

SIMONS' BASIC EXTENSION

91 ADDITIONAL PROGRAMING COMMANDS



commodore
COMPUTER

**SIMONS' BASIC EXTENSION
USER GUIDE**

SBX 6440/20

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

SIMONS' BASIC EXTENSION USER GUIDE

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

KENTCHASE LTD., SLOUGH

without compositor intervention. The manual was printed and bound by

CARTER LITHO, Slough

Special thanks to Gail Wellington and Marilyn Rutley who helped with the preparation of this manual and Mark Palmer for his help with the example programs.

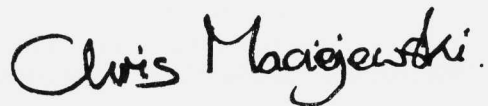
COMMENTS AND ERRATA REQUEST

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,

A handwritten signature in black ink that reads "Chris Maciejewski". The signature is written in a cursive style with a large initial 'C'.

Chris Maciejewski.

Technical Author

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

COPYRIGHT — SOFTWARE PRODUCT

This software product is copyrighted and all rights are reserved by:

D. S. Software
19 Reddings
Welwyn Garden City
Hertfordshire
AL8 7LA

The distribution and sale of this product are intended for the original purchaser only. Lawful users of these programs are hereby licensed only to read these programs from the medium into the memory of a computer solely for the purpose of executing the programs. Security copies of the programs may be made only for their own use. Duplicating for any other purpose, copying, selling or otherwise distributing this product is a violation of the law.

COPYRIGHT — MANUAL

This manual is copyrighted and all rights are reserved. This document may not, in whole or in part be copied, photocopied, reprinted, translated, reduced to any electronic medium or machine readable form or reproduced in any manner without prior consent in writing from COMMODORE BUSINESS MACHINES, LTD., Technical Publications Manager.

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 EXTENSION

1.1	INTRODUCTION	1-1
1.2	THE SIMONS' BASIC EXTENSION MANUAL	1-1
1.3	LOADING SIMONS' BASIC EXTENSION	1-3
1.3.1	FROM DISKETTE	1-3
1.3.2	FROM CASSETTE	1-3
1.4	SIMONS' BASIC EXTENSION COMMANDS	1-4
1.5	ENTERING COMMANDS	1-5
1.6	CONVENTIONS	1-5

SECTION TWO - PROGRAMMING AIDS

2.1	INTRODUCTION	2-1
2.2	RENUMBER	2-1
2.3	DELETE	2-2
2.4	ALTER	2-3
2.5	HELP	2-4
2.6	CHAIN	2-5
2.7	PROTECT	2-6
2.8	FORCE	2-7
2.9	DS\$	2-8

SECTION THREE - ARRAY MANIPULATION

3.1	INTRODUCTION	3-1
3.1.1	WHAT IS AN ARRAY?	3-1
3.1.2	MULTI-DIMENSION ARRAYS	3-1
3.1.3	MANIPULATING ARRAYS	3-1
3.2	ARRAY COMMANDS	3-3
3.2.1	CONVENTIONS	3-3
3.2.2	SET ARR	3-3
3.2.3	PRINT ARR	3-4
3.2.4	ZER ARR	3-5
3.2.5	ADD ARR	3-5
3.2.6	SUB ARR	3-6
3.2.7	MUL ARR	3-7
3.2.8	DIV ARR	3-8
3.2.9	ADD ALL	3-9
3.2.10	SUB ALL	3-9
3.2.11	MUL ALL	3-10
3.2.12	DIV ALL	3-10
3.2.13	COPY ARR	3-11
3.2.14	INPUT ARR	3-11
3.2.15	READ ARR	3-12
3.2.16	SCRATCH	3-12

3.2.17	SUM	3-13
3.2.18	ELEMENTS	3-14
3.2.19	MIN.....	3-14
3.2.20	MAX	3-15
3.3	STRING ARRAY MANIPULATION	3-16
3.3.1	SORT	3-16

SECTION FOUR - NUMERIC AIDS

4.1	INTRODUCTION	4-1
4.2	ADDITIONAL TRIGONOMETRICAL FUNCTIONS	4-1
4.2.1	GRAD.....	4-1
4.2.2	DEG	4-1
4.3	NUMERIC CONVERSION	4-2
4.3.1	BIN\$ - DECIMAL TO BINARY CONVERSION	4-2
4.3.2	HEX\$ - DECIMAL TO HEXADECIMAL CONVERSION	4-2
4.4	SPECIAL CALCULATIONS.....	4-3
4.4.1	CALCX.....	4-3
4.4.2	CALCY.....	4-4
4.5	EVALUATING A STRING AS A BASIC EXPRESSION.....	4-5
4.5.1	EVAL	4-5
4.5.2	BACK.....	4-6

SECTION FIVE - MEMORY MANIPULATION COMMANDS

5.1	INTRODUCTION	5-1
5.2	DOKE.....	5-1
5.3	DEEK	5-2
5.4	HIMEM.....	5-2
5.5	LOMEM	5-3
5.6	SCREEN	5-3

SECTION SIX - HIGH-RESOLUTION AND MULTI-COLOUR GRAPHICS

6.1	INTRODUCTION	6-1
6.2	HIGH-RESOLUTION AND MULTI-COLOUR GRAPHICS COMMANDS	6-1
6.2.1	GRID	6-1
6.2.2	TICK.....	6-2
6.2.3	HSAVE.....	6-3
6.2.4	HLOAD	6-4
6.2.5	LABEL	6-5
6.2.6	DRAW TO.....	6-6
6.2.7	VLIN	6-7
6.2.8	HLIN.....	6-7
6.3	SCALING FUNCTIONS	6-8
6.3.1	SCALE.....	6-8
6.3.2	SCX	6-9
6.3.3	SCY	6-10

SECTION SEVEN - LOW-RESOLUTION GRAPHICS COMMANDS

7.1	INTRODUCTION	7-1
7.2	LOW-RESOLUTION GRAPHICS COMMANDS	7-1
7.2.1	LOW RES	7-1
7.2.2	PREPARE	7-3
7.2.3	PUSH	7-3
7.2.4	PULL	7-4
7.3	SCREEN FUNCTIONS	7-4
7.3.1	FONT	7-4
7.3.2	UPPER	7-5
7.3.3	ECOL	7-6
7.3.4	CCOL	7-7
7.3.5	SCOL	7-7
7.3.6	SCHR	7-8
7.3.7	MCOL	7-8
7.3.8	DESIGN	7-9
7.3.9	ROTATE	7-10
7.3.10	BCKFLASH	7-11
7.3.11	★	7-12

SECTION EIGHT - SPRITE MANIPULATION COMMANDS

8.1	INTRODUCTION	8-1
8.2	SPRITES	8-1
8.2.1	INTRODUCTION	8-1
8.2.2	WHAT IS A VECTOR DRIVEN SPRITE?	8-1
8.2.3	WHAT IS A FRAME?	8-2
8.3	SPRITE MANIPULATION COMMANDS	8-2
8.3.1	SPRITE	8-2
8.3.2	XVEC	8-5
8.3.3	YVEC	8-6
8.3.4	BARRIER	8-6
8.3.5	CHANGE	8-7
8.3.6	START	8-7
8.3.7	CLEAR	8-8
8.3.8	INIT	8-9
8.3.9	NORMAL	8-9
8.3.10	INFO	8-10
8.3.11	SPRX	8-11
8.3.12	SPRY	8-12
8.3.13	SPR LOC	8-12
8.3.14	INVERT	8-13
8.3.15	REVERSE	8-15
8.3.16	SHOW	8-15
8.3.17	ON DETECT	8-17
8.3.18	CONTINUE	8-18
8.3.19	TRANSFER	8-19
8.3.12	CREATE	8-20
8.4	NOTES ON VECTOR DRIVEN SPRITES	8-20

SECTION NINE - MUSIC COMMANDS

9.1	INTRODUCTION	9-1
9.2	WHAT IS A FILTER?.....	9-1
9.3	MUSIC COMMANDS	9-2
9.3.1	FILTER	9-2
9.3.2	MODE	9-2
9.3.3	PULSE	9-3
9.3.4	BEEP	9-4

SECTION TEN - EXAMPLE PROGRAMS

10.1	INTRODUCTION	10-1
10.2	PROGRAM 1 - GRAPH PLOTTER	10-1
10.3	PROGRAM 2 - DOGGY	10-3
10.4	PROGRAM 3 - ROAD RACER	10-5

APPENDIX A - ERROR MESSAGES

GLOSSARY

INDEX

SECTION ONE

INTRODUCTION TO SIMONS' BASIC EXTENSION

1.1 INTRODUCTION

SIMONS' BASIC EXTENSION contains an additional 91 commands to supplement those supplied by the SIMONS' BASIC cartridge. These additional commands fall into seven categories as detailed below:

PROGRAMMING AIDS, such as CHAIN and ALTER, to assist in the entry and debugging of your BASIC programs.

ARRAY MANIPULATION commands, such as SET ARR, ADD ARR and SCRATCH to enable you to perform calculations on or between the information held in numeric arrays.

Extra NUMERIC AIDS, like DEG, to provide more flexibility in mathematical calculations.

MEMORY MANIPULATION commands, like HIMEM and LOMEM to allow you to set memory limits quickly and easily.

GRAPHICS commands, such as LABEL and ROTATE, to facilitate the design and presentation of graphics displays.

Extra SPRITE MANIPULATION commands, like INVERSE and CREATE, to provide you with even greater control over sprite creation and animation.

Additional MUSIC commands, such as FILTER and PULSE, to provide a simple way of utilising the extensive sound and music capabilities of the COMMODORE 64.

The commands provided by SIMONS' BASIC EXTENSION, used in conjunction with those supplied by the SIMONS' BASIC cartridge, allow even the inexperienced BASIC programmer to utilise fully the power of the COMMODORE 64.

1.2 THE SIMONS' BASIC EXTENSION MANUAL

This manual is divided into ten sections as outlined below:

SECTION ONE - INTRODUCTION TO SIMONS' BASIC EXTENSION

This section discusses the commands provided by SIMONS' BASIC EXTENSION (SBX) in broad terms. It explains how to load SBX either from diskette or cassette and how to enter a SIMONS' BASIC EXTENSION command. Also included are the conventions used in this manual to describe each command.

SECTION TWO - PROGRAMMING AIDS

Section Two contains commands, such as CHAIN and DELETE to speed up the entry and de-bugging of your BASIC programs.

SECTION THREE - ARRAY MANIPULATION

Here commands such as SET ARR and ADD ARR are explained. These enable you to carry out arithmetic operations on or between single or multi-dimensional numeric arrays.

SECTION FOUR - NUMERIC AIDS

Section Four contains commands like BIN\$ and HEX\$ which are used respectively to convert a decimal number into its binary or hexadecimal equivalent. Also included are the DEG and GRAD functions. The first converts degrees into rads while the second converts degrees into rads.

SECTION FIVE - MEMORY MANIPULATION COMMANDS

This section includes the commands DEEK and DOKE, which allow you to respectively read or assign a 16 bit number in memory, and the commands HIMEM and LOMEM which enable BASIC memory parameters to be read and/or altered.

SECTION SIX - HIGH-RESOLUTION AND MULTI-COLOUR GRAPHICS

Section Six contains commands, such as GRID, VLIN and MCOL, to enable you to draw shapes on a graphics screen.

SECTION SEVEN - LOW-RESOLUTION COMMANDS

This section contains commands for use when drawing on a normal, low-resolution, screen.

SECTION EIGHT - SPRITE MANIPULATION COMMANDS

Section Eight is concerned with the creation and movement of Sprite graphics.

SECTION NINE - MUSIC COMMANDS

Here, four additional music commands, FILTER, MODE, PULSE, and BEEP are discussed.

SECTION TEN - EXAMPLE PROGRAMS

Section Ten contains three example programs to demonstrate some of what may be achieved when using both SIMONS' BASIC and SIMONS' BASIC EXTENSION commands.

APPENDIX A - ERROR MESSAGES

A list of the SIMONS' BASIC EXTENSION error messages that you may encounter 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 LOADING SIMONS' BASIC EXTENSION

WARNING
SIMONS' BASIC EXTENSION CAN ONLY BE
LOADED WITH THE SIMONS' BASIC
CARTRIDGE IN PLACE.

1.3.1 FROM DISKETTE

Make sure that your computer is switched off. Insert the SIMONS' BASIC cartridge, label uppermost, into the cartridge slot at the rear of your computer (for more information on starting the SIMONS' BASIC cartridge see Section 1.3 of your SIMONS' BASIC MANUAL). Switch on the computer, disk drive, and TV or monitor. Insert the SIMONS' BASIC EXTENSION disk into the disk drive with the label uppermost, and towards you. Type:

LOAD ":",8,1

and press RETURN. After a short while the SIMONS' BASIC II title page appears. Once the program is loaded, the following message is displayed:

```
*** SIMONS BASIC 2 ***
(C)1984. ALL RIGHTS RESERVED
28287 BASIC BYTES FREE
```

1.3.2 FROM CASSETTE

Make sure that your computer is switched off. Insert the SIMONS' BASIC cartridge, label uppermost, into the cartridge slot at the rear of your computer (for more information on starting the SIMONS' BASIC cartridge see Section 1.3 of your SIMONS' BASIC MANUAL). Insert the SIMONS' BASIC EXTENSION cassette into the cassette unit. Switch on the computer, and TV or monitor. Type:

LOAD

and press RETURN. After a short while the SIMONS' BASIC II title page appears. Once the program is loaded, the following message is displayed:

```
*** SIMONS BASIC 2 ***
1983 (C)1984. ALL RIGHTS RESERVED
28031 28287 BASIC BYTES FREE
```

All the SIMONS' BASIC EXTENSION commands are now included in the operating system of your COMMODORE 64, together with those supplied by the SIMONS' BASIC cartridge and may be used like any other BASIC command. Note that the combination of SIMONS' BASIC with SIMONS' BASIC EXTENSION uses approximately 10K of the COMMODORE 64's memory, so you are left with 28K free for your programs.

