inverse the characters in a specified section of the screen, i.e. change normal characters into reverse-field and vice-versa.

The COPY command allows you to print the contents of a graphics screen using your Commodore serial printer. HRDCPY carries out the same function for normal screen data.

The SCRSV command enables you to store a low resolution screen. SCRLD allows you to recall and display a stored screen.

Section Seven also contains commands for scrolling a defined area of the screen in a designated direction. This may be done with wrap round, i.e. letters scrolling off of one side of the defined area and re-appearing at the other, or with blanking, i.e. letters scrolling off the defined area but not re-appearing.

Note that the commands in this section may be used in direct mode or as part of a program.

7.2 BCKGNDS

FORMAT: BCKGNDS sc,b1,b2,b3

PURPOSE: To change the background colour of a character.

When each character on the keyboard is displayed, it occupies an 8 by 8 pixel square on the screen. (A pixel is the smallest addressable point on the screen.) The colour of the square is normally that of the rest of the screen (except of course if the character is displayed in reverse-field). The BCKGNDS command allows you to change the colour of this square both for the regular screen and for reverse field. Note that only those characters inscribed on the top of each key may be used with the BCKGNDS command. Graphics characters CANNOT be used.

The parameter sc of the BCKGNDS command defines the colour of the screen. The next three parameters specify the background colour of a shifted character, a reverse-field unshifted character and a reverse-field shifted character respectively.

EXAMPLE: To print a message using three different character background colours: (Note that in the following program the characters in italics must be typed with the SHIFT key held down.)

ENTRY:

```
10 PRINT"<SHIFT CLR/HOME>"
20 BCKGNDS 1,3,5,6
30 PRINT"THIS IS AN EXAMPLE":PRINT
40 PRINT"<CTRL RVS ON>OF THE SIMONS' BASIC":PRINT
50 PRINT"<CTRL RVS ON>BCKGNDS COMMAND":PRINT
```

TYPE: RUN <RETURN>

RESULT: Three lines of text are displayed, the first on a cyan background, the second on green and the third on blue.

7.3 FLASH

FORMAT: FLASH colour,speed

or: FLASH colour

PURPOSE: To flash a screen colour.

The FLASH command enables you to alternate a specific screen colour between normal and reverse field display either once every four seconds or at a defined speed. This defined speed can range from 1 thru 255. Each unit of speed is approximately one sixtieth of a second and, once initialized, flashing continues until the OFF command (see the following section) is used. Note that FLASH cannot be used on a high-resolution or multi-colour graphics screen.

EXAMPLE: To flash, at a defined speed, those areas of the screen coloured red:

ENTRY:
```
10 PRINT"<SHIFT CLR/HOME>"
20 PRINT AT(12,10)"WHY,<CTRL/3> HELLO <CTRL/1>THERE"
30 FLASH 2,10
1000 GOTO 1000
```

TYPE: RUN <RETURN>

RESULT: The word "HELLO" flashes on and off continuously approximately every sixth of a second.

EXAMPLE: To cause those areas of the screen coloured black to flash every four seconds:

ACTION: Enter the program from the previous section and then list it on the screen.

COMMAND: FLASH 0 <RETURN>

RESULT: The program listing flashes on and off every four seconds.

7.4 OFF

FORMAT: OFF

PURPOSE: To turn the FLASH command off.

OFF terminates the FLASH command. Note that the resulting colour of the characters that have been flashed depends on when the OFF command is used.

EXAMPLE: To turn off the FLASH command in the program above:

ENTRY:
```
10 PRINT"<SHIFT CLR/HOME>"
20 PRINT AT(12,10)"WHY,<CTRL/3> HELLO <CTRL/1>THERE"
30 FLASH 2,10
40 PAUSE 10
50 OFF
60 PRINT AT(12,10)"WHY,<CTRL/3> HELLO <CTRL/1>THERE"
```

TYPE: RUN <RETURN>

RESULT: The word "HELLO" flashes for ten seconds only.

7.5 BFLASH

FORMAT: BFLASH speed,c1,c2

or BFLASH 0

PURPOSE: To flash, or turn off flashing, the screen border.

BFLASH allows you to flash the border surrounding the COMMODORE 64 screen. The first parameter in the command specifies the flashing speed in the range 1 thru 255. Each unit is approximately one sixtieth of a second. The parameters c1 and c2 are the numbers of the colours with which the border will be flashed. BFLASH 0 turns flashing off. Note that the resulting colour of the border depends on when the command is executed.

EXAMPLE: To flash the border in red then blue:

COMMAND: BFLASH 25,2,6 <RETURN>

RESULT: The screen border flashes continuously, first in red, then in blue, changing about every third of a second.

EXAMPLE: To flash the screen border and then turn flashing off:

ENTRY:
```
 10 BFLASH 25,2,6
 20 PAUSE 5
 30 BFLASH 0
1000 GOTO 1000
```

TYPE: RUN <RETURN>

RESULT: The border flashes in red then blue for five seconds.

7.6 FCHR

FORMAT: FCHR r,c,w,d,code

PURPOSE: To fill an area of the screen with a character.

The FCHR command enables you to fill a defined area of the screen with a specific character. The parameters r and c are the row and column coordinates of the start of the screen to be filled. Rows are numbered 0 to 24 and columns from 0 to 39. The parameters w and d define the width and depth of the screen area respectively. Width is measured in characters and depth in rows. The last command parameter is the 'poke' code of the character you wish to display. (A full list of poke codes is contained in your COMMODORE 64 User's Guide.)

EXAMPLE: To display a block of 'A's:

ENTRY:
```
10 PRINT"<SHIFT CLR/HOME>"
15 FCOL 0,0,10,10,0
20 FCHR 0,0,10,10,1
30 GOTO 20
```

TYPE: RUN <RETURN>

RESULT: A ten by ten block in the top left corner of the screen is filled with A's.

7.7 FCOL

FORMAT: FCOL r,c,w,l,colour

PURPOSE: To change a character colour.

The FCOL command changes the colour of all characters in a specified screen area to a defined colour. As in the FCHR command (see the previous section), the first four command parameters define the area of the screen you wish to use. The last parameter is the number of the new colour for each character that appears in this area. (A list of colour numbers appears in Section 6.3 of this manual.)

EXAMPLE: To change the character colour from black to red:

ENTRY:
```
10 PRINT"<SHIFT CLR/HOME>"
15 FCOL 12,15,10,10,0
20 FCHR 12,15,10,10,1
30 FCOL 12,15,5,5,2
```

TYPE: RUN <RETURN>

RESULT: A block of 100 As is displayed. One quarter are coloured red, the remainder are black.

7.8 FILL

FORMAT: FILL r,c,w,l,code,colour

PURPOSE: To fill a defined area on the screen with a specific character in a particular colour.

FILL allows you to fill a defined area of the screen with characters of a specific colour and type. As in the FCHR command (see Section 7.6), the first four parameters in the FILL command define the area of the screen to be used. The next parameter is the poke code of the character to be displayed. (A list of poke codes appears in your COMMODORE 64 User's Guide.) The final parameter in the FILL command is the colour in which you wish to display the character.

EXAMPLE: To display solid blocks of colour:

ENTRY:
```
10 REM"*** EXAMPLE OF FILL ***
20 PRINT"<SHIFT CLR/HOME>"
30 X = INT(40 * RND(1)):Y = INT(25 * RND(1))
40 X1 = INT(20 * RND(1)) + 1:Y1 = INT(15 * RND(1)) + 1
50 IF X + X1 > 40 OR Y + Y1 > 25 THEN 30
60 FILL Y,X,X1,Y1,160,INT(16 * RND(1)):GOTO 30
```

TYPE: RUN <RETURN>

RESULT: Blocks of different colour are displayed in random positions on the screen.

ACTION: Hold down the RUN/STOP key and press the RESTORE key.

RESULT: The normal screen is displayed.

7.9 MOVE

FORMAT: MOVE r,c,w,l,dr,dc

PURPOSE: To duplicate a section of the screen.

MOVE enables you to re-display a defined block of the screen elsewhere on the screen. The first four command parameters define the screen area you wish to reproduce (see Section 7.6). The last two parameters specify the row and column coordinates of the top left corner of the area into which the information will be reproduced.

NOTE

The depth of the screen area to be duplicated added to the row number of the area into which the information is to be reproduced must not exceed 25, i.e. the height of the screen. Likewise, the width of the area to be duplicated plus the column number of the area into which the data is to be reproduced must not be greater that 40, i.e. the width of the screen. If you exceed these totals, the message "BAD MODE" will be displayed and you must re-enter the MOVE command again.

EXAMPLE: To duplicate a block of text:

ENTRY:

```
10 REM"*** EXAMPLE OF MOVE ***
20 PRINT"<SHIFT CLR/HOME><CTRL/1>PRESS SPACE BAR"
30 PRINT"<CTRL 2>TO MOVE THIS AREA"
40 PRINT"<CTRL 3>TO ANOTHER PART"
50 PRINT"<CTRL 4>OF THE SCREEN."
60 GET A$:IF A$ <> " " THEN 60
70 MOVE 0,0,17,5,15,20
80 GOTO 80
```

TYPE: RUN <RETURN>

ACTION: Press the Space Bar.

RESULT: The message in the top left corner of the screen is duplicated in the bottom right.

7.10 INV

FORMAT: INV r,c,w,l

PURPOSE: To inverse a specified screen area.

The INV command causes all normal characters within a defined screen area to be displayed in reverse field. Any character already displayed in reverse field will be displayed normally. (Note to the more advanced programmer: the INV command simply sets or clears bit 7 of the character.)

EXAMPLE: To inverse a block of text:

ENTRY:
```
10 REM"***EXAMPLE OF INV ***
20 PRINT"<SHIFT CLR/HOME>PRESS THE SPACE BAR
30 PRINT"TO INVERSE THIS"
40 PRINT"AREA OF SCREEN!"
50 GET A$:IF A$ <> " " THEN 50
60 INV 0,0,19,4
70 GOTO 50
```

TYPE: RUN <RETURN>

ACTION: Press the space bar.

RESULT: The message shown is displayed in reverse field.

ACTION: Press the space bar again.

RESULT: The message is displayed normally.

7.11 SCROLLING

FORMAT: direction W,sr,sc,ec,r

or: direction B,sr,sc,ec,er

PURPOSE: To scroll an area of the screen.

SIMONS' BASIC provides a command to enable you to scroll specified areas of screen data in any one of four directions. The first parameter in a scrolling command specifies the direction of scrolling - LEFT, RIGHT, UP or DOWN.

The second command parameter is either a W or a B to indicate scrolling with 'wrap round' or 'blanking' respectively. If a section of the screen is scrolled with 'wrap round', any characters within the specified screen area will scroll off the edge of this area and re-appear at the opposite edge. 'Blanking' differs from 'wrap round' in that characters that are scrolled off the screen do not re-appear.

The parameters sr and sc in a scrolling command define the row and column coordinates of the start of the area you wish to scroll. Parameters ec and er specify the column and row coordinates of the end of the scroll area. Scrolling commands may be combined in order to scroll different areas of the screen in varying directions. The maximum height and width of any scroll area cannot exceed 24 lines down and 23 characters across respectively. Note that scrolling cannot be used on high-resolution or multi-colour graphics screens.

EXAMPLE: To scroll two areas of the screen in different directions:

ENTRY:
```
10 PRINT"<SHIFT CLR/HOME>"
20 FOR X = 0 TO 39
30 Y = INT(10 * SIN(X/π)) + 12
40 PRINT AT(X,Y)"*"
50 NEXT
60 LEFTW 0,0,20,25: RIGHTW 0,20,20,25
70 GOTO 60
```

TYPE: RUN <RETURN>

RESULT: A curved line is scrolled across the screen in both directions at the same time.

7.12 STORING AND RECALLING SCREEN DATA

7.12.1 SCRSV

FORMAT: SCRSV 2,8,2,"name,S,W"

or: SCRSV 1,1,1,"name"

PURPOSE: To store data from a low-resolution screen.