1.4 SIMONS' BASIC EXTENSION COMMANDS

The following is a list of those commands supplied by SIMONS' BASIC EXTENSION:

Commands for entering and debugging programs:

RENUMBER, DELETE, ALTER, HELP, CHAIN, PROTECT, FORCE, DS\$.

Commands for manipulation of numeric arrays:

SET ARR, ZER ARR, PRINT ARR, ADD ARR, SUB ARR, MUL ARR, DIV ARR, ADD ALL, SUB ALL, MUL ALL, DIV ALL, COPY ARR, INPUT ARR, READ ARR, ELEMENTS, MIN, MAX, SCRATCH, SUM, SORT.

Commands for numeric conversion and bit manipulation:

GRAD, DEG, BIN\$, HEX\$, CALCX, CALCY, EVAL, BACK.

Commands for memory manipulation and configuration:

DEEK, DOKE, HIMEM, LOMEM, SCREEN.

Commands for graphics plotting and screen handling:

GRID, TICK, LABEL, DRAW TO, VLIN, HLIN, SCALE, SCX, SCY, LOW RES, DESIGN, MCOL, ROTATE, FONT, UPPER, ECOL, BCKFLASH, CCOL, SCOL, SCHR,★.

Commands for storing screen data in memory or on an external storage device:

PREPARE, PUSH, PULL, HSAVE, HLOAD.

Commands for generating/animating Sprites:

INIT, SPRITE, CREATE, TRANSFER, INVERT, REVERSE, CHANGE, BARRIER, INFO, START, CLEAR, SPRX, SPRY, SPR LOC, XVEC, YVEC, ON DETECT, CONTINUE, SHOW, NORMAL.

Commands for music synthesis:

FILTER, MODE, PULSE, BEEP.

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, or in the section in which the command appears.

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 of your SIMONS' BASIC MANUAL), all SIMONS' BASIC keywords must be separated from the first parameter of the command with a space.

COMMANDO : 'COLD' SCHAKELT EXT. OOK UIT



SECTION TWO

PROGRAMMING AIDS

2.1 INTRODUCTION

SIMONS' BASIC EXTENSION contains additional commands to assist you when entering and debugging BASIC programs.

The RENUMBER command renumbers a program listing including all GOTOs and GOSUBs. DELETE allows you to erase a specified section of a BASIC program. The ALTER command permits you to replace one character string or command with another. The HELP command displays the program line where an error occurred and highlights the position of the error within the line. CHAIN permits you to load one program from within another without having to concern yourself with resetting memory pointers.

NOTE

With the exception of the CHAIN command, all the commands in this section can ONLY be used in direct mode, NOT as part of a program.

2.2 RENUMBER

FORMAT: RENUMBER start line,increment

PURPOSE: To renumber a BASIC program.

The RENUMBER command allows you to renumber a BASIC program including all GOTOs and GOSUBs. The first parameter in the command is the start line number of the renumbered program. The second parameter is the interval between line numbers. Using RENUMBER with no parameters renumbers the program starting at line 1000 in increments of 10.

EXAMPLE: To renumber the following program starting at line 100 in increments of 25:

ENTER: 10 GET A\$: IF A\$ = "" THEN 10
 20 IF A\$ = CHR\$(13) THEN GOSUB 40
 30 GOTO 10
 40 PRINT "YOU PRESSED RETURN"
 50 FOR X = 1 TO 2000: NEXT
 60 RETURN

COMMAND: RENUMBER 100,25
TYPE: LIST <RETURN>
DISPLAY: 100 GET A\$: IF A\$ = "" THEN 100
125 IF A\$ = CHR\$(13) THEN GOSUB 175
150 GOTO 100
175 PRINT "YOU PRESSED RETURN"
200 FOR X = 1 TO 2000: NEXT
225 RETURN

2.3 DELETE

FORMAT: DELETE start line number - end line number
PURPOSE: To delete program lines from the memory of the COMMODORE 64.

The DELETE command operates on a specified line range, in the same way as the BASIC command LIST. The various formats of the DELETE command are listed below:

DELETE In	Deletes line In only
DELETE sln - fln	Deletes all lines between sln and fln inclusive
DELETE -fln	Deletes all lines from the start of the program to fln inclusive
DELETE sln-	Deletes all lines from line sln to the end of the program inclusive.

NOTE

The parameter In represents a single line number, sln is the start line number, and fln is the finish line number.

EXAMPLE: To delete the first two lines from the following program:

ENTER: 10 REM DELETE COMMAND
20 REM ONE OF THE MANY
30 REM USEFUL SIMONS' BASIC
40 REM AND SIMONS' BASIC
50 REM EXTENSION COMMANDS

COMMAND: DELETE-20 <RETURN>

TYPE: LIST <RETURN>

DISPLAY: 30 REM USEFUL SIMONS' BASIC
 40 REM AND SIMONS' BASIC
 50 REM EXTENSION COMMANDS

RESULT: Lines 10 and 20 have been removed from the program.

EXAMPLE: To delete lines 30 and 40 of the remaining program:

COMMAND: DELETE 30-40 <RETURN>

TYPE: LIST <RETURN>

DISPLAY: 50 REM EXTENSION COMMANDS

RESULT: Lines 30 and 40 have been removed from the program.

2.4 ALTER

FORMAT: ALTER old code&new code

or: ALTER "old string"&"new string"

PURPOSE: To search for an existing code or character string and replace it with a new code or character string.

If the character string to be ALTERed is enclosed in quotation marks, ALTER only replaces the matching character strings which themselves are bounded by quotation marks in the program. Note that in REM statements, BASIC, or SIMONS' BASIC key words which are not enclosed in quotation marks, for example 10 REM PRINT, will not be changed by the command ALTER PRINT&PRINT\$.

EXAMPLE: To change the character string "JOHN" into "MIKE" in the following program:

ENTER: 10 INPUT "IS YOUR NAME JOHN";A\$
 20 IF LEFT\$(A\$,1)<>"Y" THEN 40
 30 PRINT "HI JOHN";:PRINT "GREAT TO SEE YOU"
 40 END

COMMAND: ALTER "JOHN"&"MIKE" <RETURN>

DISPLAY: READY

TYPE: LIST <RETURN>

SIMONS' BASIC EXTENSION

DISPLAY: 10 INPUT "IS YOUR NAME MIKE";A\$
20 IF LEFT\$(A\$,1)<>"Y"THEN 40
30 PRINT "HI MIKE";:PRINT "GREAT TO SEE YOU"
40 END

RESULT: All occurrences of the character strings "JOHN" have been changed to "MIKE".

2.5 HELP

FORMAT: HELP

PURPOSE: To display the line in which an error occurred during program execution and highlight the position of the error in reverse field characters.

The HELP command only works if the command is given immediately after an error has been detected by the BASIC interpreter and whilst the error message is displayed.

NOTE

Because of the way the BASIC interpreter works, the exact error may not always be displayed. It will, however, be very close to the reverse field area.

EXAMPLE: To find the error in the following program:

ENTER: 10 FOR CO=1 TO 10
20 PRINT CO+2*3.142
30 NEXT C

TYPE: RUN <RETURN>

DISPLAY: 7.284
?
? NEXT WITHOUT FOR ERROR IN 30

COMMAND: HELP <RETURN>

DISPLAY: NEXT C (the letter "C" appears in reverse field display.)

2.6 CHAIN

FORMAT: CHAIN "program name",device number

PURPOSE: To load and run one program from within another.

The CHAIN command allows you to load one program from another program and automatically RUN the second program after it has been loaded. This means that longer programs can be loaded from smaller programs without the necessity of changing BASIC memory pointers. The parameter within quotation marks is the name of the program you wish to load. Device number refers to the storage medium on which the program to be loaded is stored. This is "8" for a disk unit and "1" for a cassette unit. If not specified, the device number defaults to "1", i.e. cassette.

EXAMPLE: To load and run the program named "PROG2" from within another program:

ENTER: 10 PRINT "<SHIFT/CLR/HOME>"
 20 PRINT "THIS PROGRAM"
 30 PRINT "DEMONSTRATES THE EFFECTIVENESS"
 40 PRINT "OF THE CHAIN COMMAND"

Save this program on cassette using the name "PROG2" to identify it.

TYPE: NEW <RETURN>

Rewind the cassette tape to the beginning of the program.

ENTER: 10 PRINT "PRESS RETURN TO LOAD"
 20 PRINT "THE NEXT PROGRAM"
 30 GET A\$: IF A\$ <> CHR\$(13) THEN 30
 40 CHAIN "PROG2"

TYPE: RUN <RETURN>

DISPLAY: PRESS RETURN TO LOAD
 THE NEXT PROGRAM

COMMAND: Press the RETURN key.

DISPLAY: PRESS PLAY ON TAPE

Press the PLAY key on the cassette unit. (The screen goes blank.)

DISPLAY: THIS PROGRAM
 DEMONSTRATES THE EFFECTIVENESS
 OF THE CHAIN COMMAND

TYPE: LIST <RETURN>

DISPLAY: 10 PRINT "<SHIFT/CLR/HOME>"
20 PRINT "THIS PROGRAM"
30 PRINT "DEMONSTRATES THE EFFECTIVENESS"
40 PRINT "OF THE CHAIN COMMAND"

RESULT: The first program has been erased from memory and "PROG2" has been loaded and executed automatically.

NOTE

The above example may also be effected with a disk unit. Line 40 of the second program must be changed to:

40 CHAIN "PROG2",8

2.7 PROTECT

FORMAT: PROTECT "PROGRAM NAME",8 (for disk)

or: PROTECT "PROGRAM NAME",1 (for cassette)

PURPOSE: To protect a program from being listed.

The PROTECT command has exactly the same syntax as SAVE, and may be used to stop your programs from being studied, or saved. Should RUN/STOP, with or without RESTORE, be pressed at any time during the execution of the program, the program is re-run from the beginning. This also applies if an error should occur. When a program is being protected, the screen clears, and SAVING PROGRAM NAME appears. When finished, the 64 returns to direct mode. Once a program has been protected, it is loaded with LOAD "PROGRAM NAME",8,1 if on disk, and LOAD "PROGRAM NAME",1,1 if on tape. When the program is loaded, it runs automatically.

WARNING

ALWAYS ENSURE THAT YOU HAVE A NON-PROTECTED COPY OF THE PROGRAM, AS ONCE IT HAS BEEN PROTECTED, YOU ARE UNABLE TO LIST IT, AND THUS CHANGE IT.

EXAMPLE: To protect a program from being listed:

ENTER: 10 HIRES 15,0
 20 COLOUR 5,0
 30 MULTI 3,5,7
 40 TEXT 10,20,"THIS PROGRAM",1,2,12
 50 TEXT 70,60,"IS",2,2,12
 60 TEXT 24,100,"PROTECTED!",3,2,12
 70 PAUSE 4
 80 STOP

TYPE: LIST <RETURN>

Protect the program using:

PROTECT "PROG",8 (for disk)

or: PROTECT "PROG",1 (for cassette)

TYPE: NEW <RETURN>

Load the program using:

LOAD "PROG",8,1 (for disk)

or: LOAD "PROG",1,1 (for cassette)

RESULT: The program runs automatically, and you are unable to stop execution.

2.8 FORCE

FORMAT: FORCE errn

PURPOSE: To force an error.

This command forces an error that would otherwise not have occurred. The parameter errn is equal to the error you wish to force. For a list of the error numbers see section 10.2 of your SIMONS' BASIC manual.

2.9 DS\$

FORMAT: PRINT DS\$

or: A\$=DS\$

PURPOSE: To read the disk error channel.

For more information on the disk error channel, see page 18 of your 1541 manual. DS\$ reads the disk error channel. Four variables that describe the error condition are read. The first variable is the error number, where 0 denotes no error (for more information on the error numbers, and messages, see page 43 of your 1541 manual). The second variable is the error description. The third variable is the track number on which the error occurred, and the fourth variable is the block (also known as a sector) number inside that track.

EXAMPLE: To read the disk error channel:

TYPE: PRINT DS\$ <RETURN>

DISPLAY: 00, OK,00,00

RESULT: This message denotes that no error has occurred.

NOTE

The length of DS\$ is always 30, e.g. PRINT LEN(DS\$) gives 30.

SECTION THREE

ARRAY MANIPULATION

3.1 INTRODUCTION

3.1.1 WHAT IS AN ARRAY?

You were introduced to the concept of arrays in the User's Guide supplied with your COMMODORE 64 computer. (See pages 95 thru 103.) To refresh your memory, an array is a group of subscripted variables. For example, A(15) is an array containing 15 individual items, or elements, of information. If you ran the program below:

```
10 A(1) = 21.56: A(3) = 13.78: A(4) = 13.89
```

then the memory would look like this:

```
A(0)
A(1)      21.56
A(2)
A(3)      13.78
A(4)      13.89
```

This is called a one-dimensional array.

Arrays that contain more than 10 elements must first be DIMensioned like this:

```
10 DIM A(25)
```

This tells the computer that you want to set up a one-dimensional array with a maximum of 25 elements.

3.1.2 MULTI-DIMENSIONAL ARRAYS

Arrays may be single or multi-dimensional. For example, a two-dimensional array would be written like this:

```
A(4,6)
```

and would be represented as a two-dimensional grid in memory.

1. A(4,6)

Each subscript in the array represents the row and column number in the grid where the particular element of the array is stored.

If you assigned the value 16 to A(3,4), then 16 could be thought of as having been placed in the 4th column of the 3rd row of the grid as shown below:

2. A(3,4) = 16

			16		

As with one-dimensional arrays, multi-dimensional arrays must first be DIMensioned. For example:

10 DIM A(12,7)

would create a two-dimensional array in memory labelled A having 12 rows and 7 elements in each row.

3.1.3 MANIPULATING ARRAYS

In standard BASIC, manipulation of arrays involves multiple operations requiring a great many instructions and, consequently, a great deal of memory space. The array commands provided by SIMONS' BASIC allow you to perform these operations using simple statements.

The ZER ARR command allows you to set each element in an array to zero, while the SET ARR command enables all the elements in an array to be set to a specified number or value. PRINT ARR allows you to display the contents of an array. ADD ARR enables the elements of two arrays to be added, while the SUB ARR command allows you to subtract the contents of one array from another. MUL ARR permits the contents of two arrays to be multiplied together; the DIV ARR command allows you to divide the contents of one array by another. The ADD ALL command permits you to add a particular value to each element in the array while the SUB ALL command allows you to subtract a specific number from each element in an array. The MUL ALL command multiplies all the elements in an array by a specified number and DIV ALL will divide all the elements by a selected value. The INPUT ARR command allows an array to be filled directly from the keyboard while READ ARR reads data into an array from a block of data statements. The COPY ARR command copies the contents of one array into a second, empty array. The SCRATCH command clears an array from memory.

SIMONS' BASIC EXTENSION also supplies a number of array functions. The ELEMENTS function returns the number of elements in an array while the SUM function adds all the elements of an array together. The MIN function returns the lowest value in an array while the MAX function returns the largest value.

NOTE

The array commands provided by SIMONS' BASIC are designed to be used on real number or integer arrays. Arrays containing strings CANNOT be used.

3.2 ARRAY COMMANDS

3.2.1 CONVENTIONS

When using SIMONS' BASIC array manipulation commands, the following conventions must be observed:

1. Each array must contain no more than 8 dimensions, i.e.: DIM A(1,1,1,1,1,1,1,1).
2. The zero (0) position in an array must NOT be used.
3. The arrays in the calculation and the array into which the result of the calculation is placed must be of the same size, i.e. the same dimensions and number of elements.
4. All the arrays must be correctly DIMensioned.
5. The array into which the result of a calculation is placed must be empty.

Failure to conform to any of the conventions above will result in the message:

? SYNTAX ERROR

3.2.2 SET ARR

FORMAT: SET ARR array name, value

PURPOSE: To assign a value to each element in an array.

The SET ARR command allows you to set every element in an array to a specified number. This number can be a pre-determined value or the result of a computation.

EXAMPLE: To place the number 6 in each location of a 4 row by 3 column two-dimensional array:

ENTER: 10 DIM AA(4,3)
 20 SET ARR AA,6
 30 FOR X = 1 TO 4
 40 FOR Y = 1 TO 3
 50 PRINT AA(X,Y),: NEXT Y,X

TYPE: RUN <RETURN>

DISPLAY: 6 6 6 6
 6 6 6 6
 6 6 6 6

3.2.3 PRINT ARR

FORMAT: PRINT ARR array name, print format

PURPOSE: To display the contents of an array.

The PRINT ARR command allows you to display the contents of an array. The first parameter of the command is the name of the array you wish to display. The second parameter in the command indicates the format in which the array is shown. The commands are as follows:

- 0 - displays each element of the array on a new line
- 1 - causes each element to be separated by a space
- 2 - separates each element of the array by ten spaces (like using a comma in a PRINT statement).

EXAMPLE: To print the contents of a one-dimensional array, each element being separated by a space:

ENTER: 10 DIM BB(20)
 20 X = 1
 30 REPEAT
 40 BB(X) = X * 2
 50 X = X + 1
 60 UNTIL X >20
 70 PRINT ARR BB,1

TYPE: RUN <RETURN>

DISPLAY: 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
 32 34 36 38 40

3.2.4 ZER ARR

FORMAT: ZER ARR array name

PURPOSE: To set an array to 0.

ZER ARR allows you to place a value of zero (0) in every element in an array.

EXAMPLE: Using the program in the previous section, to set each element in the array BB to zero:

```
ENTER: 10 DIM BB(20)
        20 X = 1
        30 REPEAT
        40 BB(X) = X * 2
        50 X = X + 1
        60 UNTIL X > 20
        70 PRINT ARR BB,1
        80 ZER ARR BB
        90 PRINT: PRINT ARR BB,1
```

TYPE: RUN <RETURN>

```
DISPLAY: 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
          32 34 36 38 40
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

3.2.5 ADD ARR

FORMAT: ADD ARR result array, array1, array2

PURPOSE: To add together two arrays.

The ADD ARR command adds together the contents of two arrays and places the result in a third, empty, array. The first parameter in the command is the variable name of the array INTO which the result of the addition is placed. The second and third parameters are the variable names of the two arrays which are added.

Arrays are added, and stored element to element, e.g. the first element in array1 is added to the first element in array2, and stored in the first element of array3, the sixth element in array1 is added to the sixth element in array2, and stored in the sixth element of array3 etc. The arrays in the addition must have the same dimensions, and number of elements.

EXAMPLE: To add the contents of arrays AA and BB, and place the result in array CC:

SIMONS' BASIC EXTENSION

ENTER: 10 DIM AA(20),BB(20),CC(20)
 20 ZER ARR CC
 30 SET ARR AA,10
 40 REPEAT
 50 BB(X)=X * 2
 60 X=X+1
 70 UNTIL X>20
 80 ADD ARR CC,AA,BB
 90 PRINT "AA=";;PRINT ARR AA,1:PRINT
 100 PRINT "BB=";;PRINT ARR BB,1:PRINT
 110 PRINT "CC=";;PRINT ARR CC,1:PRINT

TYPE: RUN <RETURN>

DISPLAY: AA= 10
 BB= 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
 CC= 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50

3.2.6 SUB ARR

FORMAT: SUB ARR result array, array1, array2

PURPOSE: To subtract one array from another.

SUB ARR subtracts the contents of one array from another and places the result in a third, empty, array. The first parameter in the command is the variable name of the array INTO which the result of the subtraction is placed. The second parameter is the variable name of the array SUBTRACTED, while the third parameter in the command is the variable name of the array FROM WHICH the first array is subtracted.

Arrays are subtracted, and stored element from element, e.g. the first element in array1 is subtracted from the first element in array2, and stored in the first element of array3, the fourth element in array1 is subtracted from the fourth element in array2, and stored in the fourth element of array3 etc. The arrays in the subtraction must have the same dimensions, and number of elements.

EXAMPLE: To subtract the contents of arrays AA and BB, and place the result in array CC:

```

ENTER:      10 DIM AA(20),BB(20),CC(20)
            20 ZER ARR CC
            30 SET ARR AA,60
            40 REPEAT
            50 BB(X)=X * 2
            60 X=X+1
            70 UNTIL X>20
            80 SUB ARR CC,AA,BB
            90 PRINT "AA=";;PRINT ARR AA,1:PRINT
           100 PRINT "BB=";;PRINT ARR BB,1:PRINT
           110 PRINT "CC=";;PRINT ARR CC,1:PRINT

```

TYPE: RUN <RETURN>

```

DISPLAY:   AA= 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60
           BB= 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
           CC= 60 58 56 54 52 50 48 46 44 42 40 38 36 34 32 30 28 26 24 22 20

```

3.2.7 MUL ARR

FORMAT: MUL ARR result array, array1, array2

PURPOSE: To multiply two arrays together.

The MUL ARR command multiplies the contents of one array by another and places the result in a third, empty, array. The first parameter in the command is the variable name of the array INTO which the result of the multiplication is placed. The second and third parameters are the variable names of the arrays that are multiplied.

Arrays are multiplied, and stored element by element, e.g. the first element in array1 is multiplied by the first element in array2, and stored in the first element of array3, the third element in array1 is multiplied by the third element in array2, and stored in the third element of array3 etc. The arrays in the multiplication must have the same dimensions, and number of elements.

EXAMPLE: To multiply the contents of arrays AA and BB, and place the result in array CC:

```

ENTER:      10 DIM AA(20),BB(20),CC(20)
            20 ZER ARR CC
            30 SET ARR AA,10
            40 REPEAT
            50 BB(X)=X+1
            60 X=X+1
            70 UNTIL X>20
            80 MUL ARR CC,AA,BB
            90 PRINT "AA=";;PRINT ARR AA,1:PRINT
           100 PRINT "BB=";;PRINT ARR BB,1:PRINT
           110 PRINT "CC=";;PRINT ARR CC,1:PRINT

```

TYPE: RUN <RETURN>

DISPLAY: AA= 10
 BB= 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
 CC= 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180
 190 200 210

3.2.8 DIV ARR

FORMAT: DIV ARR result array, array1, array2

PURPOSE: To divide one array by another.

The DIV ARR command divides the contents of one array by another and places the result in a third, empty, array. The first parameter in the command is the variable name of the array INTO which the result of the division is placed. The second parameter is the variable name of the array DIVIDED, and the third parameter in the command is the variable name of the array BY WHICH the division is made.

Arrays are divided, and stored element by element, e.g. the first element in array1 is divided by the first element in array2, and stored in the first element of array3, the third element in array1 is divided by the third element in array2, and stored in the third element of array3 etc. The arrays in the division must have the same dimensions, and number of elements.

EXAMPLE: To divide the contents of array AA by the contents of array BB, and place the result in array CC:

ENTER: 10 DIM AA(6),BB(6),CC(6)
 20 ZER ARR CC
 30 SET ARR AA,720
 40 FOR I=0TO6
 50 READ B
 60 BB(I)=B
 70 NEXT I
 80 DIV ARR CC,AA,BB
 90 PRINT "AA="";PRINT ARR AA,1:PRINT
 100 PRINT "BB="";PRINT ARR BB,1:PRINT
 110 PRINT "CC="";PRINT ARR CC,1:PRINT
 1000 DATA 1,2,4,6,8,10,12

TYPE: RUN <RETURN>

DISPLAY: AA= 720 720 720 720 720 720
 BB= 1 2 4 6 8 10 12
 CC= 720 360 180 120 90 72 60

3.2.9 ADD ALL

FORMAT: ADD ALL array name, value

PURPOSE: To add a value to every element in an array

The ADD ALL command adds a specified number to every element in a defined array. The first parameter in the command is the array to which the value is added. The second parameter is the value you wish to add. This may be an actual number or the result of a computation.

EXAMPLE: To add the value 20 to every element of the array AA:

```
ENTER: 10 DIM AA(20)
        20 FOR I=0TO20
        30 AA(I)=I
        40 NEXT I
        50 PRINT "AA=";:PRINT ARR AA,1:PRINT
        60 ADD ALL AA,20
        70 PRINT "AA=";:PRINT ARR AA,1:PRINT
```

TYPE: RUN <RETURN>

```
DISPLAY: AA= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
        AA= 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

3.2.10 SUB ALL

FORMAT: SUB ALL array name, value

PURPOSE: To subtract a number from every element in an array

SUB ALL subtracts a specified value from every element in a defined array. The first parameter in the command is the array which is subtracted from. The second command parameter is the value subtracted.

EXAMPLE: To subtract 20 from every element of array AA:

```
ENTER: 10 DIM ARR AA(20)
        20 FOR I=0TO20
        30 AA(I)=I+20
        40 NEXT I
        50 PRINT "AA=";:PRINT ARR AA,1:PRINT
        60 SUB ALL AA,20
        70 PRINT "AA=";:PRINT ARR AA,1:PRINT
```

TYPE: RUN <RETURN>

```
DISPLAY: AA= 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
        AA= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```


3.2.11 MUL ALL

FORMAT: MUL ALL array name, value

PURPOSE: To multiply every element in an array by a number.

The MUL ALL command multiplies every element in a defined array by a specified number. The first parameter in the command is the array which is multiplied. The second parameter is the multiplication factor.

EXAMPLE: To multiply every element in array AA by 10:

```
ENTER: 10 DIM AA(20)
        20 FOR I=0TO20
        30 AA(I)=I
        40 NEXT I
        50 PRINT "AA=";:PRINT ARR AA,1:PRINT
        60 MUL ALL AA,10
        70 PRINT "AA=";:PRINT ARR AA,1:PRINT
```

TYPE: RUN <RETURN>

```
DISPLAY: AA= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
          AA= 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180
          190 200
```

3.2.12 DIV ALL

FORMAT: DIV ALL array name, value

PURPOSE: To divide every element in an array by a number

DIV ALL divides every element in a defined array by a specified number. The first parameter in the command is the array that is divided, while the second command parameter is the division factor.

EXAMPLE: To divide every element in array AA by 10:

```
ENTER: 10 DIM AA(20)
        20 FOR I=0TO20
        30 AA(I)=I*10
        40 NEXT I
        50 PRINT "AA=";:PRINT ARR AA,1:PRINT
        60 DIV ALL AA,10
        70 PRINT "AA=";:PRINT ARR AA,1:PRINT
```

TYPE: RUN <RETURN>

```
DISPLAY: AA= 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180
          190 200
          AA= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

3.2.13 COPY ARR

FORMAT: COPY ARR array1, array2

PURPOSE: To copy one array to another.