The SCRSV command allows you to store the data from a low-resolution screen on cassette or diskette. The first figure following the command is a logical file number. This tells the COMMODORE 64 to open a channel to the disk drive or cassette unit. The second figure specifies the storage device you wish to use. This number is 1 for cassette or 8 for diskette. The third figure is a secondary address. This is a special instruction telling the computer how to store the information. For example, a secondary address of 1 for cassette, instructs the COMMODORE 64 that a file is to be written and that an end-of-file marker is to be placed at the end of the tape when the file is closed. The 'name' is the title you wish to give to the screen data. This name must be unique for each screen you store. You may then use this name in the SCRLD command (see the following section) to recall and display the stored data. The parameter S indicates that the file being accessed is sequential. W instructs the COMMODORE 64 that this file is to be written to rather than read from. When stored, each screen occupies approximately five blocks. Note that the parameters are separated by commas and quotation marks are placed around name and S,W.

The SCRSV command cannot be used to store high-resolution or multi-colour graphics.

EXAMPLE: To draw the French Tricolor and save it on diskette:

ENTRY:
```
10 PRINT"<SHIFT CLR/HOME>"
20 FILL 6,10,20,4,160,2
30 FILL 10,10,20,4,160,1
40 FILL 14,10,20,4,160,6
50 SCRSV 2,8,2,"TRICOLOR,S,W"
80 GOTO 80
```

TYPE: RUN <RETURN>

RESULT: The flag is drawn and then stored on the diskette.

7.12.2 SCRLD

FORMAT: SCRLD 2,8,2,"name"

or: SCRLD 1,1,0,"name"

PURPOSE: To recall stored screen data.

The SCRLD command allows you to recall and display a screen that has been stored with the SCRSV command (see the previous section). The first figure following the command is a logical file number. This tells the COMMODORE 64 to open a data channel to the disk drive or cassette unit. The second figure after the command specifies the device on which the data has been stored. This number is 1 for cassette or 8 for diskette. The third figure is a secondary address. This is a special instruction telling the computer that the information is to be loaded into the same area of memory that it occupied before it was stored. The title you assigned to the screen data is the final parameter and must be enclosed in quotation marks.

EXAMPLE: To recall and display the screen data stored on diskette in the program in the previous section:

TYPE: SCRLD 2,8,2,"TRICOLOR" <RETURN>

RESULT: The Tricolor is recalled from diskette and redrawn on the screen.

7.13 PRINTING SCREEN DATA

7.13.1 INTRODUCTION

SIMONS' BASIC provides two commands which enable you to use a serial printer to reproduce information from either normal or graphics screens. These commands are extremely useful in artwork design or for producing graphs and histograms in statistical representation.

7.13.2 COPY

FORMAT: COPY

PURPOSE: To produce a hard copy of a graphics screen.

COPY outputs the contents of a graphics screen on a serial printer. Note that if you have used the CIRCLE command (see Section 6.5.9) to draw perfect circles on the screen, the radii you have defined for these circles must be changed in order to produce the same display on the printer. To print a perfect circle, the x radius must equal the y radius. To display the screen again, simply change the x radius back to its original value.

EXAMPLE: To display a distorted pie-chart on a high-resolution screen and then produce a round chart on the printer:

ENTRY:
```
10 HIRES 0,1:MULTI 5,4,6
20 CIRCLE 80,100,78,78,1
30 ANGL 80,100,120,78,78,1
40 ANGL 80,100,160,78,78,1
50 ANGL 80,100,220,78,78,1
60 ANGL 80,100,330,78,78,1
70 PAINT 90,35,1
80 PAINT 60,60,3
90 PAINT 90,120,2
105 LOW COL 7,4,6
110 PAINT 80,110,1
120 COPY
1000 GOTO 1000
```

TYPE: RUN <RETURN>

RESULT: A flattened pie-chart is displayed on the screen and then a correct circle is printed.

7.13.3 HRDCPY

FORMAT: HRDCPY

PURPOSE: To print a hard copy of a low-resolution screen.

HRDCPY enables you to reproduce a low resolution screen on a serial printer. This command is most useful in printing forms, invoices etc.

EXAMPLE: To print a message, first on the screen and then on the printer:

ENTRY:
```
10 PRINT"<SHIFT CLR/HOME>"
20 PRINT AT(5,8)"SIMONS' BASIC":PRINT
30 PRINT"THE ULTIMATE IN BASIC AIDS"
40 HRDCPY
50 END
```

TYPE: RUN <RETURN>

RESULT: The data is displayed on the screen and then printed on the Commodore printer.

SECTION EIGHT
SPRITE AND USER-DEFINED GRAPHICS

8.1 INTRODUCTION

Section Eight contains those commands concerned with the generation and animation of 'sprites' and the creation of user-defined graphics. The section is divided into two parts, one for each of these topics.

8.2 SPRITES

8.2.1 INTRODUCTION

A sprite is a programmable object that can be made into a variety of shapes. This object can be moved around the screen by simply telling the computer where the sprite should be placed. (A more detailed description of sprites can be found in your COMMODORE 64 User's Guide.)

A sprite in SIMONS' BASIC is called a 'moveable object block' or MOB. Up to eight independent MOBs can be displayed and animated on the screen at any one time. MOBs can be displayed on normal and graphics screens. There are two types of MOB - high-resolution and multi-colour. A high-resolution MOB is 24 dots wide and 21 dots deep. Each dot on this type of MOB is one pixel wide. A multi-colour MOB is 12 dots wide and 21 dots deep. Here, each point is two pixels wide. A high-resolution MOB can be painted with any ONE of the 15 COMMODORE 64 colours. Multi-colour MOBs can be painted with up to THREE different colours.

In standard BASIC, generation and animation of sprites requires many POKE commands. SIMONS' BASIC replaces POKEs with simple, easy-to-use BASIC-type commands.

The DESIGN command is used to specify the location in the memory of the COMMODORE 64 where the data for each MOB is stored. Each MOB is then designed on a grid within your program listing so that you can see its shape before it is used. MOBs can be used on a normal screen or in conjunction with high-resolution and multi-colour graphics. The MOB SET command sets-up a specified MOB and assigns its primary colour. CMOB is used to assign two extra colours for use when designing a multi-colour MOB. The MMOB command allows you to display and/or move a selected MOB to a specified screen location. RLOCMOB enables you to move a displayed MOB from one screen location to another. The DETECT and CHECK commands are used to determine whether a MOB has collided with another MOB or an item of screen data.

Note that the commands in this section can only be used as part of a program.

The examples used in this section of the manual all build towards a complete program which displays two MOBS on the screen. Therefore, do not use the NEW command between examples and do not run the program until told to do so.

8.2.2 DESIGN

FORMAT: DESIGN c,ad

or: DESIGN c,sa + gc

PURPOSE: To allocate memory space for a MOB.

The DESIGN command reserves sufficient space in the COMMODORE 64's memory for the MOB you are creating. Each MOB uses 64 bytes of memory. The first parameter in the DESIGN command specifies whether the mob is in high-resolution or multi-colour mode. A "1" in this position indicates multi-colour and a "0" high-resolution. The second parameter, ad, defines the start address of the first byte of MOB data. This number must be a multiple of 64 within the range 2048 to 16319 and can be entered in decimal or hexadecimal form. A hexadecimal number must be preceded by a dollar sign ($). If a MOB is to be used on a high-resolution graphics screen, you must add a graphics-constant value of 49152 decimal or $C000 hexadecimal to this figure.

Each 64-byte area of available MOB memory is called a Block. If you divide the MOB data start-address by 64, you will produce a Block Number. This number is used within the MOB SET command (see Section 8.2.3) to set up the MOB.

NOTE

The graphics constant figure MUST NOT be added to the start address when calculating a block number.

The areas available within the COMMODORE 64's memory for MOB data and the associated block numbers are listed below:

BLOCK NUMBERS	MEMORY LOCATIONS
32 - 63	2048 - 4095
128 - 255	8192 - 16383

If you have used the MEM command (see Section 8.3.1), only Blocks 192 thru 255 are available for MOB data.

NOTE

You may set up as many MOBs as the memory of the COMMODORE 64 can accommodate. However, you may only display up to eight MOBs at a time. If, during the course of a program, you wish to get rid of one MOB and create another in its place, simply design the new MOB using the start address of the MOB you are replacing.

EXAMPLE: To allocate memory space for a high-resolution MOB on a normal screen:

ENTRY: 90 DESIGN 0,2048

RESULT: When the multi-colour MOB is created, its data is stored from memory location 2048 onwards in Block 32, i.e. 2048 divided by 64.

EXAMPLE: To allocate memory space for a multi-colour MOB on a normal screen:

ENTRY: 320 DESIGN 1,2112

RESULT: When the high resolution MOB is created, its data is stored from memory location 2112 in Block 33, i.e. 2112 divided by 64.

8.2.3 @

FORMAT: @.....

or: @.....

PURPOSE: To set up the design grid for a MOB.

The @ command allows you to set up a grid for the design of a MOB. The grid is 24 dots wide for high-resolution MOBs and 12 dots wide for multi-colour MOBs. In both cases, the grid is 21 lines deep.

NOTE

Ensure that each line number for the grid is the same length, i.e. two digits or four digits. By doing this, you will avoid indentation of part of the grid, thus facilitating the MOB design process.

As explained in Section 8.1, one colour can be used for high-resolution MOBs and three colours for multi-colour MOBs. The high-resolution MOB colour and the primary colour for a multi-colour MOB are defined in the MOB SET command (see the following section). The two additional multi-colour MOB colours are assigned with the CMOB command (see Section 8.2.4). The colours for each point on the MOB are assigned by using one of the characters on the table below:

HIGH-RESOLUTION MOBS

COLOUR CODE	COLOUR USED
B	The colour assigned in the MOB SET command

MULTI-COLOUR MOBS

COLOUR CODE	COLOUR USED
B	Colour 1 in the CMOB command
C	The colour assigned in the MOB SET command
D	Colour 2 in the CMOB command

You may of course also use the screen background colour in either type of MOB by simply not entering a character.

EXAMPLE: To design a high-resolution MOB:

ENTRY:

```
5    PRINT"<SHIFT CLR/HOME"
10   REM"*** EXAMPLE OF MOBS ***"
80   REM"*** DESIGN THE MOBS ***"
90   DESIGN 0,2048
100  @.........BBBBB..........
110  @........BB...BB.........
120  @.......BB.....BB........
130  @........BB...BB.........
140  @.........BBBBB..........
150  @...........B............
160  @....BBBBBBBBBBBBBBBB....
170  @....BBBBBBB.BBBBBBBB....
180  @....BBBBBB..BBBBBBBB....
190  @...BBB..BBB.BBB..BBBB...
200  @...BBB..BBB.BBB..BBBB...
210  @...BBBBBBB...BBBBBBBB...
220  @..BBBBBBBBBBBBBBBBBBBB..
230  @..BBB...B...B..BBB.BBB..
240  @..BBBB.BB.B.B.B.BB.BBB..
250  @.BBBBB.BB.B.B.BB.B.BBBB.
260  @.BBBBB.BB...B.BBB..BBBB.
270  @.BBBBBBBBBBBBBBBBBBBBBB.
280  @.BBBBBBBBBBBBBBBBBBBBBB.
290  @........................
300  @........................
```

RESULT: When this section of the program is run, the drawing within the grid is stored as MOB data in memory block 32.

EXAMPLE: To design a multi-colour MOB:

ENTRY:

```
320 DESIGN 1,2112
400 @......BB....
410 @.....BCDB...
420 @....BBCCCBBB
430 @...BBCCCB...
440 @.....BCB....
450 @.BB..BCB....
460 @BCCB.BCB....
470 @.BCB.BCB....
480 @..BB.BCB....
490 @...BBCCCB...
500 @..BDCDCDCB..
510 @.BDCDCDCDCB.
520 @.BCDCDCDCCB.
530 @..BCDCDCBB..
540 @...BBBBB....
550 @....B..B....
560 @...B....B...
570 @..B......B..
580 @...B......B.
590 @..BBB....BBB
600 @.....B.....B
```

RESULT: When this section of the program is run, the drawing within the grid is stored as MOB data in memory block 33.

8.2.4 CMOB

FORMAT: CMOB c1,c2

PURPOSE: To set up colours for a multi-colour MOB.

The CMOB command allows you to define the two additional colours for a multi-colour MOB, i.e. the colour of those points on the MOB drawn with the letters B and D in the MOB grid (see Section 8.2.3).

EXAMPLE: Continuing with the program above, to assign the colours black and green to the multi-colour MOB:

ENTRY: 610 CMOB 0,5

RESULT: When the multi-colour MOB is displayed, all points drawn with B are black and all those drawn with D green.