The COPY ARR command duplicates the contents of one array into another, empty array. The first parameter in the command is the variable name of the array into which the information is copied. The second command parameter is the variable name of the array whose contents are copied.

EXAMPLE: To copy the contents of array AA into array BB:

```
ENTER: 10 DIM ARR AA(20),BB(20)
        20 ZER ARR BB
        30 FOR I=0TO20
        40 AA(I)=I*10
        50 NEXT I
        60 PRINT "AA=";;PRINT ARR AA,1:PRINT
        70 PRINT "BB=";;PRINT ARR BB,1:PRINT
        80 COPY ARR AA,BB
        90 PRINT "AA=";;PRINT ARR AA,1:PRINT
        100 PRINT "BB=";;PRINT ARR BB,1:PRINT
```

TYPE: RUN <RETURN>

```
DISPLAY: AA= 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180
          190 200
        BB= 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
        AA= 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180
          190 200
        BB= 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180
          190 200
```

3.2.14 INPUT ARR

FORMAT: INPUT ARR array name

PURPOSE: To fill an array from the keyboard.

INPUT ARR allows an array to be filled directly from the keyboard. The parameter 'array name' is the variable name of the array to be filled. Once the array has been filled, program execution continues.

EXAMPLE: To fill the array AA from the keyboard, and then display it:

ENTER: 10 DIM AA(20)
 20 PRINT "ENTER 21 NUMBERS"
 30 INPUT ARR AA
 40 PRINT "AA=";;PRINT ARR AA,1:PRINT

TYPE: RUN <RETURN>

RESULT: The numbers that you typed on the keyboard, are now elements of the array AA.

3.2.15 READ ARR

FORMAT: READ ARR array name

PURPOSE: To read data into an array.

The READ ARR command reads data from DATA statements into an array until the array is filled. The command parameter is the variable name of the array into which the data is stored.

EXAMPLE: To set all the elements in array AA to 720, read the contents of array BB from a DATA statement, and divide array AA by array BB:

ENTER: 10 DIM AA(6),BB(6),CC(6)
 20 ZER ARR CC
 30 SET ARR AA,720
 40 READ ARR BB
 50 DIV ARR CC,AA,BB
 60 PRINT "AA=";;PRINT ARR AA,1:PRINT
 70 PRINT "BB=";;PRINT ARR BB,1:PRINT
 80 PRINT "CC=";;PRINT ARR CC,1:PRINT
 1000 DATA 1,2,4,6,8,10,12

TYPE: RUN <RETURN>

DISPLAY: AA= 720 720 720 720 720 720
 BB= 1 2 4 6 8 10 12
 CC= 720 360 180 120 90 72 60

3.2.16 SCRATCH

FORMAT: SCRATCH array name

PURPOSE: To delete an array from memory.

The SCRATCH command removes an array from memory. It differs from the ZER ARR command in that the scratched array name can no longer be used as a parameter in an array manipulation command. The parameter 'array name' is the variable name of the array to be SCRATCHed.

NOTE

If you wish to use a variable name of an array that has been SCRATCHed, always remember to re-DIMension the array.

EXAMPLE: To DIMension arrays AA and BB, and SCRATCH array AA:

ENTER: 10 DIM AA(20,20)
 20 PRINT "AA DIMENSIONED";FRE(0)
 30 DIM BB(20,20)
 40 PRINT "BB DIMENSIONED";FRE(0)
 50 SCRATCH AA
 60 PRINT "AA SCRATCHED ";FRE(0)

TYPE: RUN <RETURN>

DISPLAY: AA DIMENSIONED 25948
 BB DIMENSIONED 23734
 AA SCRATCHED 25948

3.2.17 SUM

FORMAT: A=SUM(array name)

or: PRINT SUM(array name)

PURPOSE: To add the contents of an array.

The SUM function gives the result of adding together all the elements in an array. The parameter enclosed in brackets is the variable name of the array whose elements you wish to add.

EXAMPLE: To set all the elements of array AA to 10, and add them together:

ENTER: 10 DIM AA(20)
 20 SET ARR AA,10
 30 A=SUM(AA)
 40 PRINT "THE TOTAL SUM OF AA=";A

TYPE: RUN <RETURN>

DISPLAY: THE TOTAL SUM OF AA= 210

3.2.18 ELEMENTS

FORMAT: variable=ELEMENTS(array name)

or: PRINT ELEMENTS(array name)

PURPOSE: To determine the number of elements in an array.

The ELEMENTS function returns the number of elements in a specified array. The parameter inside brackets is the variable name of the array whose contents you wish to examine.

EXAMPLE: To determine the number of elements in array AA:

ENTER: 10 DIM AA(20)
20 SET ARR AA,10
30 A=SUM(AA)
40 E=ELEMENTS(AA)
50 PRINT "THE TOTAL SUM OF AA=";A
60 PRINT "THE TOTAL NUMBER OF ELEMENTS IN AA=";E

TYPE: RUN <RETURN>

DISPLAY: THE TOTAL SUM OF AA= 210
THE TOTAL NUMBER OF ELEMENTS IN AA= 21

3.2.19 MIN

FORMAT: A=MIN(array name)

or: PRINT MIN(array name)

PURPOSE: To return the lowest number in an array.

The MIN function returns the value of the lowest element in a specified array. The parameter enclosed in brackets is the variable name of the array whose minimum element you wish to determine.

EXAMPLE: To determine the minimum value in array AA:

ENTER: 10 DIM AA(20)
20 READ ARR AA
30 PRINT "AA=";:PRINT ARR AA,1:PRINT
40 MN=MIN(AA)
50 PRINT "MINIMUM VALUE IN AA=";MN
1000 DATA -40,-31,22,-29,23,7,-50
1010 DATA -17,-28,45,-7,47,-44,-20
1020 DATA 40,17,27,18,7,13,-40

TYPE: RUN <RETURN>

DISPLAY: AA= -40 -31 22 -29 23 7 -50 -17 -28 45 -7 47 -44 -20 40 17 27 18 7 13 -40
MINIMUM VALUE IN AA=-50

3.2.20 MAX

FORMAT: variable = MAX(array name)

or: PRINT MAX(array name)

PURPOSE: To return the highest number in an array.

The MAX function returns the value of the largest element in a specified array. The parameter enclosed in brackets is the variable name of the array whose maximum element you wish to determine.

EXAMPLE: To determine the maximum value in array AA:

ENTER: 10 DIM AA(20)
20 READ ARR AA
30 PRINT "AA=";:PRINT ARR AA,1:PRINT
40 MX=MAX(AA)
50 PRINT "MAXIMUM VALUE IN AA=";MX
1000 DATA -40,-31,22,-29,23,7,-50
1010 DATA -17,-28,45,-7,47,-44,-20
1020 DATA 40,17,27,18,7,13,-40

TYPE: RUN <RETURN>

DISPLAY: AA= -40 -31 22 -29 23 7 -50 -17 -28 45 -7 47 -44 -20 40 17 27 18 7 13 -40
MAXIMUM VALUE IN AA= 47

3.3 STRING ARRAY MANIPULATION

3.3.1 SORT

FORMAT: SORT array\$,dflag,start,end

PURPOSE: To sort a single dimension string array.

The SORT command sorts any single dimension string containing alpha and/or numeric characters into ascending or descending order. The first parameter, array\$, is the name of the string array. The next parameter, dflag, specifies whether the array is to be sorted into ascending, or descending order. A "1" in this position denotes ascending order, and a "0" in this position denotes descending order. The last two parameters, start and end, dictate which part of the string element is sorted. Either the whole of each string element, or just a part. For example, SORT A\$,0,3,255 sorts the string array A\$ into descending order. But only from the third character of each element onwards. This is especially useful should you want to sort an array containing names, but wish to ignore the initials. For more information on string arrays see Appendix C of your COMMODORE 64 USER'S MANUAL, and pages 4 - 9 of the COMMODORE 64 PROGRAMMER'S REFERENCE GUIDE.

EXAMPLE: To read names from a DATA statement into the array A\$, and sort that array into descending order:

```
ENTER: 10 DIM A$(5)
        20 PRINT "UNSORTED NAMES"
        30 PRINT "-----"
        40 FOR I=0TO5
        50 READ A$(I)
        60 PRINT A$(I)
        70 NEXT I
        80 SORT A$,0,3,255
        90 PRINT
        100 PRINT "SORTED NAMES"
        110 PRINT "-----"
        120 FOR I=0TO5
        130 PRINT A$(I)
        140 NEXT I
        1000 DATA "MR JONES","MR SMITH","MR BLOGGS"
        1010 DATA "MR BROWN","MR WHITE","MR BLACK"
```

TYPE: RUN <RETURN>

DISPLAY: UNSORTED NAMES

MR JONES
MR SMITH
MR BLOGGS
MR BROWN
MR WHITE
MR BLACK

SORTED NAMES

MR WHITE
MR SMITH
MR JONES
MR BROWN
MR BLOGGS
MR BLACK



SECTION FOUR NUMERIC AIDS

4.1 INTRODUCTION

Section Four contains additional commands to help you when manipulating numeric data. GRAD and DEG are two additional trigonometrical functions. The first converts a number expressed in gradians into radians, while the second converts a number expressed in degrees into radians. Also included in this section are commands to convert decimal numbers into their binary or hexadecimal equivalent.

4.2 ADDITIONAL TRIGONOMETRICAL FUNCTIONS

4.2.1 GRAD

FORMAT: variable=GRAD(number)

PURPOSE: To convert gradians into rads.

The GRAD function converts a value expressed in gradians into rads. It is used like any normal mathematical function.

EXAMPLE: To use the GRAD function to plot a sine wave:

```
ENTER: 10 HIRES 11,15
        20 FOR I=0 TO GRAD(400) STEP GRAD(400)/64
        30 Y=SIN(I)*50+100
        40 X=I*50
        50 PLOT X,Y,1
        60 NEXT I
        70 PAUSE 10
```

TYPE: RUN <RETURN>

RESULT: A sine wave is plotted on the high-resolution screen.

4.2.2 DEG

FORMAT: variable=DEG(number)

PURPOSE: To convert degrees into rads.

The DEG function converts a value expressed in degrees into rads. It is used like any normal mathematical function.

EXAMPLE: To use the DEG function to plot a sine wave:

ENTER: 10 HIRES 11,15
20 FOR I=0 TO DEG(360) STEP DEG(360)/64
30 Y=SIN(I)*50+100
40 X=I*50
50 PLOT X,Y,1
60 NEXT I
70 PAUSE 10

TYPE: RUN <RETURN>

RESULT: A sine wave is plotted on the high-resolution screen.

4.3 NUMERIC CONVERSION

4.3.1 BIN\$ - DECIMAL/HEXADECIMAL TO BINARY CONVERSION

FORMAT: BIN\$(n)

PURPOSE: To convert from decimal, or hexadecimal into binary.

The BIN\$ command converts a decimal, or hexadecimal number into its binary equivalent.

If a decimal number outside the range 0-255, or a hexadecimal number outside the range \$0000-\$00FF is used as the argument in the command, the message:

? ILLEGAL QUANTITY ERROR

is displayed.

EXAMPLE: To convert the decimal number 135 into its binary equivalent:

COMMAND: PRINT BIN\$(135) <RETURN>

DISPLAY: 10000111

4.3.2 HEX\$ - DECIMAL/BINARY TO HEXADECIMAL CONVERSION

FORMAT: HEX\$(n)

PURPOSE: To convert from decimal, or binary into hexadecimal.

The HEX\$ command converts a decimal, or binary number into its hexadecimal equivalent.

If a decimal number outside the range 0-65535 is used as the argument in the command, the message:

? ILLEGAL QUANTITY ERROR

is displayed.

Only 8-bit binary numbers in the range %00000000-%11111111 may be used. If a negative binary number is used as the argument in the command, the message:

? ILLEGAL QUANTITY ERROR

is displayed.

EXAMPLE: To convert the decimal number 220 into its hexadecimal equivalent:

COMMAND: PRINT HEX\$(220) <RETURN>

DISPLAY: 00DC

4.4 SPECIAL CALCULATIONS

4.4.1 CALCX

FORMAT: A=CALCX(x,y,angle,xr,yr)

or: PRINT CALCX(x,y,angle,xr,yr)

PURPOSE: To calculate the x coordinate of a point on the circumference of a circle.

The CALCX function returns the x coordinate of a point on the circumference of a circle. It is essentially ANGL (see section 6.5.13 of your SIMONS' BASIC MANUAL), but without the y coordinate, and instead of drawing a line, CALCX returns the point. The first two parameters, x and y, are the centre coordinates of the imaginary circle. The next parameter, angle, is the angle of the point to the perpendicular, and xr and yr are the x and y radii of the circle.

EXAMPLE: To use the CALCX function to draw a pattern on the high-resolution screen:

ENTER: 10 HIRES 1,0
 20 FOR R=10 TO 100
 30 FOR A=0 TO 180 STEP 20
 40 X=CALCX(160,100,A+R,R,R)
 50 Y=CALCY(160,100,A+R,R,R)
 60 PLOT X,Y,1
 70 NEXT
 80 NEXT

TYPE: RUN <RETURN>

RESULT: A pattern is drawn on the high-resolution screen.

4.4.2 CALCY

FORMAT: A=CALCY(x,y,angle,xr,yr)

or: PRINT CALCY(x,y,angle,xr,yr)

PURPOSE: To calculate the y coordinate of a point on the circumference of a circle.

The CALCY function returns the y coordinate of a point on the circumference of a circle. It is essentially ANGL (see section 6.5.13 of your SIMONS' BASIC MANUAL), but without the x coordinate, and instead of drawing a line, CALCY returns the point. The first two parameters, x and y, are the centre coordinates of the imaginary circle. The next parameter, angle, is the angle of the point to the perpendicular, and xr and yr are the x and y radii of the circle.

EXAMPLE: To use the CALCY function to draw a pattern on the high-resolution screen:

ENTER: 10 HIRES 1,0
 20 FOR R=10 TO 100
 30 FOR A=0 TO 180 STEP 20
 40 X=CALCX(160,100,A+R,R,R)
 50 Y=CALCY(160,100,A+R,R,R)
 60 PLOT X,Y,1
 70 NEXT
 80 NEXT

TYPE: RUN <RETURN>

RESULT: A pattern is drawn on the high-resolution screen.

WARNING
IN CERTAIN SITUATIONS CALCX AND CALCY MUST NOT BE INCORPORATED IN HIGH-RESOLUTION COMMANDS, IF YOU RECEIVE AN ERROR MESSAGE, ALTER YOUR PROGRAM SO THAT THE RESULTS OF THE CALCX AND CALCY CALCULATIONS ARE STORED IN VARIABLES, AND USE THE VARIABLES IN THE HIGH-RESOLUTION COMMAND.

4.5 EVALUATING A STRING AS A BASIC EXPRESSION

4.5.1 EVAL

FORMAT: EVAL str\$

PURPOSE: To evaluate a string as a BASIC expression.

EVAL evaluates a string as a BASIC expression. The single parameter, str\$, is any string that is stored in the 64's memory. This command is extremely useful for any program that requires data entry of expressions that need to be calculated.

NOTE

Temporary strings will not work with EVAL, i.e.:

```
10 A$="Y=X*10:BACK"
20 EVAL A$
30 PRINT Y
```

A\$ in this example is a temporary string, and will not be evaluated. If you are uncertain, use a string add on any string that needs to be evaluated, i.e.:

```
10 A$="Y="+X*10:BACK"
20 EVAL A$
30 PRINT Y
```

EXAMPLE: To enter the "X" part of the equation, evaluate the whole equation, and give the answer:

```
ENTER: 10 A$="Y="+X*10:BACK"
        20 PRINT "Y=X*10"
        30 INPUT "ENTER NUMBER FOR X";X
        40 EVAL A$
        50 PRINT "Y=";Y
        60 GOTO 30
```

TYPE: RUN <RETURN>

RESULT: When you enter the "X" part of the equation and press RETURN, A\$ is evaluated and the answer is put into variable Y.

4.5.2 BACK

FORMAT: str\$:BACK

PURPOSE: To ensure that program execution is correctly resumed after EVAL.

BACK must always be used in conjunction with EVAL. It is added to the end of the string, and ensures that program execution is correctly resumed after the string has been evaluated. BACK is always preceded by a colon.

EXAMPLE: To input a string, evaluate it, and give the answer:

```
ENTER: 10 INPUT A$
        20 EVAL "A="+A$+":BACK"
        30 PRINT "ANSWER=";A
        40 GOTO 10
```

TYPE: RUN <RETURN>

RESULT: When you enter an equation and press RETURN, A\$ is evaluated and the answer is put into variable A.

SECTION FIVE

MEMORY MANIPULATION COMMANDS

5.1 INTRODUCTION

In Section Five, the commands DEEK and DOKE are explained. The first allows you to read a 16 bit number in memory while the second enables you to assign a 16 bit number to two consecutive bytes of memory. Also included in this section are the commands HIMEM and LOMEM which allow the top and bottom of BASIC memory to be read or changed.

5.2 DOKE

FORMAT: DOKE a,b

PURPOSE: To assign a 16 bit value to memory.

The DOKE command enables you to place a 16 bit value in a specific memory location. This saves you having to first convert the number into low byte/high byte format and then using two POKEs.

EXAMPLE: To use the DOKE command to change the frequency control registers of the SID chip:

```
ENTER: 10 SID=54272
        20 VOL 15
        30 ENVELOPE 1,0,8,8,8
        40 WAVE 1,00010001
        50 FOR I=0 TO 65535 STEP 10
        60 DOKE SID,I
        70 NEXT I
```

TYPE: RUN <RETURN>

RESULT: A low to high glide is produced.

5.3 DEEK

FORMAT: DEEK a,b

PURPOSE: To read two consecutive bytes of memory.

DEEK is the converse of the DOKE command, i.e. it allows you to read two consecutive bytes of memory. The value returned is automatically converted into decimal form.

EXAMPLE: To use the DEEK command to determine the end of program address:

ENTER: 10 PRINT "END OF PROGRAM ADDRESS IS:";DEEK(45)

TYPE: RUN <RETURN>

DISPLAY: END OF PROGRAM ADDRESS IS: 2092

5.4 HIMEM

FORMAT: variable=HIMEM

or: HIMEM(address)

PURPOSE: To read or set the top of BASIC memory.

The HIMEM command allows you to read, or re-set the top of BASIC memory pointer.

EXAMPLE: To read the top of BASIC memory:

TYPE: PRINT "THE TOP OF MEMORY IS AT";HIMEM <RETURN>

DISPLAY: THE TOP OF MEMORY IS AT 30336

5.5 LOMEM

FORMAT: variable=LOMEM

or: LOMEM(address)

PURPOSE: To read or set the bottom of BASIC memory.

The LOMEM command allows you to read, or re-set the bottom of BASIC memory pointer.

EXAMPLE: To read the bottom of BASIC memory:

TYPE: PRINT "THE BOTTOM OF MEMORY IS AT";LOMEM<RETURN>

DISPLAY: THE BOTTOM OF BASIC MEMORY IS AT 2049

5.6 SCREEN

FORMAT: A=SCREEN

or: PRINT SCREEN

PURPOSE: To determine the current position of the screen in memory.

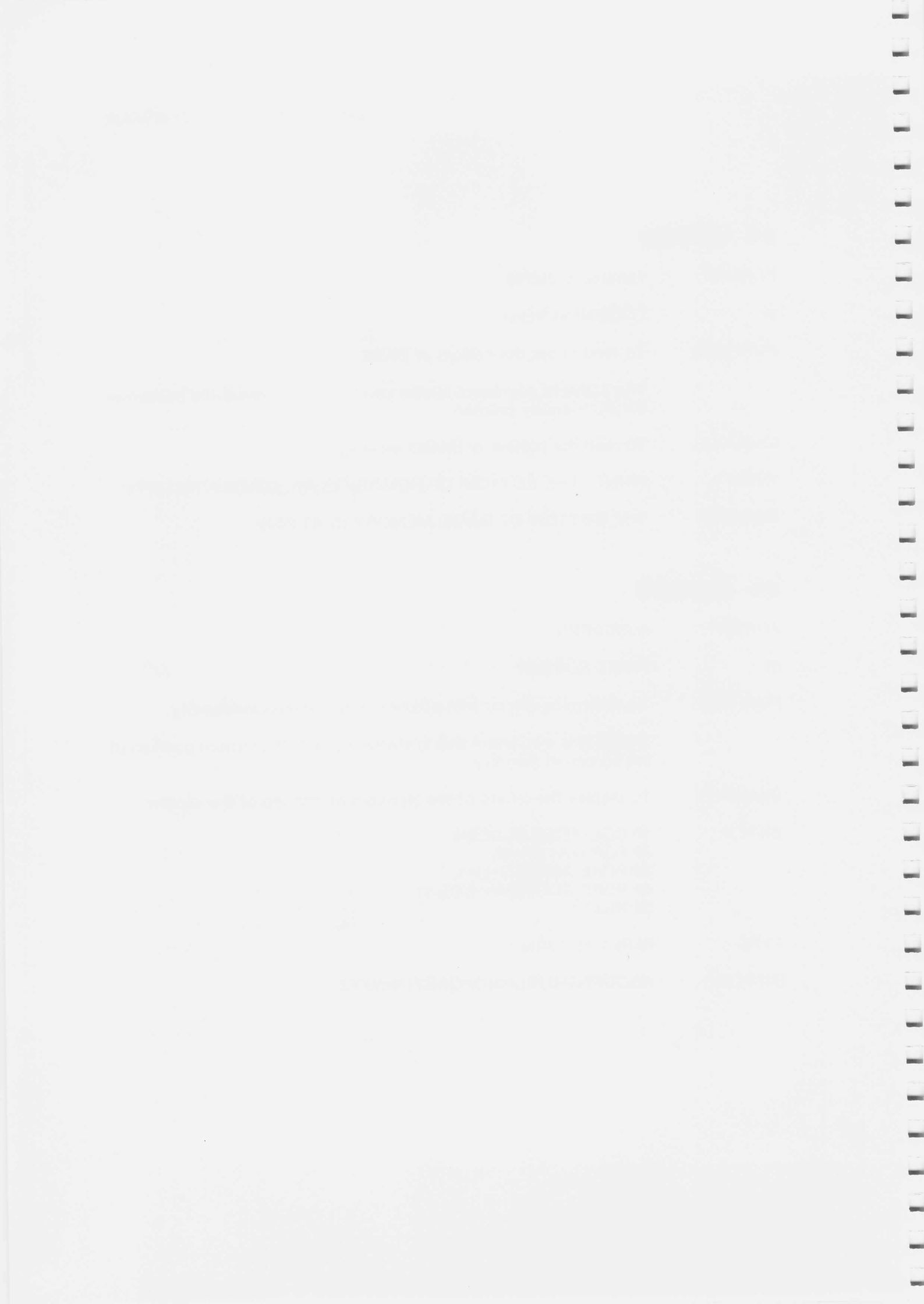
SCREEN is a constant that is always equal to the current position of the screen in memory.

EXAMPLE: To display the letters of the alphabet on the top of the screen:

```
ENTER: 10 COL=55296-SCREEN
        20 FOR I=ATO A+25
        30 POKE SCREEN+I,I+1
        40 POKE SCREEN+I+COL,11
        50 NEXT I
```

TYPE: RUN <RETURN>

DISPLAY: ABCDEFGHIJKLMNOPQRSTUVWXYZ



SECTION SIX

HIGH-RESOLUTION AND MULTI-COLOUR GRAPHICS

6.1 INTRODUCTION

This section contains additional commands for use on high-resolution and/or multi-colour graphics screen. These commands allow you to draw grids for producing graphs and histograms, display characters in a specified size at a defined slant, store/load graphics screens and draw horizontal and vertical lines.

The GRID command plots a grid on a graphics screen while the TICK command allows you to draw calibrations on the x and y axes of the grid. The HSAVE and HLOAD commands enable you to respectively store and recall graphics screens. The LABEL command enables you to place characters on a graphics screen in a specified size at a defined degree of slant. Also included in this section are the VLIN and HLIN commands which permit you to draw vertical and horizontal lines respectively on a graphics screen.

6.2 HIGH-RESOLUTION AND MULTI-COLOUR GRAPHICS COMMANDS

6.2.1 GRID

FORMAT: GRID x,y,tx,ty,n,n1,plot type

PURPOSE: To draw a grid.

The GRID command allows you to draw a GRID on a high-resolution graphics screen. The first two parameters, x and y, refer to the location of the intersection of the x and y axes of the grid, i.e. the bottom left-hand corner of the grid. The parameters tx and ty indicate, respectively, the distance between calibrations on the x and y axes. The next two parameters, nx and ny, specify the number of calibrations required on first the x, then the y axes. Plot type is as described in in Section 6.4 of the SIMONS' BASIC manual.

Note that, if the calibration marks go off the screen, the message:

BAD MODE

is displayed and you must amend your code accordingly.

EXAMPLE: To draw a large grid, with a smaller grid inside it:

ENTER: 10 HIRES 15,0
20 GRID 10,190,60,60,4,3,1
30 GRID 70,130,5,5,12,12,1
40 PAUSE 5

TYPE: RUN <RETURN>

RESULT: A large grid, with a smaller grid inside one of its squares is drawn on the high-resolution screen.

6.2.2 TICK

FORMAT: TICK x,y,tx,ty,n,n1,plot type

PURPOSE: To draw the axes of a grid.

The TICK command differs from GRID (see the previous section) in that it only draws and calibrates the x and y axes of a grid. This allows you to use the grid to produce graphs, histograms etc. The parameters are the same as those used in the GRID command.

Note that, if the calibration marks go off the screen, the message:

BAD MODE

is displayed and you must amend your code accordingly.

EXAMPLE: To "tick" the x and y axes of the large grid:

ENTER: 10 HIRES 15,0
20 GRID 10,190,60,60,4,3,1
30 GRID 70,130,5,5,12,12,1
40 TICK 10,190,5,5,48,36,1
50 PAUSE 5

TYPE: RUN <RETURN>

RESULT: The x and y axes of the large grid are "ticked".

6.2.3 HSAVE

FORMAT: HSAVE 2,8,2,"drive number:program name,S,W"

or: HSAVE 1,1,1,"program name"

PURPOSE: To save a graphics screen to disk or tape.

The HSAVE command stores a graphics screen on diskette or cassette. 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. When using HSAVE, the secondary address must be 1 for cassette, and any number in the range 2-14 for disk. 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 HLOAD 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 41 blocks. Note that the parameters are separated by commas and quotation marks are placed around name and S,W.