8.2.5 MOB SET

FORMAT: MOB SET mb,blk,col,pr,res

PURPOSE: To set up a MOB

The MOB SET command, as its name suggests, initializes a MOB. The parameter mb specifies the number of the MOB you are setting up. This number must be unique for each MOB. The lower the MOB number the greater its priority over other MOBs, i.e. if two or more MOBs are travelling across the screen, a MOB with a lower number passes over a MOB with a higher number.

The second parameter of the MOB SET command, blk, defines the memory block from which the MOB data will be taken (see Section 8.2.2). The next parameter, col, defines the main MOB colour, i.e. the colour to assign to each point on the MOB drawn with a B in high-resolution mode or a C in multi-colour mode.

The parameter, pr, specifies the priority of the MOB over screen data, i.e. whether you wish the MOB to pass OVER or UNDER other characters on the screen. A "0" in this position gives the MOB priority over screen data, a "1" gives screen data priority over MOBs. The last parameter in the MOB SET command, res, indicates whether the MOB was created in multi-colour or high-resolution mode. A "1" in this position indicates multi-colour; "0" defines high-resolution.

EXAMPLE: To set up the high-resolution MOB in the program from the previous section:

ENTRY: 700 MOB SET 0,32,0,1,0

RESULT: When this section of the program is executed, the high-resolution MOB number 0 in memory block 32 is initialized. When displayed, the MOB is coloured black and passes over all screen data.

EXAMPLE: To set up the multi-colour MOB in the program from the previous section.

ENTRY: 710 MOB SET 1,33,2,0,1

RESULT: When this part of the program is executed, the multi-colour MOB numbered 1 in memory block 33 is set up. When displayed, the MOB has a main colour of red and passes over all screen data.

8.2.6 MMOB

FORMAT: MMOB mn,x1,y1,x2,y2,expansion,speed

PURPOSE: To display and/or move a MOB.

The MMOB command allows you to display a MOB at one point on the screen and then, if you wish, move it to another location. The first parameter, mn, specifies the number of the MOB you wish to display and move. The parameters x1 and x2 are the coordinates of the point on the screen where the MOB will be displayed before it is moved. Parameters x2 and y2 indicate the MOB destination point after movement has taken place. If you do not wish to move a MOB but merely display it, simply use the same coordinates for both the start and end screen locations.

Expansion refers to the size of the MOB when it is displayed. The expansion numbers and resulting display sizes are shown on the table below:

EXPANSION	RESULT
0	The MOB is displayed in normal size
1	The MOB is expanded in the x axis, i.e. displayed at twice its normal width
2	The MOB is expanded in the y axis, i.e. displayed at twice its normal height
3	The MOB is expanded in both axes, i.e. displayed at twice its normal width AND height

Speed specifies the rate at which the MOB will travel. This number must be in the range 1 thru 255: 1 is the fastest speed, 255 is the slowest.

EXAMPLE: To move the MOBs in the program above:

ENTRY:
```
800 MMOB 1,0,0,200,200,2,20
810 MMOB 0,0,0,185,70,3,20
```

RESULT: When this section of the program is run, the high-resolution MOB, expanded in the y axis, is displayed at the top of the screen. The multi-colour MOB appears at twice its normal size at the bottom of the screen.

8.2.7 RLOCMOB

FORMAT: RLOCMOB mn,x,y,expansion,speed

PURPOSE: To move a MOB between two screen locations.

RLOCMOB enables you to move a displayed MOB to a different location on the screen. The parameters x and y are the screen coordinates of the point to which the MOB will be moved. The other parameters are the same as those used in the MMOB command (see the previous section).

EXAMPLE: To relocate both MOBs in the program above:

ENTRY:
```
820 FOR I = 1 TO 20:X = 150 * INT(RND(1)) + 50
830 RLOCMOB 1,X,200,2,10
840 RLOCMOB 0,X-15,70,3,10
850 NEXT
```

RESULT: When this part of the program is run, the two MOBs appear to chase each other across the screen.

8.2.8 DETECT

FORMAT: DETECT n

PURPOSE: To initialize MOB collision detection.

The DETECT command turns on MOB collision detection. A value of 0 assigned to n causes the COMMODORE 64 to detect collision between one MOB and another. If 1 is used as the command parameter, collision detection between MOBs and screen data is initialized. Note that the DETECT command must always be used TWICE. The command is first used to clear the area in the computer's memory which indicates whether collision has taken place. (This area is called the 'sprite collision register'.) The second time the command is used, collision detection is initialized.

EXAMPLE: To clear the sprite collision register in the program above:

ENTRY:
```
825 DETECT 0
```

8.2.9 CHECK

FORMAT: IF CHECK (mn1,mn2) = 0 THEN action

or: IF CHECK (0) = 0 THEN action

PURPOSE: To check for MOB collision.

The CHECK command is used to test for collision between MOBs or between a MOB and screen data. The MOBs on which you wish to test for collision are indicated within the brackets following the command.

A parameter of zero within the brackets causes the COMMODORE 64 to check for collision between any sprite and screen data. If collision has occurred, the defined action is taken.

EXAMPLE: To scroll the high-resolution MOB down the screen and check for collision between it and the multi-colour MOB:

ENTRY:
```
858 FOR P = 70 TO 200
859 DETECT 0:IF CHECK(0,1) = 0 THEN 865
860 RLOCMOB 0,X - 15,P,3,10: NEXT
```

RESULT: When this section of the program is run, the action specified in line 865 (see the following section) is carried out when one MOB touches the other MOB.

8.2.10 MOB OFF

FORMAT: MOB OFF mn

PURPOSE: To clear a MOB from the screen.

The MOB OFF command blanks a MOB from the screen. The parameter mn specifies the number of the MOB you wish to remove.

EXAMPLE: To complete the program above:

ENTRY:
```
855 PRINT AT(X/8 + 2,20)"OH SH..."
856 PAUSE 1
865 PRINT AT(X/8 + 2,20)"OH SHUCKS!!"
870 PAUSE 1
875 MOB OFF 1:RLOCMOB 0,X-15,196,3,10
```

TYPE: RUN <RETURN>

ACTION: Watch the Birdie!!

RESULT: The road-runner is crushed by the weight. (Apologies to all bird lovers.)

8.3 CREATING USER-DEFINED CHARACTERS

8.3.1 INTRODUCTION

SIMONS' BASIC provides a facility to enable you to replace existing keyboard characters with user-defined characters of your own.

The COMMODORE 64 character set is held in ROM, i.e. Read Only Memory. In order to re-define these characters, the character set must be moved into RAM, i.e. Random Access Memory. The MEM command carries out this function. The DESIGN command allows you to specify the character you wish to re-define in terms of its poke code. (A full list of poke codes is contained in your COMMODORE 64 User's Guide.) Each character is designed within a grid. This allows you to view the character as it is being created. Note that user-defined characters CANNOT be used on a graphics screen.

8.3.2 MEM

FORMAT: MEM

PURPOSE: To move the character ROM to RAM.

The MEM command moves the character set in ROM into RAM behind the Kernal. The screen is moved to location $CC00 and sprite data may only be inserted from location $F000, i.e. Block 192, onwards (see Section 8.2.1). To revert back to the original COMMODORE 64 character set, simply hold down the RUN/STOP key and press the RESTORE key.

Figures 8.1 and 8.2 show the configuration of the COMMODORE 64's memory before and after the MEM command has been used.

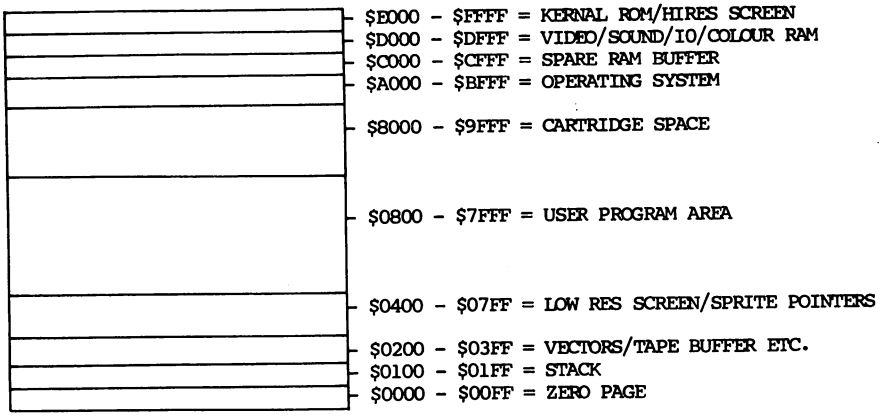


FIGURE 8-1 MEMORY CONFIGURATION BEFORE MEM

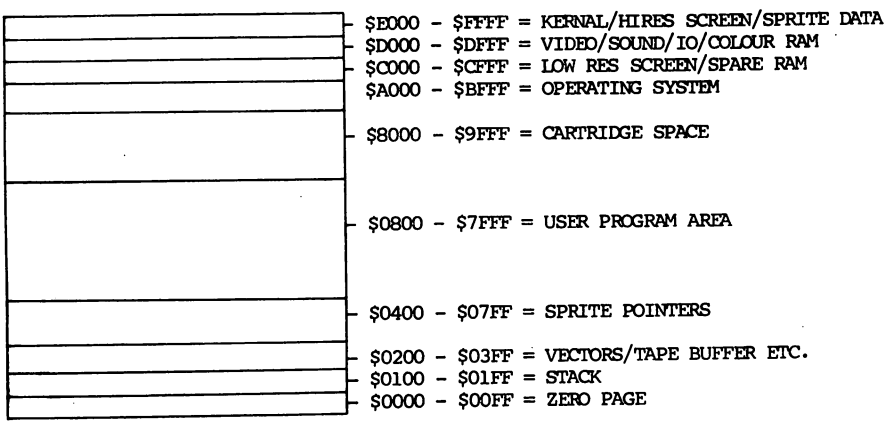


FIGURE 8-2 MEMORY CONFIGURATION AFTER MEM

NOTE

The TRACE command (see Section 2.11.1) cannot be used if a program contains the MEM command.

EXAMPLE: To move the character ROM to RAM:

ENTRY: 10 MEM

RESULT: When this section of the program is run, the COMMODORE 64 character set is moved to RAM in preparation for re-definition of characters.

8.3.3 DESIGN

FORMAT: DESIGN 2,$E000 + ch * 8

PURPOSE: To specify the character which a user-defined graphics character is to replace.

The DESIGN command allows you to define the character which is to be replaced by a user-defined character of your own. Note that there is also a DESIGN command associated with sprite graphics (see Section 8.2.1) using a different format.

User-defined characters can only be used on a low-resolution screen. The digit 2 following the DESIGN command tells the COMMODORE 64 that user-defined character data will follow. Each new character occupies 8 bytes of memory. The hexadecimal figure $E000 is the start address of the character data. The parameter ch is the poke code of the existing character that you wish to change. (A list of poke codes can be found in your COMMODORE 64 User's Guide.) The new character is designed within an 8 by 8 grid (see the following section). It is displayed each time you use the key that is inscribed with the character that has been replaced.

EXAMPLE: To re-define the character "Z":

ENTRY: 20 DESIGN 2,$E000 + 26 * 8

RESULT: When the graphics character has been created, it is displayed each time the letter "Z" is used.

8.3.4 @

FORMAT: @.....

The @ command allows you to set up a grid for the design of a user-defined graphics character. The grid is 8 dots wide and 8 lines deep. The new character is designed by placing a letter 'B' over the appropriate dot on the grid.

> NOTE
> Ensure that each line number for the grid is the same length, i.e. two digits, three digits or four digits. By doing this, you will avoid indentation of part of the grid and facilitate the character design process.

EXAMPLE: To design a 'top hat' character:

ENTRY:

```
20 DESIGN 2,$E000 + 26 * 8
30 @........
40 @........
50 @..BBBB..
60 @..BBBB..
70 @..BBBB..
80 @BBBBBBBB
90 @BBBBBBBB
30 @........
```

ACTION: Type RUN <RETURN>

DISPLAY: READY

ACTION: Press the Z key a few times.

RESULT: The 'top hat' character is displayed.

To revert to the original COMMODORE 64 character set, simply hold down the RUN/STOP key and press the RESTORE key.