EXAMPLE: To draw a picture on the high-resolution screen, and save it on disk or cassette:

ENTER: 10 HIRES 15,0
 20 FOR R=10TO90 STEP 5
 30 CIRCLE 160,100,R,100-R,1
 40 NEXT
 50 PAUSE 5
 60 HSAVE 2,8,2,"HIRES SAVE,S,W":REM *** FOR DISK ***

or: 60 HSAVE 1,1,1,"HIRES SAVE":REM *** FOR CASSETTE ***

TYPE: RUN <RETURN>

RESULT: A design is drawn on the high-resolution screen, and is saved on disk or cassette after a pause of five seconds.

6.2.4 HLOAD

FORMAT: HLOAD 2,8,2,"drive number:program name"

or: HLOAD 1,1,0,"program name"

PURPOSE: To recall and re-display a previously saved graphics screen from diskette or cassette.

The HLOAD command allows you to recall and display a graphics screen that has been stored with the HSAVE 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. When using HLOAD, the secondary address must be 0 for cassette, and any number in the range 2-14 for disk. The title you assigned to the screen data is the final parameter and must be enclosed in quotation marks.

NOTE

Before you recall a graphics screen, you must ensure that the screen is in the same graphics mode as it was when the screen was stored. For example, before using HLOAD, enter :

HIRES 0,1:MULTI 2,3,4

For recalling a multicolour screen.

HIRES 0,1

For recalling a high resolution screen.

EXAMPLE: To recall a previously saved high-resolution screen from disk or cassette:

ENTER: 10 HIRES 15,0
20 HLOAD 2,8,2,"HIRES SAVE,S,R":REM *** FOR DISK ***
30 PAUSE 5

or: 20 HLOAD 1,1,0,"HIRES SAVE":REM *** FOR CASSETTE ***

TYPE: RUN <RETURN>

RESULT: Your high-resolution screen is recalled from disk or cassette.

6.2.5 LABEL

FORMAT: LABEL x,y,"text",plot type,xe,ye,sl,x1,y1,r

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

LABEL allows you to print character strings on a graphics screen at a defined size, slant and rotation. The parameters x and y specify the screen coordinates of the first letter of the string. The next parameter is the string itself. Plot type is as described in Section 6.4 of the SIMONS' BASIC manual. The parameters xe and ye specify the expansion factor of each character in the string in the horizontal and vertical directions respectively. Normal expansion is "1". Any increase in this figure causes a corresponding increase in size, e.g. a value of "3" for the x expansion would cause the characters to be displayed at three times their normal width. Note that you may not use an expansion factor greater than "30". The parameter sl refers to the slant of each character to the vertical. A value of "1" is used to display characters vertically. "0" displays the characters slanted to the left, while "2" causes the characters to be slanted to the right. The final two parameters in the LABEL command indicate the offsets in the horizontal and vertical directions respectively between each character in the string.

EXAMPLE: To display text of various sizes, slants, and angles of rotation on the high-resolution screen:

ENTER:

```

10 HIRES 15,0
20 LABEL 10,10,"<CTRL B><SHIFT A>LL THESE
   LETTERS",1,2,4,1,12,0,0
30 LABEL 250,0,"<CTRL B>ARE PRODUCED",1,4,2,0,0,16,1
40 LABEL 20,160,"<CTRL B>USING",1,4,2,0,30,-5,0
50 LABEL 10,80,"LABEL",1,4,4,1,30,5,0
60 PAUSE 5

```

TYPE: RUN <RETURN>

RESULT: Different parts of the message: "All these letters are produced using LABEL" are displayed in different sizes, slants, and angles of rotation on the high-resolution screen.

6.2.6 DRAW TO

FORMAT: DRAW TO x,y,plot type

or: DRAW TO,x,y,plot type TO x1,y1 TO x2,y2 (etc.)

PURPOSE: To draw a line from the last plotted point.

The DRAW TO command draws a line from the last plotted point to the position specified. The parameters x and y specify the horizontal and vertical screen coordinates of the point to be drawn to. Plot type is as described in Section 6.4 of your SIMONS' BASIC MANUAL. The DRAW TO command may also be used to draw a series of lines by adding extra TOs. When using this second type of format, plot type need only be specified once.

EXAMPLE: To draw a design on the high-resolution screen using the DRAW TO command:

```
ENTER: 10 HIRES 15,0:PLOT 0,0,0
        20 X1=0:X2=320:Y1=0:Y2=200
        30 REPEAT
        40 DRAW TO X2,Y1,1
        50 DRAW TO X2,Y2,1
        60 DRAW TO X1,Y2,1
        70 DRAW TO X1,Y1+5,1
        80 X1=X1+5
        90 X2=X2-5
        100 Y1=Y1+5
        110 Y2=Y2-5
        120 UNTIL Y1>200
        130 PAUSE 5
```

TYPE: RUN <RETURN>

RESULT: A design made up of decreasing rectangles is drawn on the high-resolution screen.

6.2.7 VLIN

FORMAT: VLIN x,y1,y2,plot type

PURPOSE: To draw a vertical line.

VLIN draws a vertical line from y1 to y2 in the xth column across the screen. The advantage of using VLIN over LINE when drawing vertical lines, is that VLIN is much faster, and needs one less parameter. Plot type is as described in Section 6.4 of your SIMONS' BASIC MANUAL.

NOTE

The parameter y1 must be less than y2. If y1 is greater than y2, then only one point is plotted, that at y1.

EXAMPLE: To draw a design on the high-resolution screen using the VLIN and HLIN commands:

```
ENTER: 10 HIRES 15,0:PLOT 0,0,0
        20 X1=2:X2=318:Y1=2:Y2=198
        30 REPEAT
        40 HLIN Y1,X1,X2,1
        50 VLIN X2,Y1,Y2,1
        60 HLIN Y2,X1,X2,1
        70 VLIN X1,Y1,Y2,1
        80 X1=X1+8
        90 X2=X2-8
        100 Y1=Y1+8
        110 Y2=Y2-8
        120 UNTIL Y1>160
        130 PAUSE 5
```

TYPE: RUN <RETURN>

RESULT: A design consisting of decreasing rectangles is displayed on the high-resolution screen.

6.2.8 HLIN

FORMAT: HLIN y,x1,x2,plot type

PURPOSE: To draw a horizontal line.

HLIN draws a horizontal line from x1 to x2 in yth row down the screen. The advantage of using HLIN over LINE when drawing horizontal lines, is that HLIN is much faster, and needs one less parameter. Plot type is as described in Section 6.4 of your SIMONS' BASIC MANUAL.

NOTE

The parameter x1 must be less than x2. If x1 is greater than x2, then only one point is plotted, that at x1.

EXAMPLE: To draw a design on the high-resolution screen using the VLIN and HLIN commands:

```
ENTER: 10 HIRES 15,0:PLOT 0,0,0
        20 X1=2:X2=318:Y1=2:Y2=198
        30 REPEAT
        40 HLIN Y1,X1,X2,1
        50 VLIN X2,Y1,Y2,1
        60 HLIN Y2,X1,X2,1
        70 VLIN X1,Y1,Y2,1
        80 X1=X1+8
        90 X2=X2-8
        100 Y1=Y1+8
        110 Y2=Y2-8
        120 UNTIL Y1>160
        130 PAUSE 5
```

TYPE: RUN <RETURN>

RESULT: A design consisting of decreasing rectangles is displayed on the high-resolution screen.

6.3 SCALING FUNCTIONS

6.3.1 SCALE

FORMAT: SCALE hx,hy

PURPOSE: To set up a scaling factor.

The SCALE command sets up a scaling factor for the high-resolution screen. The parameter hx is the largest number that the x coordinate can be, and the parameter hy is the largest number that the y coordinate can be. These numbers can be as large or as little as you wish, within the standard floating point limits (see pages 26 and 27 of your COMMODORE 64 USER'S GUIDE).

EXAMPLE: To set up a scaling factor, and draw a sine wave on the high-resolution screen using that scaling factor:

ENTER: 10 HIRES 15,0
 20 SCALE 1000,1000
 30 C=500
 40 FOR I=0TO1000
 50 P=SIN(I/160)*400+C
 60 X=SCX(I)
 70 Y=SCY(P)
 80 PLOT X,Y,1
 90 NEXT I

TYPE: RUN <RETURN>

RESULT: A sine wave is drawn on the high-resolution screen.

6.3.2 SCX

FORMAT: A=SCX(expression)

PURPOSE: To return a scaled value of X.

The SCX function returns a scaled value of X. It is used like any normal mathematical operator.

EXAMPLE: To set up a scaling factor, and draw a sine wave on the high-resolution screen using that scaling factor:

ENTER: 10 HIRES 15,0
 20 SCALE 1000,1000
 30 C=500
 40 FOR I=0TO1000
 50 P=SIN(I/160)*400+C
 60 X=SCX(I)
 70 Y=SCY(P)
 80 PLOT X,Y,1
 90 NEXT I

TYPE: RUN <RETURN>

RESULT: A sine wave is drawn on the high-resolution screen.

6.3.3 SCY

FORMAT: A=SCY(expression)

PURPOSE: To return a scaled value of Y.

The SCY function returns a scaled value of Y. It is used like any normal mathematical operator.

EXAMPLE: To set up a scaling factor, and draw a sine wave on the high-resolution screen using that scaling factor:

```
ENTER: 10 HIRES 15,0
        20 SCALE 1000,1000
        30 C=500
        40 FOR I=0TO1000
        50 P=SIN(I/160)*400+C
        60 X=SCX(I)
        70 Y=SCY(P)
        80 PLOT X,Y,1
        90 NEXT I
```

TYPE: RUN <RETURN>

RESULT: A sine wave is drawn on the high-resolution screen.

SECTION SEVEN

LOW-RESOLUTION GRAPHICS COMMANDS

7.1 INTRODUCTION

Section Seven contains graphics commands for use on a low-resolution screen. LOW RES initializes the low-resolution graphics mode. In this mode, each point plotted is four pixels high, and four pixels wide. The PREPARE command prepares a low-resolution screen so that the user can fill it with any information he wishes. The PUSH and PULL commands respectively store and retrieve this screen from memory. MCOL allows multi-coloured characters to be displayed, each character being made up of three colours. The DESIGN command, implemented in SIMONS' BASIC can be used to create user-defined multi-coloured characters. The ROTATE command allows you to rotate characters on a low-resolution screen. The FONT command enables double-sized characters to be displayed on a low-resolution screen. UPPER is used in conjunction with FONT to define the characters you wish to display. The ECOL command allows you to swap one colour for another in a specified area of a low-resolution screen. The BCKFLASH command allows you to flash character colours.

Also included in this section are various screen functions. The CCOL function returns the current colour of the cursor; SCOL returns the colour at a defined screen location and the SCHR function enables you to determine the character at a specified position on the screen. To complete this section, the "★" command is explained. This allows you to direct output to the printer without having to use the standard BASIC CMD command.

7.2 LOW-RESOLUTION GRAPHICS COMMANDS

7.2.1 LOW RES

FORMAT LOW RES 10,pc

or: LOW RES 0

PURPOSE: To initialize the low-resolution graphics mode, and select a plotting colour.

The LOW RES command sets the screen into the low-resolution graphics mode, and selects a plotting colour, pc. All plotting now takes place exactly as in high-resolution and multi-colour graphics modes, except that the screen is divided into an 80 by 50 dot matrix. This means that each dot plotted in low-resolution mode is four pixels high, and four pixels wide. LOW RES 0 terminates low-resolution plotting.

WARNING
CERTAIN HIGH-RESOLUTION GRAPHICS
COMMANDS SHOULD NOT BE USED IN
LOW-RESOLUTION MODE. THEY ARE:

BLOCK
GRID
TICK
LABEL
REC

EXAMPLE: To draw a low-resolution circle, and colour it red, a square, and colour it white, and a triangle, and colour it blue:

ENTER: 10 PRINT "<SHIFT CLR/HOME>":COLOUR 5,0
20 LOW RES 10,11
30 CIRCLE 10,25,10,10,1
40 PAINT 10,25,1:FCOL 8,0,10,10,2
50 LINE 25,15,45,15,1
60 DRAW TO 45,35,1
70 DRAW TO 25,35,1
80 DRAW TO 25,15,1
90 PAINT 26,16,1:FCOL 7,12,12,12,1
100 LINE 50,35,70,35,1
110 DRAW TO 60,15,1
120 DRAW TO 50,35,1
130 PAINT 60,30,1:FCOL 7,25,12,12,6

TYPE: RUN <RETURN>

RESULT: A red circle, white square, and blue triangle are displayed on the low-resolution screen.

NOTE

If the low-resolution graphics mode has been initialized (through LOW RES 10), then LOW RES 0 must be used to terminate low-resolution plotting, or before using HIRES.

7.2.2 PREPARE

FORMAT: PREPARE x

PURPOSE: To prepare a low-resolution screen for storage in another part of the 64's memory.

This command, used in direct mode, prepares a low-resolution screen, text or graphics, for storage in one of four buffers behind KERNAL. The parameter x, which must be a number between 0 and 3, specifies which buffer the information is sent to. Once the command has been initialized, text or graphics may be typed anywhere on the screen. You can use the cursor keys to move around screen, and change colour by using the colour keys. When your screen is ready, press <CTRL-A> to send it to the selected buffer. You are then returned to direct mode.

NOTE

For purposes of data integrity, the <INST/DEL> key has been disabled in PREPARE mode. PREPARE must NOT be used in MEM mode.

EXAMPLE: To prepare a low-resolution screen for storage in the first buffer behind KERNAL:

TYPE: PREPARE 0 <RETURN>

RESULT: Design your own screen using text and graphics. When it is ready, press <CTRL-A> to store it in the first buffer behind KERNAL. Type PULL 0 <RETURN> to retrieve your screen.