SECTION NINE
STRUCTURED PROGRAMMING

9.1 INTRODUCTION

One of the main problems when programming in standard BASIC is the lack of a structured flow to the more involved programs. The use of GOTOs and GOSUBs causes all but the simplest program listings to become incomprehensible - even to the program's author! This illegibility can be eased partially with the use of multiple REM statements to explain what each routine does. This is not only time-consuming but also uses up a great deal of memory space.

SIMONS' BASIC removes these problems with special structured programming commands. These commands largely obviate the need for GOTOs and GOSUBs in your BASIC programs. For example, the PROC command is used to label each program routine you use. (This function equates to Paragraph naming in the Procedure Division in COBOL.) These routines are executed using either the CALL or EXEC commands.

The structure of FOR...NEXT loops is also changed. The REPEAT...UNTIL command allows you to execute a procedure a defined number of times. The LOOP...EXIT IF...END LOOP provides multiple condition-testing within a loop. The normal IF...THEN condition test now includes ELSE to enable you to simplify specification (on the same program line) of the routes to be taken if an expression matches or does not match a pre-defined condition. The RCOMP command allows you to use the previous IF...THEN...ELSE condition test without having to re-enter the code.

Note that the commands in this section may only be used as part of a program.

9.2 CONDITION TESTING AND PROGRAM LOOPS

9.2.1 IF...THEN...ELSE

FORMAT: IF condition THEN true:ELSE:false.

PURPOSE: To test for a condition and branch to one instruction if the condition is true or to another instruction if the condition is false.

The IF...THEN...ELSE command acts in a similar way to the standard BASIC IF...THEN condition test. The one important difference is that branches to specific sections of code can be made for both true and false results to the test (on the same program line). Note that ELSE must always be separated from the preceding and following code with a colon (:).

EXAMPLE: To check the response to a question:

ENTRY:
```
10 PRINT"DO YOU OWN A COMMODORE COMPUTER?"
20 PRINT "PLEASE ANSWER YES OR NO (Y/N)"
30 FETCH "<CLR HOME>",1,A$
40 IF A$ = "Y" THEN 60
50 IF A$ = "N" THEN 70:ELSE:GOTO 30
60 PRINT "CONGRATULATIONS":END
70 PRINT "COMMISERATIONS":END
```

TYPE: RUN <RETURN>

ACTION: When prompted, press either the 'y' or 'n' keys followed by <RETURN>.

RESULT: The appropriate message is displayed depending on whether Y or N is pressed. Any other key results in no action.

9.2.2 REPEAT.....UNTIL

FORMAT: REPEAT loop UNTIL condition is met.

PURPOSE: To perform a program loop until a specified condition is met.

REPEAT....UNTIL carries out the same function as a FOR...NEXT loop in standard BASIC except that instead of specifying how many times the code is to be executed at the start of the loop, the number of loops is determined by a condition test at the end of the code. REPEAT starts the loop; UNTIL tests for a condition , e.g. X>10, which, when true, causes the program to leave the loop. If the condition is not met, the loop is re-executed.

WARNING

YOU MAY NOT HAVE MORE THAN NINE NESTED LOOPS. IF YOU EXCEED THIS FIGURE, THE MESSAGE "? STACK TOO LARGE" IS DISPLAYED.

EXAMPLE: To print the letters of the alphabet from A to G:

ENTRY:
```
10 A = 65
20 REPEAT:PRINT CHR$(A):A = A + 1:UNTIL A>70
30 PRINT "DONE!"
```

TYPE: RUN <RETURN>

DISPLAY:
A
B
C
D
E
F
G
DONE!

9.2.3 RCOMP

FORMAT: RCOMP:true:ELSE:false

PURPOSE: To re-execute the last IF...THEN...ELSE condition test.

RCOMP causes the the most recently defined IF...THEN...ELSE condition test in a program to be repeated. This removes the necessity of having to re-enter the same code again.

EXAMPLE: To repeat the same condition test three times:

ENTRY:
```
10 INPUT A
20 IF A = 10 THEN PRINT "HELLO ";:ELSE: PRINT "BYE ";
30 RCOMP:PRINT "MIKE ";:ELSE:PRINT "STRANGER ";
40 RCOMP:PRINT "WELOCME":ELSE:PRINT "SEE YOU AGAIN I
  HOPE"
50 GOTO 10
```

TYPE: RUN <RETURN>

DISPLAY: ?

TYPE: 10 <RETURN>

DISPLAY: HELLO MIKE WELCOME

TYPE: 5 <RETURN>

DISPLAY: BYE STRANGER SEE YOU AGAIN I HOPE

RESULT: When a figure is typed, this input is tested three times producing the display appropriate to the value entered.

9.2.4 LOOP...EXIT IF...END LOOP

FORMAT: LOOP program loop EXIT IF condition true END LOOP

PURPOSE: To perform a continuous loop until a specified condition is met.

LOOP...EXIT IF ... END LOOP performs a program loop in a similar way to the command REPEAT...UNTIL (See Section 9.2.2) with one important difference. REPEAT...UNTIL only allows condition testing at the end of the loop. LOOP...EXIT IF ... END LOOP allows any number of condition tests to be made within the loop. If a condition is met, the program EXITs to the statement following END LOOP. If the condition is not met, the program loop is re-executed.

WARNING

YOU MAY NOT HAVE MORE THAN FIVE NESTED LOOPS. IF YOU EXCEED THIS FIGURE, THE MESSAGE "? STACK TOO LARGE" IS DISPLAYED.

EXAMPLE: To get a character between A and F from the keyboard:

ENTRY:
```
10 PRINT"ENTER A LETTER BETWEEN A AND F"
20 LOOP
30 GET A$:IF A$ = "" THEN 30
40 EXIT IF ASC(A$)<65
50 EXIT IF ASC(A$)>70
60 PRINT A$;
65 END LOOP
70 PRINT CHR$(13)"NOT IN RANGE":END
```

TYPE: RUN <RETURN>

TYPE: C <RETURN>

DISPLAY: C

TYPE: B <RETURN>

DISPLAY: CB

TYPE: S <RETURN>

DISPLAY: NOT IN RANGE

RESULT: The keyboard is scanned and all letters in the range defined are displayed on the screen. A letter outside the range causes the program to leave the loop.

9.3 PROGRAM PROCEDURES

9.3.1 INTRODUCTION

To facilitate the writing of more structured code, SIMONS' BASIC provides four commands which enable you to label BASIC program routines and then call these routines by name when they are required. To a great extent, this removes the necessity of having to use GOTOs and GOSUBs in your programs. These routines are called 'procedures'. Any procedure that is used frequently in different programs can be stored in a procedure 'library' and then loaded when required.The PROC command (see Section 9.3.2) is used to assign names to procedures. These are then either executed with the CALL command (see Section 9.3.4) or with the EXEC command (see Section 9.3.5). The CALL command acts in the same way as the standard BASIC GOTO command, i.e the program jumps to the start of the procedure and continues execution from that point. The EXEC command acts like a GOSUB, i.e. the program jumps to the named procedure and then returns to the program line following the EXEC command when the procedure has been completed. The completion of the latter must always be indicated by the END PROC command (see Section 9.3.3), which acts in the same way as RETURN in standard BASIC.

The examples used in this section of the manual all build towards a complete program. Therefore, do not use the NEW command between examples and do not RUN the program until told to do so.

WARNING
YOU MAY NOT HAVE MORE THAN FIVE NESTED PROCEDURES. IF YOU EXCEED THIS FIGURE, THE MESSAGE "? STACK TOO LARGE" IS DISPLAYED.

9.3.2 PROC

FORMAT: PROC name

PURPOSE: To label a program routine.

PROC enables you to label program routines and then call these routines by name when they are required. All characters on the line following the PROC command are taken as the name of the procedure. Therefore, PROC and the procedure name must not be followed by any other code on the same program line.

EXAMPLE: To assign the label "INPUT NAME" to a program routine:

ENTRY: 100 PROC INPUT NAME

RESULT: When the program is run, the code following this line forms a procedure called INPUT NAME.

9.3.3 END PROC

FORMAT: END PROC

PURPOSE: To indicate the end of a procedure.

END PROC indicates the end of a 'closed' procedure, i.e. one to be called by the EXEC command (see Section 9.3.5). This command acts in the same way as RETURN in standard BASIC, i.e. when the procedure ends, the program returns to the line following that on which the procedure was called.

EXAMPLE: To set up a procedure for entering the user's name into a variable:

ENTRY:
```
100 PROC INPUT NAME
110 PRINT "WHAT IS YOUR NAME"
120 FETCH "<CLR/HOME>",15,A$
130 END PROC
```

RESULT: When the program is run, the INPUT NAME procedure can be called using the EXEC command (see Section 9.3.5).

9.3.4 CALL

FORMAT: CALL procedure name

PURPOSE: To transfer program execution to a specific line of code.

The CALL command acts in the same way as GOTO in standard BASIC except that a procedure name is used in place of a line number. Everything that follows CALL on the same program line is used as the name of the procedure being called. Therefore, CALL and the procedure name must not be followed by any other code on the same program line. The procedure called must be 'open', i.e. one that does not contain END PROC.

EXAMPLE: The start of a Sort program:

ENTRY:
```
10 PRINT"<SHIFT CLR/HOME>"
20 PRINT"HOW MANY NAMES DO YOU WISH TO SORT?'
30 PRINT"NO MORE THAN 15 NAMES"
32 PRINT"AND NO LONGER THAN TEN CHARACTERS"
34 PRINT"FOR EACH NAME"
40 FETCH"<CRSR DOWN>",2,X
45 IF X > 10 AND X < 16 THEN DIM A$(X)
50 IF X < 16 THEN CALL ENTER NAMES
55 GOTO 10
60 PROC ENTER NAMES
70 FOR I = 1 TO X
80 FETCH"<CLR/HOME>",10,A$(I):PRINT TAB(20)"OK"
90 NEXT
```

RESULT: If the number of names entered is less than 16, the pr continues from the procedure ENTER NAMES in line 60.

9.3.5 EXEC

FORMAT: EXEC procedure name

PURPOSE: To call a program routine and return to the line following the call when the procedure has been completed.

EXEC performs the same function as GOSUB in standard BASIC, i.e. the program jumps to a specific section of code, executes the code and then returns to the line following EXEC when END PROC (see Section 9.3.3) is reached. Everything that follows EXEC on the same program line is taken as the name of the procedure being called. Therefore, EXEC and the procedure name must not be followed by any other code on the same program line.

EXAMPLE: To complete the Sort program from the previous section.

ENTRY:

```
100 EXEC SORT
110 PRINT"<SHIFT CLR/HOME CRSR/DOWN>"
120 FOR I = 1 TO X:PRINT TAB(20)A$(I):NEXT
130 END
140 :
150 :
1000 PROC SORT
1020 M = 1
1030 REPEAT
1040 T = 0:FOR I = 1 TO N - M
1050 IF A$(I) < A$(I + 1) THEN 1070
1060 W$ = A$(I):A$(I) = A$(I + 1):A$(I + 1) = W$:T = 1
1070 NEXT I
1080 M = M + 1
1090 UNTIL T = 0
1110 END PROC
```

TYPE: RUN <RETURN>

ACTION: When prompted, enter up to 15 names, pressing RETURN between each name.

RESULT: The names are sorted and then displayed.

9.4 PROGRAM VARIABLES

9.4.1 INTRODUCTION

The use of variables in standard BASIC can become confusing when many variables are required for different purposes. SIMONS' BASIC resolves this problem by allowing you to use the same variable in two ways - locally within a specific program routine or globally throughout the whole program. This reduces the number of variables you need and, consequently, frees more memory space so that you can write and run longer programs.

The value of each variable within a BASIC program changes depending on when and where the variable is used. The LOCAL command allows you to store the values currently held by the variables, clear them and then use the same variable names within a specific section of code. The GLOBAL command restores the values contained by the variables before the LOCAL command was executed.

9.4.2 LOCAL

FORMAT: LOCAL variable1, variable2, variable3.....

PURPOSE: To assign variables to a specific program routine.

The LOCAL command allows you clear the values of previously defined variables on a temporary basis and then use these variables locally within a specific program routine. The GLOBAL command (see the following section) restores the original values to the variables.

WARNING
THE VARIABLES DEFINED WITH THE LOCAL COMMAND MUST HAVE PREVIOUSLY BEEN DECLARED. FAILURE TO ADHERE TO THIS WARNING WILL RESULT IN THE PROGRAM 'HANGING', i.e. NO FURTHER EXECUTION WILL OCCUR. YOU MUST THEN PRESS THE RUN/STOP AND RESTORE KEYS TO BREAK OUT OF THE PROGRAM.

EXAMPLE: To assign variables locally:

ENTRY:
```
10 REM"*** EXAMPLE OF LOCAL ***
20 PRINT"<SHIFT CLR/HOME>"
30 A$ = "INITIAL VALUE":A% = 123:A = 456.7
40 LOCAL A$,A%,A
50 A$ = "NEW VALUE     ":A% = 789:A = 321.4
60 PRINT A$,A%,A
```

TYPE: RUN <RETURN>

DISPLAY: NEW VALUE 789 321.4

RESULT: The values originally assigned to the variables are stored. New values are then assigned to the variables and these values printed.

9.4.3 GLOBAL

FORMAT: GLOBAL

PURPOSE: To restore original values to local variables.

The GLOBAL command causes all variables that have been used locally within a program routine to be cleared. The values they held before the LOCAL command was used (see the previous section) are then re-assigned.

EXAMPLE: Using the program from the previous section, to restore GLOBAL values to locally used variables:

ENTRY:
```
10 REM"*** EXAMPLE OF LOCAL/GLOBAL ***
20 PRINT"<SHIFT CLR/HOME>"
30 A$ = "INITIAL VALUE":A% = 123:A = 456.7
40 LOCAL A$,A%,A
50 A$ = "NEW VALUE     ":A% = 789:A = 321.4
60 PRINT A$,A%,A
70 GLOBAL
80 PRINT A$,A%,A
```

TYPE: RUN <RETURN>

DISPLAY: NEW VALUE 789 321.4

INITIAL VALUE 123 456.7

RESULT: The values assigned to the variables before the LOCAL command were used are restored.

SECTION TEN
ERROR TRAPPING

10.1 INTRODUCTION

SIMONS' BASIC provides commands to trap program errors in order to prevent your BASIC programs from 'crashing'. The ON ERROR command allows you to branch to a specified point in the program should an error be found. The variable ERRLN contains the number of the program line on which the error has occurred and the variable ERRN contains the error number. By testing the value held in ERRN, you can then take appropriate action, including, if you wish, the generation and display of your own error message. OUT turns off the most recently used ON ERROR command. The NO ERROR command returns you to the normal COMMODORE 64 error-handling routines.

10.2 ON ERROR

FORMAT: ON ERROR: GOTO line number

PURPOSE: To trap program errors.

The ON ERROR command allows you to trap BASIC program errors to prevent your programs from 'crashing'. When an error is found, the program jumps to the line specified with the GOTO. The error number is held in the variable ERRN. The line in which the error has occurred is held in the variable ERRLN. By testing the value held in ERRN, you can check to see which error has occurred and then take any necessary action including, if you wish, the display of an error message of your own.

NOTE

After testing for a specific error and taking the appropriate action, you must always use the OUT command (see the following section) before continuing program execution. This command must also be used if you have stopped a program containing the ON ERROR command and wish to edit some of your code.

The errors that can be trapped by SIMONS' BASIC and the associated error numbers are shown below:

ERROR NUMBER	ERROR
1	Too many files
2	File open
3	File not open
4	File not found
5	Device not present
10	Next without for
11	Syntax
12	Return without gosub
13	Out of data
14	Illegal quantity
15	Overflow
16	Out of memory
17	Undefined statement
18	Bad subscript
19	Re-dimensioned array
20	Division by zero
21	Illegal direct
22	Type mismatch
23	String too long

In the examples that follow, please type in the information EXACTLY as shown. The typing mistakes are deliberate and are included to demonstrate the use of the SIMONS' BASIC error-trapping commands.

EXAMPLE: To trap a SYNTAX error and display a user-defined error message:

ENTRY:
```
  5 REM"*** EXAMPLE OF ERROR HANDLING ***
 10 ON ERROR: GOTO 100
 15 PRIN"<SHIFT CLR/HOME>"
 20 READ B
 25 PRINT B: GOTO 20
 30 DATA 1,2,3,4,5
100 IF ERRN = 11 THEN PRINT"SPELLING MISTAKE IN
LINE";ERRLN
```

TYPE: RUN <RETURN>

DISPLAY: SPELLING MISTAKE IN LINE 15

RESULT: Because the spelling of the BASIC keyword PRINT is wrong, the program has jumped to the error-handling routine at line 100. The program is searched for a SYNTAX error, i.e. error number 11. Because this error has occurred, the user-defined error message is displayed.

ACTION: Press the RUN/STOP key.

DISPLAY: READY

EXAMPLE: To trap an OUT OF DATA error:

TYPE: OUT <RETURN>

(The OUT command is explained in the following section.)

ACTION: Correct the spelling error in line 15 of the program example from the previous section and then enter the following line of code:

```
110 IF ERRN = 13 THEN PRINT"NOT ENOUGH INFORMATION
    IN LINE";ERRLN
```

TYPE: RUN <RETURN>

DISPLAY:

```
1
2
3
4
5
NOT ENOUGH INFORMATION IN LINE 20
```

RESULT: Because there are only five items of data, the program jumps to the error-trapping routine at line 100 when an attempt is made to read item number six. Line 110 tests for error 13, i.e. OUT OF DATA and, because this error has occurred, displays the defined error message.

10.3 OUT

FORMAT: OUT

PURPOSE: To disable the last ON ERROR command.

OUT turns off the most recently used ON ERROR command. This command must always be used if you wish to return to the COMMODORE 64 error-handling routine that has been trapped by the ON ERROR command.

EXAMPLE: Using the program example from the previous section, to turn off the ON ERROR command:

ACTION: Enter the following revised line of code:

```
20 READ B: J = J + 1: IF J = 5 THEN OUT
```

TYPE: RUN <RETURN>

DISPLAY:
1
2
3
4
5
?OUT OF DATA ERROR

RESULT: As there are only four items of data, the program crashes when an attempt is made to read item number 5. The COMMODORE 64 message applicable to this type of error is then displayed.

10.4 NO ERROR

FORMAT: NO ERROR

PURPOSE: To re-enable the COMMODORE 64 error-handling routines.

The NO ERROR command turns off ALL the SIMONS' BASIC error-trapping commands and returns control to the COMMODORE 64 error-handling routines.

EXAMPLE: Using the program from the previous section, to return error-handling control to the COMMODORE 64:

ACTION: Enter the following revised lines of code:

```
15 PRIN"<SHIFT CLR/HOME>"
100 NO ERROR:IF ERRN = 11 THEN PRINT"SPELLING MISTAKE
IN LINE";ERRLN
```

ACTION: Type RUN <RETURN>

DISPLAY: SYNTAX ERROR IN LINE 100

SECTION ELEVEN
MAKING MUSIC WITH SIMONS' BASIC

11.1 INTRODUCTION

Among the attractive features of the COMMODORE 64 is its music-synthesizing capability. With practice and experience, it is possible to reproduce the sounds of many different musical instruments. The music and sound attributes of the COMMODORE 64 are extensive. SIMONS' BASIC has not been designed to utilise all these features. The sound and music commands supplied by the cartridge are intended primarily to introduce you to the art of sound programming on the COMMODORE 64. If you wish to further your music programming skills on Commodore computers, please ask your local dealer for details of the special software and books that deal with this subject.

11.1.1 SOUND SHAPING

Different sounds are produced by different frequencies. The higher the frequency, the higher the note produced. Most personal computers have an audio capability but unlike most others, the COMMODORE 64 gives you the ability to 'shape' each frequency. Shaping simply means telling the computer how each part of a frequency should be played. The volume of a musical note or sound changes from when you first hear it until it dies out and you cannot hear it anymore. All frequencies are generated in four cycles: ATTACK, DECAY, SUSTAIN and RELEASE. These cycles together form a sound 'envelope'. The function of each cycle within the envelope is described below:

ATTACK

This determines the rate at which a frequency rises from zero to peak volume.

DECAY

This defines the rate at which the frequency falls from its peak volume to a middle-ranged volume level.

SUSTAIN

This determines the mid-range volume.

RELEASE

This determines the rate at which the frequency falls from the SUSTAIN level to zero volume.

For the purposes of clarification, a diagram representing a sound envelope is shown in Figure 11-1.

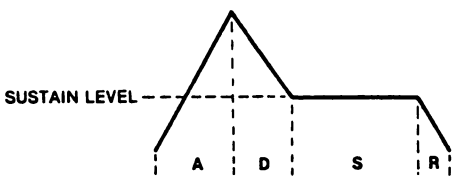


FIGURE 11-1 A SOUND ENVELOPE

11.1.2 SOUND WAVES

The COMMODORE 64 allows you to select the type of sound wave that you wish to use to play the music or sound effects you have created. Each type of sound wave produces a different effect. The waveforms and the effects they produce are described below. A diagram of each waveform is also shown.

TRIANGLE

This waveform is low in harmonics and has a mellow flute-like quality.

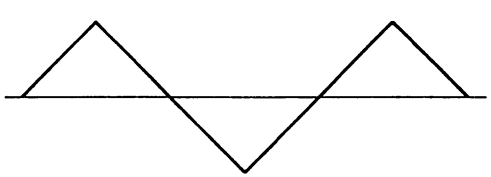


FIGURE 11-2 A TRIANGULAR SOUND WAVE

NOISE

As its name suggests, this waveform produces various types of noise for special sound effects.

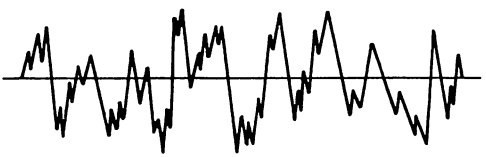


FIGURE 11-5 A NOISE WAVE

11.1.3 PROGRAMMING SOUND

In standard BASIC, playing music or creating special sound effects on the COMMODORE 64 requires the use of multiple POKE commands. This can be tedious and time-consuming but with SIMONS' BASIC, this is no longer a constraint. Special music and sound commands provided by the cartridge remove the need to access memory locations yourself with POKEs.

The VOL command allows you to define how loud or soft your music or sound effects will be played. WAVE enables you to select the type of waveform you wish to use for your sounds. The ENVELOPE command allows you to define the 'shape' of each note within a sound envelope. The MUSIC command is used to compose the sounds you wish to produce, while the PLAY command causes the sounds to be generated. Note that these commands can only be used as part of a program.

The sections that follow describe the format and purpose of each SIMONS' BASIC music command. A brief example of the use of each command is also given.

The examples used in this section of the manual all build towards a complete program which plays a tune. Therefore, do not use the NEW command between examples and do not RUN the program until told to do so.

11.2 MUSIC COMMANDS

11.2.1 VOL

FORMAT: VOL n

PURPOSE: To select music volume.

The VOL command enables you to define the volume level at which the music or sound that follows the command will be played. Volume levels range from 0 thru 15. Level 15 is the loudest volume and 0 turns the sound off. A volume level remains set until a new VOL command is given.

EXAMPLE: To set a volume level of 15:

ENTRY: 10 VOL 15

RESULT: Any sound following this code will be played at the highest volume.

11.2.2 WAVE

FORMAT: WAVE voice number,binary number

PURPOSE: To set the music voice type

The WAVE command allows you to select the type of waveform you wish to use to play your music or sound effects. (See Section 11.1.2.) The first parameter in the WAVE command specifies the 'voice' through which the sound will be played. The COMMODORE 64 has three voices numbered 1 thru 3. Each voice contains the same nine octaves. This means that you can play a sound through one voice and then mix in a sound from another voice.

The second parameter in the WAVE command is a binary number. (Note that with the WAVE command, this number is not preceded by a dollar sign.) This number tells the COMMODORE 64 how to play each sound. Each of the eight bits within the number perform a specific function. To select a function, the associated bit is set, i.e. a 1 is placed in that position. The bits are numbered from 0 thru 7, bit 7 being the **leftmost** bit of the number. The function each bit performs is shown on the following table:

BIT NUMBER	FUNCTION PERFORMED
0	Sets the gate bit (not required)
1	Sets synchronisation
2	Sets ring modulation
3	Sets the test bit (should never be set)
4	Sets Triangular waveform
5	Sets Sawtooth waveform
6	Sets Pulse/Square waveform
7	Noise

These functions are described in greater detail below:

BIT 0 - THE GATE BIT

On a COMMODORE 64 without SIMONS' BASIC, this bit, when set, 'triggers' the Envelope Generator, i.e. it causes the four cycles of a frequency to begin. However, because SIMONS' BASIC sets this bit automatically when the PLAY command is executed (see Section 11.2.5), you must always leave the value of this bit at 0.