7.2.3 PUSH

FORMAT: PUSH x

PURPOSE: To store the current screen in another part of the 64's memory.

This command can be used as part of a program to store all information currently on screen in a buffer behind KERNAL. It is essentially the same as PREPARE. The parameter x, which must be a number between 0 and 3, designates which buffer the screen will be sent to. When the screen has been stored, program execution continues.

EXAMPLE: To save a screen to the first buffer behind KERNAL from within a program:

ENTER: 10 PRINT CHR\$(147)
 20 COLOUR 5,15:LOW RES 10,0
 30 CIRCLE 40,25,10,10,1
 40 PUSH 0
 50 PRINT CHR\$(147)
 60 PAUSE 2
 70 PULL 0

TYPE: RUN <RETURN>

RESULT: A black circle is drawn on the low-resolution screen. The screen is saved, and retrieved from the first buffer behind KERNAL.

7.2.4 PULL

FORMAT: PULL x

PURPOSE: To retrieve a low-resolution screen from another part of the 64's memory.

This command, which can be used either in direct mode, or as part of a program, is used to retrieve a low-resolution screen from one of the four buffers behind KERNAL. The parameter x, specifies which buffer the screen will be retrieved from.

NOTE

When the PULL command is executed, the current screen will be cleared before the retrieved one is displayed.

EXAMPLE: To retrieve a low-resolution screen from within a program:

ENTER: 10 PRINT CHR\$(147)
 20 COLOUR 5,15:LOW RES 10,0
 30 CIRCLE 40,25,10,10,1
 40 PUSH 0
 50 PRINT CHR\$(147)
 60 PAUSE 2
 70 PULL 0

TYPE: RUN <RETURN>

RESULT: A black circle is drawn on the low-resolution screen. The screen is saved, and retrieved from the first buffer behind KERNAL.

7.3 SCREEN FUNCTIONS

7.3.1 FONT

FORMAT: FONTn

PURPOSE: To select one of the two built-in character sets of SIMONS' BASIC EXTENSION.

FONT, similar to the MEM command in SIMONS' BASIC downloads one of the two SIMONS' BASIC EXTENSION character sets. The parameter n, which must be either 1 or 2, specifies which character set is selected. FONT1 downloads the first 128 characters normally, then the next 128 characters are loaded such that the first 64 are the upper parts of the non-inverse characters, and the second 64 are the lower parts. Using FONT1 in conjunction with UPPER (see next section) allows double-sized characters in normal text modes. FONT2 downloads a futuristic, computer-style character set. The FONT command may be used either in direct mode, or as part of a program. The NRM command returns you to the standard character set.

NOTE

Because there are no inverse characters in the FONT 1 character set, the cursor does not appear in this mode. There is no space between the command FONT, and the parameter, n.

EXAMPLE: To display the numbers, letters, and symbols of the futuristic character set:

ENTER: 10 FONT2
20 FOR I=32TO90
30 PRINT CHR\$(I);
40 NEXT I
50 PAUSE 5
60 NRM

TYPE: RUN <RETURN>

RESULT: The numbers, letters, and symbols of the futuristic character set are displayed.

7.3.2 UPPER

FORMAT: UPPER "text"

or: UPPER text\$

PURPOSE: To print double-sized characters on the screen.

The UPPER command, when used in conjunction with FONT1 allows double-sized text to be printed on the screen. When in FONT1 mode, UPPER is used instead of PRINT to display double-sized characters.

NOTE

Cursor commands, HOME, CLR, DEL, INST, are ignored when printing. Care should be taken to ensure that the string is not going to overflow onto the next line.

SIMONS' BASIC EXTENSION

EXAMPLE: To use UPPER in conjunction with FONT1 to display double height text:

ENTER: 10 FONT1
20 UPPER "THIS TEXT":PRINT
30 UPPER "IS":PRINT
40 UPPER "DOUBLE HEIGHT"
50 PAUSE 5
60 NRM

TYPE: RUN <RETURN>

RESULT: The message "THIS TEXT IS DOUBLE HEIGHT" is displayed using double height letters.

7.3.3 ECOL

FORMAT: ECOL In,ac,x+,y+,col1,col2

PURPOSE: To exchange one colour for another on a low-resolution screen.

The ECOL command allows you to exchange one colour for another in a specified area of a low-resolution screen. The first two parameters, In and ac specify the top left coordinates of the screen area to be checked. The next parameter, x+ is the distance between the top left corner, and the top right corner. The parameter y+, is the distance between the top left corner, and the bottom left corner. Col1 is the colour to be exchanged, and col2 is the colour that it will be exchanged for.

EXAMPLE: To draw, and paint a circle in dark grey, and exchange it for red:

ENTER: 10 COLOUR 5,15:LOW RES 10,11
20 CIRCLE 40,25,10,10,1
30 PAINT 40,25,1
40 ECOL 0,0,40,25,11,2

TYPE: RUN <RETURN>

RESULT: A circle is drawn, and painted in dark grey, and then exchanged for red.

7.3.4 CCOL

FORMAT: A=CCOL

or: PRINT CCOL

PURPOSE: To determine the current cursor colour.

The CCOL command returns the current cursor colour, the colour currently being used for all print statements.

EXAMPLE: To use the CCOL command to determine the current cursor colour:

ENTER: 10 PRINT CHR\$(28)
 20 LOW COL 2,0,0
 30 PRINT "COLOUR=";CCOL
 40 PRINT CHR\$(30)
 50 PRINT "COLOUR=";CCOL

TYPE: RUN <RETURN>

DISPLAY: COLOUR= 2 (in red)
 COLOUR= 5 (in green)

7.3.5 SCOL

FORMAT: A=SCOL(x,y)

or: PRINT SCOL(x,y)

PURPOSE: To determine the colour at a specified screen location.

The SCOL command returns the colour at a specified screen location. The parameter x, is the number of columns across, and the parameter y, is the number of rows down.

EXAMPLE: To clear the screen, set the colour to green, and determine the colour at screen location 0,0:

ENTER: 10 PRINT CHR\$(147);CHR\$(30)
 20 COLOUR 5,0
 30 PRINT "THIS IS GREEN"
 40 PRINT "THE COLOUR AT 0,0=";SCOL(0,0)

TYPE: RUN <RETURN>

DISPLAY: THIS IS IN GREEN (in green)
 THE COLOUR AT 0,0= 5 (in green)

7.3.6 SCHR

FORMAT: A=SCHR(x,y)

or: PRINT SCHR(x,y)

PURPOSE: To determine the character at a specified screen location.

The SCHR command returns the poke code (see your COMMODORE 64 user's manual, pages 132-134) of the character at a specified screen location. The parameter x, is the number of columns across, and the parameter y, is the number of rows down. Note that if there is no character at the specified location, then the number 32, the poke code for a space, is returned.

EXAMPLE: To clear the screen, set the colour to green, and determine the poke code at screen location 0,0:

ENTER: 10 PRINT CHR\$(147);CHR\$(30);
20 COLOUR 5,0
30 PRINT "THIS IS GREEN"
40 CH=SCHR(0,0)
50 PRINT "THE POKE CODE AT 0,0=";CH

TYPE: RUN <RETURN>

DISPLAY: THIS IS IN GREEN (in green)
THE POKE CODE AT 0,0= 20 (in green)

7.3.7 MCOL

FORMAT: MCOL col1,col2

PURPOSE: To initialize the multi-colour mode, and select two additional colours.

This command sets up the multi-colour character mode, and selects two additional colours. In this mode, each character may be standard, or multi-colour. This is selected by the colour nybble, if bit 3 is set, then the character is displayed in multi-colour. In practice this means that a CTRL colour is standard, and a CBM colour is multi-colour. Col1 and col2 are two numbers in the range 0-15, and specify the two colours which in addition to the current cursor colour, make up a multi-colour character.

EXAMPLE: To use MCOL in conjunction with DESIGN 3 to design a multi-colour character:

ENTER: 10 MEM
 20 DESIGN 3,\$E000+0*8
 30 @BBBB
 40 @BBBB
 50 @BBBB
 60 @CCCC
 70 @CCCC
 80 @DDDD
 90 @DDDD
 100 @DDDD
 110 MCOL 2,1
 120 PRINT CHR\$(154):REM *** LIGHT BLUE CURSOR ***
 130 PRINT "@@@@@@@@@@@@@@"

TYPE: RUN <RETURN>

RESULT: A line of red, white, and blue is printed at the top of the screen.

7.3.8 DESIGN

FORMAT: DESIGN 3,\$E000+ch*8

PURPOSE: To design multi-colour characters.

The DESIGN command, implemented in SIMONS' BASIC, may be used in conjunction with MCOL (see previous section), and enables you to design your own multi-colour characters. Ch is the poke code of the character to be redesigned. The line containing the DESIGN command is followed by eight lines of four characters. All the characters on one line MUST be the same, and determine the structure of the multi-colour character.

- A or . displays the background colour
- B displays col1 (from MCOL)
- C displays col2 (from MCOL)
- D displays the current cursor colour

NOTE

As in MCOL, CTRL colours are standard, and CBM colours are multi-colour. This means that if CTRL BLK is selected, normal black characters are printed. If CBM BLK is selected, multi-colour characters are printed, with black being colour D. As in the other DESIGN applications, ALL lines of the design grid must begin with the @ symbol.

EXAMPLE: To use DESIGN in conjunction with MCOL to design a multi-colour character:

SIMONS' BASIC EXTENSION

ENTER: 10 MEM
 20 DESIGN 3,\$E000+0*8
 30 @BBBB
 40 @BBBB
 50 @BBBB
 60 @CCCC
 70 @CCCC
 80 @DDDD
 90 @DDDD
 100 @DDDD
 110 MCOL 2,1
 120 PRINT CHR\$(154):REM *** LIGHT BLUE CURSOR ***
 130 PRINT "@@@@@@@@@@@@@@@"

TYPE: RUN <RETURN>

RESULT: A line of red, white, and blue is printed at the top of the screen.

7.3.9 ROTATE

FORMAT: ROTATE \$E000+ch*8

or: ROTATE \$E000+ch*8,\$E000+an*8

PURPOSE: To rotate a group of eight bytes.

The ROTATE command rotates a character, a group of eight bytes clockwise through 90 degrees. The address is the same as the one used for DESIGN; the parameter ch is the poke code for the character you wish to rotate. There are two types of format. The first rotates the character clockwise through 90 degrees, and stores it back in its original address. The second also rotates the character clockwise through 90 degrees, but instead of returning it to its original address, places it in another address. The parameter an, specifies which address the rotated character will be placed in.

NOTE

When using the second type of format, it is important to remember that the new rotated character overwrites the character whose poke code is an. However, this is only temporary and you may return to the normal character set by pressing <RESTORE>. ROTATE can only be used in MEM mode.

EXAMPLE: To rotate the letters of the alphabet through 180 degrees:

ENTER: 10 MEM
 20 PRINT "ABCDEFGHJKLMNOPQRSTUVWXYZ"
 30 FOR I=1TO26
 40 ROTATE \$E000+I*8
 50 ROTATE \$E000+I*8
 60 NEXT I
 70 PAUSE 5
 80 NRM

TYPE: RUN <RETURN>

RESULT: The letters of the alphabet are displayed upside-down.

7.3.10 BCKFLASH

FORMAT: BCKFLASH speed, col1, col2

or: BCKFLASH 0

PURPOSE: To flash, or stop flashing two background colours.

The BCKFLASH command allows you to flash two background colours. The first parameter, speed, must be a number between 1 and 255, and determines the speed at which the flashing takes place, 1 being the fastest, and 255 the slowest. The next two parameters, col1 and col2, must be two integers in the range 0-15, and specify the two colours which are flashed. BCKFLASH 0 turns background flashing off, without affecting sprite handling. The BCKFLASH command proves more useful than FLASH (see Section 7.3 in your SIMONS' BASIC manual) in certain situations, because speed of execution is not affected so badly.

NOTE

You must call INIT (see Section 8.3.8) before background flashing can take place. The BCKFLASH command can only be used as part of a program.

EXAMPLE: To flash two background colours:

ENTER: 10 INIT
 20 BCKFLASH 8,14,6
 30 PAUSE 10

TYPE: RUN <RETURN>

RESULT: The background colour flashes alternatively between light blue, and dark blue for ten seconds.

7.3.11 *

FORMAT: ★any command

PURPOSE: To direct the output from any command to the printer.

An asterisk inserted before any command sends the output from that command to the printer, thus eliminating the need to OPEN a channel, and send output via the CMD command. This command can be used either in direct mode, or as part of a program.

EXAMPLE: To send the output from PRINT and LIST commands to the printer:

ENTER: 10 ★PRINT "THIS PROGRAM USES THE '★' COMMAND"
 20 ★PRINT "TO SEND COMMANDS TO THE PRINTER"
 30 ★LIST

TYPE: RUN <RETURN>

RESULT: The message:

THIS PROGRAM USES THE '★' COMMAND
TO SEND COMMANDS TO THE PRINTER

and a listing of the program are printed out on the printer.

SECTION EIGHT

SPRITE MANIPULATION COMMANDS

8.1 INTRODUCTION

This section contains additional commands for use with sprites.

8.2 SPRITES

8.2.1 INTRODUCTION

For more information on sprites, see Section 8 of your SIMONS' BASIC MANUAL.