BIT 1 - SYNCHRONIZATION

The Synchronization bit enables a note (frequency) from one voice to be synchronized with a note from another. By playing one steady note (static frequency) from a voice and playing multiple notes (variable frequency) from another, a wide range of complex harmonies can be produced. For the best effect, the static frequency should always be lower than the lowest value of the variable frequency. The voice chosen to output the variable frequency determines the voice you can select for the static frequency. This is outlined on the following table:

VARIABLE FREQUENCY VOICE	STATIC FREQUENCY VOICE
1	3
2	1
3	2

The voice number is specified as the first parameter in the WAVE command (see above).

BIT 2 - RING MODULATION

Bit 2, when set, initializes Ring Modulation. This effect is similar to Synchronization (see the previous section) except that both frequency and amplitude (volume) can be varied at the same time to create a 'wow-wow' effect. By varying the frequency of one voice against a static frequency from another, a wide range of non-harmonic structures can be produced for creating bell or gong sounds and special effects.

For Ring Modulation to be audible, a triangular waveform must be selected for the variable frequency voice. This waveform is then replaced with a modulated combination of the output from this and another defined voice. As in Synchronization, the voice chosen to output the variable frequency determines the voice you can select for the static frequency. This is outlined on the following table:

VARIABLE FREQUENCY VOICE	STATIC FREQUENCY VOICE
1	3
2	1
3	2

BIT 3 - THE TEST BIT

This bit is not used in SIMONS' BASIC. Therefore, it must never be set i.e. it must always be 0.

BIT 4 - TRIANGULAR WAVEFORM

A value of 1 in bit 4 sets up a Triangular waveform.

BIT 5 - SAWTOOTH WAVEFORM

A value of 1 in this bit sets up a Sawtooth waveform.

BIT 6 - PULSE/SQUARE WAVEFORM

A value of 1 in bit 6 sets up a Pulse/Square waveform.

BIT 7 - NOISE

A value of 1 in this bit sets up a Noise waveform.

NOTE

In bits 4 thru 7, if one bit is set to 1, the remaining bits must be left at 0.

EXAMPLE: To set up a Triangular waveform for voice 1.

ENTRY: 20 WAVE 1,00010000

RESULT: When this section of the program is executed, the music following the command is played using a Triangular waveform.

11.2.3 ENVELOPE

FORMAT: ENVELOPE vn,a,d,s,r

PURPOSE: To define the 'shape' of a sound.

As explained in Section 11.1.1, the COMMODORE 64 allows you to define an envelope which determines the shape of the sound you wish to play. The ENVELOPE command allows you to design this shape. The parameter vn is the number of the voice through which you wish to play the sound. The parameters a, d, and r specify, respectively, the duration of the ATTACK, DECAY and RELEASE cycles of the frequency to be produced. The duration of the ATTACK, DECAY and RELEASE cycles are measured in units of one thousandth of a second. This is represented by a number in the range 0 thru 15. These numbers and the corresponding time cycles are listed on the table below:

VALUE (TIME/CYCLE)	ATTACK RATE	DECAY RELEASE (TIME CYCLE)
0	2	6
1	8	24
2	16	48
3	24	72
4	38	114
5	56	168
6	68	204
7	80	240
8	100	300
9	250	750
10	500	1500
11	800	2400
12	1000	3000
13	3000	9000
14	5000	15000
15	8000	24000

At the end of each music string, hold down the SHIFT key, press the CLR/HOME key and enter the letter G. This causes the Release cycle (see Section 11.1.1) of the last note to be triggered.

The SUSTAIN parameter, s, is in the range 0 thru 15. This defines an intermediate volume level at which the sound will be held and is equivalent to changing the volume level set up by VOL (see Section 11.2.1) while the selected note is being played. Note, however, that this affects only the note selected. The volume of sounds played through other voices remains unaltered.

EXAMPLE: To create a sound envelope for the music played through voice 1:

ENTRY: 30 ENVELOPE 1,8,8,8,0

RESULT: All music notes following this command are played through voice 1 with equal rates for the Attack, Decay and Release cycles and an intermediate volume level set at 8.

11.2.4 MUSIC

FORMAT: MUSIC n,"music string"

or: MUSIC n,variable + variable + variable....

PURPOSE: To write music or create sound effects.

The MUSIC command allows you to compose and play music or create sound effects. The first command parameter refers to the duration of one music beat. This number must be in the range 1 thru 255 - 1 is the longest duration, 255 is the shortest. Following this parameter, you then define the string of musical notes you wish to play. Strings of notes may be added up to a maximum of 255 characters. The voice through which you wish to play the notes is specified at the beginning of the string. To do this, hold down the SHIFT key, press the CLR/HOME key (a reverse-field 'heart' is displayed) and enter the relevant voice number. Note that only ONE voice can be used in a string of notes.

Music notes are in the range A thru G. C is the first note in each octave, i.e the sequence of notes is C,D,E,F,G,A,B. A music sharp is defined by holding down the SHIFT key and pressing the letter of the relevant note. If you wish to play a note flat, you must sharpen the previous note, e.g. E flat would be D sharp and B flat would be entered as A sharp. Music rests are indicated by the letter Z.

The octave in which the note will be played is defined by a number from 0 thru 8. This number is entered AFTER the note. (Rests of course are entered without an octave number.) The duration of each note is specified by a control character following the octave number. This character is entered by pressing one of the four function keys. The function keys and associated note durations are shown in the table below:

FUNCTION KEY	NOTE DURATION
F1 KEY	One sixteenth of a beat
F3 KEY	One eighth of a beat
F5 KEY	One quarter of a beat
F7 KEY	Half a beat
F2 KEY	One beat
F4 KEY	Two beats
F6 KEY	Four beats
F8 KEY	Eight beats

After you have specified the duration of the last note in the string, hold down the SHIFT key, press the CLR/HOME key and enter the letter G. This causes the Release cycle (see Section 11.1.1) of each note to be triggered.

EXAMPLE: To compose a tune:

ENTRY:

```
40 A$ = "<SHIFT CLR/HOME>1Z<F1>C5<F1>E5<F1>F5<F1>"
50 A2$ = "G5<F7>C5<F1>E5<F1>F5<F1>G5<F7>C5<F1>E5
   <F1>F5<F1>G5<F3>E5<F3>C5<F3>E5<F3>D5<F5>E5<F1>
   E5<F1>D5<F1>C5<F7>C5<F1>"
60 A2$ = A2$ + "E5<F3>G5<F3>G5<F1>F5<F5>F5<F3>E5
   <F1>F5<F1>G5<F3>E5<F3>C5<F3>D5<F3>C5<F3>C5<F1>
   C5<F1>E5<F1>F5<F1>"
70 A3$ = "C5<F7>C5<F1><SHIFT CLR/HOME>G"
80 MUSIC 8,A$ + A2$ + A2$ + A3$
```

RESULT: When this section of the program is executed, the notes are stored in the variables A$, A2$ and A3$ and 'compounded' into a tune.

11.2.5 PLAY

FORMAT: PLAY n

PURPOSE: To play composed music.

The PLAY command, as its name suggests, allows you to play the music you have composed. The parameter following the command indicates how the music will be played in relation the rest of the program. A "0" in this position turns music off. A "1" plays the music and waits for it to end before proceeding with the program. A "2" plays the music and continues executing the program.

EXAMPLE: To play the music you have composed and continue program execution:

ENTRY:
```
90 PLAY 2
100 GOTO 100
```

ACTION: Type RUN <RETURN>

RESULT: 'When The Saints Go Marching In' is played.

EXAMPLE: To create a sound effect.

ENTRY:
```
10 VOL 15
20 WAVE 1,10000000
30 ENVELOPE 1,0,10,0,0
40 MUSIC 5,"<SHIFT CLR/HOME>1C5<F2>"
45 REPEAT
50 PLAY 1
55 A = A + 1: UNTIL A = 5
```

ACTION: Type RUN <RETURN>

RESULT: Five shots are fired.

Note that PLAY 2 CANNOT be used in conjunction with high-resolution or multi-colour graphics.

SECTION TWELVE
READ FUNCTIONS

12.1 INTRODUCTION

Section Twelve illustrates the four read functions supplied by the SIMONS' BASIC cartridge. If you have incorporated the use of a lightpen, joystick or paddle in your programs, these functions will enable you to determine the position of these devices.

NOTE

The light pen must be inserted ONLY into games port 1, i.e the port furthest away from the ON/OFF switch. A joystick can be inserted into either games port. (See your COMMODORE 64 User's Guide.)

The PENX and PENY commands allow you to determine the position on the screen at which a lightpen is pointing. The POT command reads the screen position of a paddle, while the JOY command allows you to determine in which direction a joystick is pointing.

12.2 PENX

FORMAT: variable = PENX

PURPOSE: To return the x coordinate of the light pen.

The PENX function returns the position of the light pen across the screen, i.e. from the left edge. The number returned is in the range 0 to 320. The example for this function is contained in the following section. Note that the PENX value must always be read before that of PENY.

12.3 PENY

FORMAT: variable = PENY

PURPOSE: To return the y coordinate of the light pen.

The PENX function returns the position of the light pen down the screen, i.e. from the top. The number returned is in the range 0 to 200.

EXAMPLE: To sketch on the screen with the lightpen and then print out the drawing:

ACTION: Enter the following program and then RUN it. Instructions for the program will be displayed on the screen.

```
10 REM:***LIGHT PEN PROGRAM***
20 HIRES0,1
30 TEXT10,10,"AFTER YOU HAVE TYPED IN 'RUN'",1,1,8
40 TEXT10,25,"YOU HAVE 15 SECONDS TO INSERT",1,1,8
50 TEXT10,40,"THE LIGHTPEN IN 'CONTROL PORT 1'",1,1,8
60 TEXT10,55,"WHEN YOUR DRAWING IS COMPLETE",1,1,8
70 TEXT10,70,"REMOVE LIGHTPEN FROM USER PORT",1,1,8
80 TEXT10,85,"AND PRESS SPACE-BAR FOR PRINT-OUT",1,1,8
90 PAUSE 15
100 HIRES0,1
110 LINE10,10,300,10,1:LINE300,10,300,180,1
120 LINE300,180,10,180,1:LINE10,180,10,10,1
130 GETA$:IFA$=""THEN130
140 IFA$=" "THEN160
150 PLOT (PENX+51)AND255,(PENY-50)AND255,1:GOTO 150
160 COPY:END
```

12.4 POT

FORMAT: variable = POT(0)

or: variable = POT(1)

PURPOSE: To return the resistance of a paddle.

The POT function enables you to determine how far a paddle control has been rotated. The number returned is in the range 0 thru 255. The number following the command defines the paddle whose value is to be read.

EXAMPLE: To move a sprite using the paddles:

ENTRY:
```
10 REM:****PADDLE PROGRAM****
20 REM:******BY K MORRIS*****
30 HIRES0,1
40 TEXT10,10,"BY MOVING THE PADDLES YOU CAN",1,1,8
50 TEXT10,25,"GET THE SPRITES TO MEET.",1,1,8
60 TEXT10,40,"WHY DON'T YOU TRY ?",1,1,8
65 TEXT10,55,"(SET PADDLE X=0 AND PADDLE Y=255)",1,1,8
70 DESIGN 0,64*32+49152
80 @.........................B
90 @.........................B
100 @.........................B
110 @...........BBBBBBBBBBBBB
120 @...........B............B
130 @...........B............B
140 @...........B............B
150 @...........B.......BBBBB
160 @...........B.......B...B
170 @...........B.......B...B
180 @.BBBBBBBBBBBBBBBBBBBBBBB
190 @...........B.......B...B
200 @...........B.......B...B
210 @...........B.......BBBBB
220 @...........B............B
230 @...........B............B
240 @...........B............B
250 @...........BBBBBBBBBBBBB
260 @.........................B
270 @.........................B
280 @.........................B
290 DESIGN 0,64*33+49152
300 @B.........................
310 @B.........................
```

```
320 @B.....................
330 @BBBBBBBBBBBBB.........
340 @B...........B.........
350 @B...........B.........
360 @B...........B.........
370 @BBBBB.......B.........
380 @B...B.......B.........
390 @B...B.......B.........
400 @BBBBBBBBBBBBBBBBBBBBB.
410 @B...B.......B.........
420 @B...B.......B.........
430 @BBBBB.......B.........
440 @B...........B.........
450 @B...........B.........
460 @B...........B.........
470 @BBBBBBBBBBBBB.........
480 @B.....................
490 @B.....................
500 @B.....................
510 MOB SET 0,32,2,0,0
520 MOB SET 1,33,2,0,0
530 Z=10:W=245
535 MMOB0,Z,170,Z,170,3,50
536 MMOB1,W,170,W,170,3,50
540 X=INT(POT(0))
550 FORL=ZTOX
555 IFL=100THEN615
560 RLOCMOB0,L,170,3,50
580 NEXTL
590 Z=X:GOTO540
615 Y=INT(POT(1))
620 FORB=WTOY
625 IFB=145THEN700
630 RLOCMOB1,B,170,3,50
640 NEXT B
650 W=Y:GOTO615
700 TEXT105,170,"FIRE",1,3,8:PAUSE1
703 MOB OFF 0:MOB OFF 1
705 TEXT105,170,"FIRE",0,3,8
710 TEXT75,145,"BOOM !!!",1,5,20
716 BFLASH10,0,7
720 VOL15
730 WAVE 1,10000000
740 ENVELOPE 1,3,0,15,0
750 A$="□1Z■G5■"
760 MUSIC 8,A$
770 PLAY 1
820 VOL0:BFLASH0:HIRES0,1:PRINT"□
```

12.5 JOY

FORMAT: variable = JOY

PURPOSE: To return the value associated with the position of a joystick.

The JOY function allows to you to test the direction in which a joystick is pointing or if the fire button is being held down. The values returned and the associated joystick positions are shown in Figure 12-1.

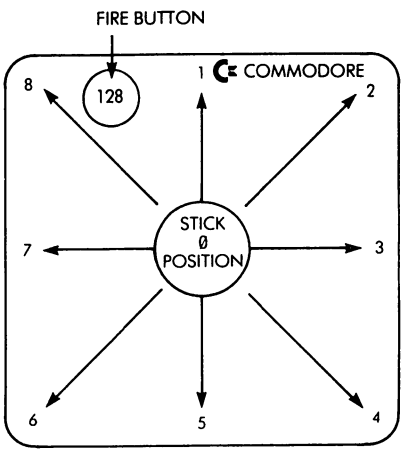


FIGURE 12-1 JOYSTICK VALUES

EXAMPLE: To draw a shape with the joystick and then paint it:

ACTION: Enter the following program and then RUN it. Instructions for the program will be displayed on the screen.

```
10 REM:********JOY STICK PROGRAM*****
11 HIRES0,1
12 LINE10,20,310,20,1:LINE310,20,310,160,1
13 LINE310,160,10,160,1:LINE10,160,10,20,1
15 TEXT88,30,"THE MOZOSKETCH",1,2,10
16 TEXT90,43,"──────────────",1,2,8
17 TEXT18,60,"DRAW A SHAPE MAKING SURE THAT",1,2,10
18 TEXT26,80,"THERE ARE NO GAPS,AND KEEP",1,2,10
19 TEXT60,100,"WITHIN THE BOUNDARIES.",1,2,10
20 TEXT105,140,"BY K J MORRIS",1,2,8:PAUSE 3
30 HIRES0,1
35 TEXT93,5,"THE MOZOSKETCH",1,1,8
40 TEXT4,18,"DRAW A SHAPE AND PRESS THE FIRE BUTTON",1,1,8
45 TEXT73,31,"TO SEE WHAT HAPPENS",1,1,8
50 LINE10,50,310,50,1:LINE310,50,310,150,1
55 LINE310,150,10,150,1:LINE10,150,10,50,1
80 X=160:Y=100
90 PLOTX,Y,1
100 IF JOY=1 THEN Y=Y-1:GOTO200
110 IF JOY=2 THEN Y=Y-1:X=X+1:GOTO200
120 IF JOY=3 THEN X=X+1:GOTO200
130 IF JOY=4 THEN X=X+1:Y=Y+1:GOTO200
140 IF JOY=5 THEN Y=Y+1:GOTO200
150 IF JOY=6 THEN Y=Y+1:X=X-1:GOTO200
160 IF JOY=7 THEN X=X-1:GOTO200
170 IF JOY=8 THEN X=X-1:Y=Y-1:GOTO200
180 IF JOY=128 THEN TEXT27,170,"WELL DONE PICASSO",1,3,16
190 PAUSE2:GOTO340
200 IFX<20THENX=20:GOTO 100
300 IFX>300THENX=300:GOTO100
310 IFY<60THENY=60:GOTO100
320 IFY>140THENY=140:GOTO100
330 PLOTX,Y,1:GOTO100
340 LOW COL 2,1,1
350 PAINTX+1,Y+1,1:PAUSE5
360 HIRES0,1
370 TEXT50,50,"PRESS SPACE BAR FOR ANOTHER GO",1,2,8
380 GET A$:IFA$=""THEN380
390 IFA$=" "THEN 30
400 GOTO380
```

SECTION THIRTEEN
EXAMPLES OF SIMONS' BASIC PROGRAMS

13.1 INTRODUCTION

Section Thirteen contains four programs to illustrate what may be achieved using the SIMONS' BASIC cartridge. Simply type each program in and then RUN it.

13.2 PROGRAM 1—DRAWING A POLYHEDRON

The following program draws a multi-sided figure at an ever decreasing size.

```
10 PRINT"□"
20 CENTRE "SIMONS BASIC POLYHEDRA":PRINT:PRINT
30 CENTRE "BY S BEATS":PRINT:PRINT
40 PRINT "NUMBER OF SIDES  ";:FETCH "#",2,N
50 EXEC SPIRAL
60 TEXT 10,10,"PRESS A KEY",1,1,8
70 GETA$:IFA$=""THEN70
80 CSET0:GOTO10
996 :
997 :
998 :
999 :
1000 PROC SPIRAL
1010 MP=100:HIRES 0,1
1020 FORJ = 0 TO 2.5*π STEPπ/20
1030 FORK = 0+J TO 2*π+J+.1STEP2*π/N
1040 X=INT(MP*1.3*SIN(K)+160)
1050 Y=INT(MP*COS(K)+100)
1060 IFK>0+J THEN LINE X1,Y1,X,Y,1
1070 X1=X:Y1=Y:NEXT
1080 MP=MP-2:NEXT
1090 END PROC
```

13.3 PROGRAM 2—WORDSEARCH

The program below allows you to enter up to 20 words of your own choice. It then mixes up all the words within a grid. Your task is to pinpoint the coordinates on the grid where each word begins. You are also given the option of printing out the grid so that you can play the game away from the computer.

```
10 REM ****************************
20 REM *                          *
30 REM *  WORDSEARCH              *
40 REM *                          *
50 REM *    BY STEVE BEATS        *
60 REM *                          *
70 REM ****************************
80 :
90 :
100 EXEC SETUP
110 EXEC GETWORDS
120 EXEC SCREEN
130 EXEC SORTLENGTHS
140 EXEC PLACEWORDS
150 EXEC PRINTGRID
160 EXEC GAME
170 CALL FINISH
180 :
181 :
3000 PROC GAME
3002 PRINT AT(28,1) "PRINT(Y/N)"
3004 GETQ$:IFQ$<>"Y"ANDQ$<>"N"THEN3004
3005 TI$="000000":TU=0
3006 PRINT AT(28,1) "
3008 IFQ$="N"THEN 3010
3009 HRDCPY
3010 WF=0:REPEAT
3011 PRINT AT(28,1) "ROW ";:FETCH "W",3,RO$
3012 IF TI$>"001000"THEN TU=1:END PROC
3020 PRINT AT(28,1) "COLUMN ";:FETCH "W",3,CO$
3030 PRINT AT(28,1) "                "
3040 RO=VAL(RO$):CO=VAL(CO$)
3050 IF RO>0ANDRO<21ANDCO>0ANDCO<21THEN3070
3060 PRINT AT(28,1) "ERROR":PAUSE 5000:PRINT AT(28,1) "                ":GOTO3011
3070 F=0:FORI=1TONW:IFRO=PY(I)ANDCO=PX(I)THENF=1:XI=I
3080 NEXT:IFF=1THEN3100
3090 PRINT AT(28,1) "WRONG":PAUSE5000:PRINT AT(28,1) "                ":GOTO3011
3100 FORI=0TOLEN(W$(XI))-1
3110 PRINT AT(3+PX(XI)+I*DX(TW(XI)),2+PY(XI)+I*DY(TW(XI))) "■";
3115 PRINTMID$(W$(XI),I+1,1)
3120 NEXT:PRINT AT(25,2+XI) W$(XI):WF=WF+1
3130 UNTIL WF=NW
```

```
3140 END PROC
3150 :
3160 :
4000 PROC PRINTGRID
4010 PRINT"■": FORY=1TO20:FORX=1TO20
4020 IFA$(X,Y)=""THENA$(X,Y)=MID$(W$(NW*RND(1)+1),5*RND(1)+1,1)
4040 PRINT AT(3+X,2+Y) A$(X,Y)
4050 NEXT:NEXT
4060 END PROC
4070 :
4080 :
5000 PROC PLACEWORDS
5010 PW=0:REPEAT:PW=PW+1
5020 PX(PW)=INT(20*RND(1)+1)
5030 PY(PW)=INT(20*RND(1)+1)
5040 DR=INT(8*RND(1)+1):TW(PW)=DR
5041 CX=PX(PW)+LEN(W$(PW))*DX(DR):CY=PY(PW)+LEN(W$(PW))*DY(DR)
5050 IFCX<1ORCX>20ORCY<1ORCY>20THEN5020
5051 REM IT FITS THE GRID SO CHECK LETTERS
5060 F=0:FORCK=0TOLEN(W$(PW))-1
5070 Z1$=MID$(W$(PW),CK+1,1):Z2$=A$(PX(PW)+CK*DX(DR),PY(PW)+CK*DY(DR))
5080 IFZ2$<>""ANDZ1$<>Z2$THENF=1
5090 NEXT:IFF=1THEN5020
5091 REM IT FITS SO SLOT IT IN
5100 FORCK=0TOLEN(W$(PW))-1
5110 Z1$=MID$(W$(PW),CK+1,1):A$(PX(PW)+CK*DX(DR),PY(PW)+CK*DY(DR))=Z1$
5120 PRINT AT(25,2+PW) W$(PW):NEXT
5130 UNTIL PW=NW
5140 END PROC
5150 :
5160 :
6000 PROC SORTLENGTHS
6010 PRINT AT(25,1) "■SORTING"
6020 F=0:FORI=1TONW-1
6030 IFLEN(W$(I))<LEN(W$(I+1))THENT$=W$(I+1):W$(I+1)=W$(I):W$(I)=T$:F=1
6040 NEXT:IFF=1THEN6020
6050 PRINT AT(25,1) "■         "
6060 END PROC
6070 :
6080 :
7000 PROC SCREEN
7010 PRINT"[redacted]";:COLOUR 5,0
7020 PRINT"╱";:FORI=1TO20:PRINT"─";:NEXT
7030 PRINT"╲[redacted]";:FORI=1TO20:PRINT"▌[redacted]";:NEXT
7040 PRINT"╱[redacted]";:FORI=1TO20:PRINT"▀[redacted]";:NEXT
7050 PRINT"╲[redacted]";:FORI=1TO20:PRINT"▌[redacted]";:NEXT
7060 FORI=1TO20:NU$=STR$(I):NU$=MID$(NU$,2):IFLEN(NU$)=1THENNU$=" "+NU$
7070 PRINT AT(1,I+2) NU$:PRINT AT(I+3,0) LEFT$(NU$,1)
7080 PRINT AT(I+3,1) RIGHT$(NU$,1):NEXT
7090 END PROC
7100 :
```

```
7110 :
8000 PROC GETWORDS
8010 COLOUR10,0:PRINT"■■■■HOW MANY WORDS (MAX 20)■"
8020 FETCH "■",3,NW$
8030 PRINT:NW=VAL(NW$):IFNW<1ORNW>20THENS010
8040 PRINT"■          NOW TYPE IN THE WORDS■"
8050 PRINT" THEY MUST BE BETWEEN 5 AND 15 LETTERS■"
8060 FORI=1TONW
8070 PRINTI"■) ",
8080 FETCH "■",15,W$(I)
8090 IFLEN(W$(I))<5THENPRINTCHR$(13)"■■":GOTO8070
8100 PRINT:NEXT
8110 END PROC
8998 :
8999 :
9000 PROC SETUP
9010 COLOUR 5,0:PRINT"■■SETTING UP, PLEASE WAIT.....
9020 DIMA$(20,20),W$(20),PX(20),PY(20),DX(8),DY(8),TW(20)
9030 FORI=1TO20:FORJ=1TO20:A$(I,J)="":NEXT:NEXT
9040 RESTORE:FORI=1TO8:READDX(I),DY(I):NEXT
9050 DATA 0,-1,1,-1,1,0,1,1,0,1,-1,1,-1,0,-1,-1
9060 FORI=1TO20:W$(I)="":PX(I)=0:PY(I)=0:NEXT
9070 END PROC
9998 :
9999 :
10000 PROC FINISH
10010 IFTU=1THEN10100
10020 PRINT AT(20,1) "AGAIN(Y/N)"
10030 GET A$:IFA$<>"Y"ANDA$<>"N"THEN10030
10040 IF A$="Y"THENCLR:RUN
10050 PRINT"■":END
10100 FORX1=1TONW
10110 FORI=0TOLEN(W$(X1))-1
10120 PRINT AT(3+PX(X1)+I*DX(TW(X1)),2+PY(X1)+I*DY(TW(X1))) "■■";
10130 PRINTMID$(W$(X1),I+1,1)
10140 NEXT:PRINT AT(25,2+X1) W$(X1):NEXT
10150 GOTO10020
```

13.4 PROGRAM 3—LETTER SLIDER

This program mixes the letters A thru O within a 4 by 4 square. There is one vacant space in the square. You must rearrange the letters into alphabetical order by sliding letters around the square.

```
1 REM ***** LETTER SLIDER GAME *****
2 REM *****                   *****
3 REM *****   BY STEVE BEATS  *****
4 :
5 EXEC INSTRUCTIONS
10 DIMA$(4,4),B$(4,4):MN=1
20 S$="
30 C$="
40 L$="ABCDEFGHIJKLMNO "
50 PT=1:FORY=1TO4:FORX=1TO4
60 I1$=MID$(C$,PT,1):I2$=MID$(L$,PT,1):PT=PT+1
70 W$= INST(I1$,S$,1)
80 W$= INST(I2$,W$,10)
90 A$(X,Y)=W$
100 NEXT:NEXT
110 A$(4,4)="
120 PT=1:FORY=1TO4:FORX=1TO4
130 B$(X,Y)=MID$(L$,PT,1):PT=PT+1
140 NEXT:NEXT:PX=4:PY=4
150 PRINT"
160 PRINT AT(12,5) "
170 FORI=1TO12:PRINT"
180 FORI=1TO12:PRINT"
190 FORI=1TO12:PRINT"
200 EXEC LETTERS
205 EXEC SHUFFLE
206 REPEAT
210 GETR$:IFR$<"A"ORR$>"O"THEN210
215 PRINT AT(12,3) "
220 EXEC CHECKIT
230 PRINT AT(12,3) MS$
240 EXEC FINISHED
250 UNTIL WI=1
260 RUN 10
998 :
999 :
1000 PROC LETTERS
1010 FORY=1TO4:FORX=1TO4
1020 PRINT AT(3*X+10,3*Y+3) A$(X,Y)
1030 NEXT:NEXT
1040 END PROC
1050 :
1060 :
2000 PROC SHUFFLE
2010 RESTORE:FORI=1TO4:READXD(I),YD(I):NEXT
2015 PRINT AT(12,3) "SHUFFLING...."
2020 DATA -1,0,0,1,1,0,0,-1
2025 I=0:REPEAT:I=I+1
2030 D=INT(8*RND(RND(1))+1)
```

```
2040 IFPX+XD(D)<1ORPX+XD(D)>4ORPY+YD(D)<1ORPY+YD(D)>4THEN2030
2050 X1=PX+XD(D):Y1=PY+YD(D):IFX2=X1ANDY2=Y1THEN2030
2060 T$=A$(X1,Y1):A$(X1,Y1)=A$(PX,PY):A$(PX,PY)=T$
2070 T$=B$(X1,Y1):B$(X1,Y1)=B$(PX,PY):B$(PX,PY)=T$
2080 X2=PX:Y2=PY:PX=X1:PY=Y1
2090 EXEC LETTERS
2100 UNTIL I>100
2110 PRINT AT(12,3) "
2120 END PROC
2130 :
2140 :
3000 PROC CHECKIT
3005 PRINT AT(1,1) "■MOVE NUMBER"MN
3010 FL=0:FORI=1TO4
3020 IFPX+XD(I)<1ORPX+XD(I)>4ORPY+YD(I)<1ORPY+YD(I)>4THEN3040
3030 IFB$(PX+XD(I),PY+YD(I))=R$THENFL=1:TE=I
3040 NEXT:IFFL=0THENMS$="ILLEGAL MOVE !!!":GOTO3100
3050 X1=PX+XD(TE):Y1=PY+YD(TE)
3060 T$=A$(X1,Y1):A$(X1,Y1)=A$(PX,PY):A$(PX,PY)=T$
3070 T$=B$(X1,Y1):B$(X1,Y1)=B$(PX,PY):B$(PX,PY)=T$
3080 EXEC LETTERS
3090 MS$="MOVE OK :":PX=X1:PY=Y1:MN=MN+1
3100 END PROC
3110 :
3120 :
4000 PROC FINISHED
4010 FI$="":FORY=1TO4:FORX=1TO4
4020 FI$=FI$+B$(X,Y):NEXT:NEXT
4030 IFFI$=L$THEN4050
4040 END PROC
4050 PRINT AT(12,3) "A WINNER !!!":PAUSE10:WI=1:GOTO4040
4060 :
4070 :
5000 PROC INSTRUCTIONS
5005 COLOUR10,0
5010 PRINT"▲DO YOU REQUIRE INSTRUCTIONS (Y/N)?"
5020 GETA$:IFA$<>"Y"ANDA$<>"N"THEN5020
5030 IF A$="N"THEN END PROC
5040 PRINT"■■■    THE OBJECT OF THE GAME IS TO GET
5050 PRINT"■    ALL OF THE LETTERS IN THE CORRECT"
5060 PRINT"■    ORDER....."
5070 PRINT"■■                  A B C D "
5080 PRINT"■                   E F G H "
5090 PRINT"■                   I J K L "
5100 PRINT"■                   M N O ■ "
5110 PRINT"■■■    TO MOVE A LETTER INTO THE VACANT"
5120 PRINT"■    SPACE, JUST TYPE THAT LETTER ON"
5130 PRINT"■    THE KEYBOARD."
5140 PRINT"■■                  TRY IT NOW"
5150 GET A$:IFA$=""THEN5150
5160 END PROC.
```

13.4 PROGRAM 4—A VINTAGE CAR

This program draws a vintage car on a multi colour graphics screen.

```
1 COLOUR10,5
10 HIRES 0,1:MULTI 0,2,1
20 CIRCLE 30,150,10,11,1
30 CIRCLE 30,150,13,15,1
31 PAINT 28,150,2
32 PAINT 19,150,1
35 ARC 30,150,270,90,10,17,21,3
36 LINE 15,150,17,130,3
37 LINE 43,150,45,150,3
38 PAINT 16,148,3
40 BLOCK 3,105,13,150,1
41 LINE 3,105,6,105,0
42 LINE 3,105,3,107,0:LINE 10,105,13,105,0:LINE13,105,13,107,0
43 LINE 3,150,6,150,0
44 LINE 3,148,3,150,0
45 LINE 3,115,0,123,1
46 LINE 3,140,0,125,1
50 LINE 16,139,16,105,1
60 LINE 16,105,30,105,1
70 LINE 30,105,30,129,1
71 PAINT 18,107,1
80 LINE 30,105,78,105,2
90 LINE 70,105,70,145,2
91 LINE 70,145,47,145,2
92 PAINT 33,107,2
93 LINE 43,152,56,152,1
94 BLOCK 56,152,74,154,1
95 LINE 74,152,80,152,1
96 LINE 80,152,80,144,1
98 BLOCK 72,105,94,145,2:PLOT 71,105,1
99 PAINT 75,151,1
100 BLOCK 82,146,108,153,1
101 BLOCK 110,144,120,152,1
102 BLOCK 95,105,108,146,1
103 BLOCK 108,105,114,141,1
105 BLOCK 31,70,36,104,1
106 LINE 31,70,95,70,1
```

APPENDIX A
ERROR MESSAGES

In the course of using SIMONS' BASIC, an error message may appear. These messages are unique to the SIMONS' BASIC cartridge. Each error message, its meaning and probable cause is given in this appendix.

? BAD MODE

This occurs when any parameter in a command is outside the range allowed.

? NOT HEX CHARACTER

An attempt has been made to convert a non-hexadecimal number into its decimal equivalent.

? NOT BINARY CHARACTER

An attempt has been made to convert a non-binary number into its decimal equivalent.

? UNTIL WITHOUT REPEAT

The UNTIL command has been used without any previously declared REPEAT.

? END LOOP WITHOUT LOOP

The END LOOP command has been used without any previously declared LOOP.

? END PROC WITHOUT EXEC

The END PROC command has been used without any procedure having been executed.

? PROC NOT FOUND

An attempt has been made to select a procedure that does not exist.

? NOT ENOUGH LINES

Not enough lines have been set up for a MOB design grid.

? BAD CHAR FOR A MOB IN LINE n

A parameter within the MOB design stage is outside the range defined. The line number of the error is always that where the DESIGN command was executed although this does not necessarily mean that the fault is in that line.

? STACK TOO LARGE

This occurs if you have nested more than nine procedures or program loops.

GLOSSARY

A list of terms used in this manual.

BIT
Abbreviation for 'Binary Digit'. The smallest unit of computer memory.

BRANCH
The transfer of program execution from one line to another.

COORDINATE
The distance of a point on a grid from the x or y axes.

DATA
Information held in the memory of the computer or on a storage device.

DEBUGGING
Correcting programming mistakes.

FLOATING POINT
A system that holds numbers in exponential form.

INTEGER
A whole number.

KERNAL
The operating system of the COMMODORE 64.

LIBRARY
A stock of programs and/or program sub-routines.

MOB
Moveable object block. A term used to describe a sprite.

OCTAVE
A group of eight musical notes.

ORIGIN
The point on the screen from which a shape is drawn.

PERIPHERAL
An external device which is connected to the computer.

PIXEL
The smallest addressable location on the screen.

PROGRAM CRASH
An unwanted halt in program execution.

REGISTER
A reserved area within the computer's memory.

SECONDARY ADDRESS
A program/file storage instruction.

SCROLLING
Moving across the screen in a vertical or horizontal direction.

SPRITE
A programmable object.

START ADDRESS
The point from which a block of data is stored within the computer's memory.

TIME CYCLE
The duration of a frequency component measured in thousandths of a second.

INDEX

Commodore Italiana SPA
Via Fratelli Gracchi 48,
Cinisello Balsamo, Milano, Italy.

Commodore Computer BV
Marksingel, 2e4811 NV Breda. Postbus 720,
4803aS Breda, Netherlands.

Commodore Business Machines Ltd.,
3370, Pharmacy Avenue, Agincourt,
Ontario, M1W 2K4, Canada.

Commodore A.G. Schweiz,
Aeschenvorstadt 57, 4010,
Basel, Switzerland.

Commodore Business Machines Inc.,
1200, Wilson Drive, West Chester,
PA 19380, USA.

Commodore Buromaschinen GmbH,
Lyoner Str. 38, Postfach 710126,
6000 Frankfurt. West Germany.

Commodore Business Machines Pty. Ltd.,
5, Orion Road, Lane Cove.
New South Wales 2066, Australia.

Commodore Business Machines (UK) Ltd.,
675 Ajax Avenue, Slough Trading Estate,
Slough, Berks. SL1 4BG England.

Printed in Hong Kong

commodore COMPUTER