8.2.2 WHAT IS A VECTOR DRIVEN SPRITE?

SIMONS' BASIC EXTENSION takes the sprite commands implemented in SIMONS' BASIC one step further, by introducing VECTOR DRIVEN SPRITES.

A vector quantity is something that has both direction and magnitude. The SPRITE command allows you to set up the vectors for each sprite, direction and magnitude, or speed. However, the SPRITE command does not end at vectors. You can set up four barriers, low x, low y, high x, high y, and specify what your sprite does when it reaches one.

Having set up your sprites, you can START them moving at exactly the same time. Using BARRIER, you can change x and y flags, and using CHANGE, you can alter the enab register, which controls sprite movement, and detects sprite/barrier collisions. CLEAR enables you to stop one or more sprites at the same time, and the INFO command returns the status of a specific sprite. SPRX and SPRY respectively return the coordinates of a selected sprite, and SPR LOC tells you the start location of your sprite data in RAM. The INVERT command turns a sprite upside-down, and REVERSE reverses a sprite.

Using ON DETECT you can check for sprite/text collisions, sprite/sprite collisions, or both. You also specify a line, which the program jumps to in the event of a collision. The CONTINUE command is used at the end of sprite collision routines, it returns program execution to where it was before the collision took place.

The SHOW command can be used to examine the sprite data in a specific memory location, or to set up the sprite design grid.

The INIT command sets the sprite handler into operation, and NORMAL disables it.

Finally, TRANSFER and CREATE. The TRANSFER command transfers sprite data to high-resolution data, and the CREATE command creates sprite data from high-resolution data. Using TRANSFER and CREATE, you can create the illusion that there are many more than eight sprites on screen.

8.2.3 WHAT IS A FRAME?

Most televisions and monitors work along the "raster scan" principle. Imagine a dot that starts off in the top left-hand corner of the screen, and moves horizontally to the top right-hand corner of the screen. When it reaches the right-hand edge of the screen, it simultaneously jumps down a line, and returns to the left-hand side of the screen. It then moves horizontally to the right-hand edge of the screen again. In this way the dot continues until it eventually reaches the bottom right-hand corner of the screen. Once it has reached the bottom right-hand corner of the screen, the dot jumps up to the top left-hand corner, and starts all over again. This dot is the raster beam.

The reason we don't see the raster beam is that it moves incredibly fast. As the raster moves across and down the screen, it receives information from the video chip. This information tells the raster beam which parts of each line, or raster, to "light up", in this way we can see that a television picture is made up of many horizontal lines, or rasters.

One complete "frame" is the time that it takes for the raster beam to travel from the top left-hand corner of the screen, to the bottom right-hand corner of the screen. There are approximately 60 frames a second!

With SIMONS' BASIC EXTENSION, all sprites are moved while the raster beam is off the screen. Because of this, sprite movement (even with eight sprites moving at once) is very smooth, and flicker-free.

8.3 SPRITE MANIPULATION COMMANDS

8.3.1 SPRITE

FORMAT: SPRITE spr,xvec,yvec,sp

or: SPRITE spr,xvec,yvec,sp,lox,loy,hix,hiy,flx,fly,enab

PURPOSE: To set up a sprite vector, speed, barriers, sprite flags, and an enab register.

The SPRITE command sets up a sprite vector, speed, up to four barriers, sprite flags (which determine what happens in the event of a sprite/barrier collision), and the enab register (which checks for sprite/barrier collisions).

The first parameter, `spr`, is the sprite number. It is unique for each sprite, and is set up in the `MOB SET` command (see Section 8.2.5 of your `SIMONS' BASIC MANUAL`). The next two parameters, `xvec` and `yvec`, are the two elements of the sprite vector. The parameter `xvec`, determines how many pixels the sprite moves along the x-axis each time it is updated, and the parameter `yvec`, determines how many pixels the sprite moves along the y-axis each time it is updated. The next parameter, `sp`, is the speed. This dictates the frequency with which sprite updates take place, a "1" in this position means that the sprite is moved every frame, a "2" in this position means that the sprite is moved every other frame, and so on. `lox`, `loy`, `hix`, and `hiy` are the four barriers, `lox` and `loy` are the low barriers of the x and y axes, and `hix` and `hiy` are the high barriers of the x and y axes. The next two parameters, `flx` and `fly`, specify what happens in the event of a sprite/barrier collision. The commands are as follows:

- 1 - turn sprite off
- 2 - set `INFO(spr)` (see Section 8.3.8)
- 4 - reverse `xvec`
- 8 - reverse `yvec`
- 16 - stop sprite

NOTE

If you require two or more of the above functions, add their respective numbers together, i.e. to reverse both `xvec` and `yvec`, use "12".

The final parameter of the `SPRITE` command is `enab`, this controls sprite movement, and checks for sprite/barrier collisions. The commands are as follows:

- 1 - move sprite, this sets a sprite moving. But, you may not want this to happen. To start a number of sprites moving at the same time, see Section 8.3.4.
- 2 - do x comparison, this checks to see if either the `lox` or `hix` barriers have been crossed. If either of the x barriers are crossed, then control is transferred to the `flx` register. If neither of the x barriers is crossed, then the sprite continues along its vector.
- 4 - do y comparison, this checks to see if either the `loy` or `hoy` barriers have been crossed. If either of the y barriers are crossed, then control is transferred to the `fly` register. If neither of the y barriers is crossed, then the sprite continues along its vector.

NOTE

As with flx and fly, if you require two or more functions, add their respective numbers together, i.e. to do both x and y comparisons, use "6". If a "0" is put in the enab register, then the sprite in question travels along its vector with total wrap-around, i.e. if a sprite goes off one side of the screen, it reappears in the same position on the opposite side.

EXAMPLE: To use the SPRITE command to set up a sprite vector, speed, barriers, sprite flags, and an enab register:

```

ENTER:  10 PRINT CHR$(147)
        20 INIT
        30 MOB OFF 0
        40 MOB OFF 1
        50 DESIGN 1,240*64
        60 @.....
        70 @..BBBBB.....
        80 @.BBBBBB.....
        90 @.BBBBBB.....
        100 @BBBBBBB.....
        110 @BBBB...BBBBB
        120 @BBB...BBBB.
        130 @BBB...BBB..
        140 @BBB...BB...
        150 @BBB.....
        160 @BBB...CC...
        170 @BBB...CCC..
        180 @BBB...CCCC.
        190 @BBBB...CCCCC
        200 @BBBBBBB.....
        210 @.BBBBBB.....
        220 @.BBBBBB.....
        230 @..BBBBB.....
        240 @.....
        250 @.....
        260 @.....
        270 MOB SET 0,240,2,0,1
        280 CMOB 6,2
        290 RLOCMOB 0,100,100,0,1
        300 SPRITE 0,2,2,1,23,49,323,231,4,8,7
        310 PAUSE 5
    
```

TYPE: RUN <RETURN>

RESULT: The sprite moves around the screen diagonally, and bounces off the screen border.

NOTE

If the first type of format (i.e. SPRITE spr,xvec,yvec,sp) is used, then all the other parameters in the SPRITE command are set to zero.

8.3.2 XVEC

FORMAT: A=XVEC(spr)

or: PRINT XVEC(spr)

PURPOSE: To read the x vector.

The XVEC command returns the x vector of sprite spr.

EXAMPLE: To read the x and y vectors:

```

ENTER:      10 PRINT CHR$(147)
            20 INIT
            30 MOB OFF 0
            40 MOB OFF 1
            50 DESIGN 1,240*64
            60 @.....
            70 @.BBBBB.....
            80 @.BBBBBB.....
            90 @.BBBBBB.....
           100 @BBBBBBB.....
           110 @BBBBB...BBBBB
           120 @BBB...BBBB.
           130 @BBB...BBB..
           140 @BBB...BB...
           150 @BBB.....
           160 @BBB...CC...
           170 @BBB...CCC..
           180 @BBB...CCCC.
           190 @BBBB...CCCCC
           200 @BBBBBBB.....
           210 @.BBBBBB.....
           220 @.BBBBBB.....
           230 @.BBBBB.....
           240 @.....
           250 @.....
           260 @.....
           270 MOB SET 0,240,2,0,1
           280 CMOB 6,2
           290 RLOCMOB 0,100,100,0,1
           300 SPRITE 0,2,2,1,23,49,323,231,6,10,7
           310 PRINT "XVEC=";XVEC(0),"YVEC=";YVEC(0)
           320 GOTO310

```

TYPE: RUN <RETURN>

RESULT: As the sprite bounces around the screen, its x and y vectors are displayed.

8.3.3 YVEC

FORMAT: A=YVEC(spr)

or: PRINT YVEC(spr)

PURPOSE: To read the y vector.

The YVEC command returns the y vector of sprite spr.

EXAMPLE: See example program in Section 8.3.2.

8.3.4 BARRIER

FORMAT: BARRIER spr,lox,loy,hix,hiy,flx,fly,enab

PURPOSE: To change the barriers.

The BARRIER command allows you to change the barriers whilst a program is running. Spr is the sprite number, and the next four parameters, lox, loy, hix, and hiy, are the barriers. Flx and fly are the x and y flags, and enab is the value for the enab register.

EXAMPLE: To change the sprite barriers whilst the program is running:

ENTER: 320 RLOCMOB 0,180,150,0,1
330 BARRIER 0,103,123,243,151,4,8,7
340 PAUSE 5

TYPE: RUN <RETURN>

RESULT: After a pause of five seconds, the sprite barriers are altered such that the sprite is confined to the central part of the screen.

8.3.5 CHANGE

FORMAT: CHANGE spr,enab

PURPOSE: To change the enab register.

The CHANGE command allows you to alter the enab register whilst a program is running. Spr is the sprite number, and the parameter enab, is the enab register. This controls sprite movement, and checks for sprite/barrier collisions. The commands are as follows:

- 1 - move sprite, this sets a sprite moving. But, you may not want this to happen. To start a number of sprites moving at the same time, see Section 8.3.4.
- 2 - do x comparison, this checks to see if either the lox or hix barriers have been crossed. If either of the x barriers are crossed, then control is transferred to the flx register. If neither of the x barriers is crossed, then the sprite continues along its vector.
- 4 - do y comparison, this checks to see if either the loy or hiy barriers have been crossed. If either of the y barriers is crossed, then control is transferred to the fly register. If neither of the y barriers is crossed, then the sprite continues along its vector.

EXAMPLE: To change the enab register whilst the program is running:

ENTER: 350 CHANGE 0,5
360 PAUSE 5

TYPE: RUN <RETURN>

RESULT: After a pause of five seconds, the enab register is altered such that the x comparison no longer takes place.

8.3.6 START

FORMAT: START spr

or: START spr,spr

PURPOSE: To set one or more sprites into motion.

This command sets one or more sprites moving along their vectors. Spr is the the number of the sprite that you wish to set into motion, for example, to start sprites 0, 5, and 6 moving at the same time, use START 0,5,6. The START command ensures that all the sprites specified start moving at exactly the same time. Using START with no parameters, sets all the sprites that have been set up into motion. Up to eight sprites may be started at any one time.

NOTE

In order to utilise the START command, the enab register in the SPRITE command must not include the move sprite command, a "1".

EXAMPLE: To set up a second sprite, identical to the first, and start them both moving at the same time:

ENTER: 370 MOB SET 1,240,2,1,1
380 RLOCMOB 0,180,150,0,1
390 RLOCMOB 1,180,150,2.1
400 SPRITE 1,2,-2,1,103,129,243,151,4,8,6
410 START 0,1
420 PAUSE 5

TYPE: RUN <RETURN>

RESULT: After a pause of five seconds, a second sprite is set up. Both sprites are relocated, and set into motion at the same time.

8.3.7 CLEAR

FORMAT: CLEAR spr

or: CLEAR spr,spr

PURPOSE: To stop one or more sprites.

This command stops one or more sprites. Spr is the number of the sprite that you wish to stop, for example, to stop sprites 0, 5, and 6 all at the same time, use CLEAR 0,5,6. The CLEAR command ensures that all the sprites specified stop moving at exactly the same time. Using CLEAR with no parameters, stops all the sprites that are currently moving along their vectors. Up to eight sprites may be stopped at any one time.

EXAMPLE: To stop both sprites at the same time:

ENTER: 430 CLEAR 0,1
440 PAUSE 5

TYPE: RUN <RETURN>

RESULT: After a pause of five seconds, both sprites are stopped at the same time.

8.3.8 INIT

FORMAT: INIT

PURPOSE: To set the sprite handler into operation.

INIT sets the sprite handler into operation, and also sets INFO (0-7) to zero. INIT must be called before any of the SIMONS' BASIC EXTENSION sprite commands can be used. The INIT command also enables background flashing to take place, although flashing does not start until BCKFLASH is used, see Section 7.3.10.

WARNING

Sprites must not be in motion when INIT is called. If there is any possibility that the sprites are moving, use CLEAR 0,1,2,3,4,5,6,7 before calling INIT.

EXAMPLE To initialize the sprite handler, and set up a sprite:

ENTER: See example program in Section 8.3.1

TYPE: RUN <RETURN>

RESULT: Because the sprite handler has been initialized, all SIMONS' BASIC EXTENSION sprite commands now function correctly.

8.3.9 NORMAL

FORMAT: NORMAL

PURPOSE: To disable the sprite handler.

NORMAL disables the SIMONS' BASIC EXTENSION sprite handler, and should be used whenever VECTOR DRIVEN SPRITES are no longer required. This command is executed automatically whenever you leave a program.

EXAMPLE: To disable the sprite handler:

ENTER: 450 NORMAL
460 PRINT "SPRITE HANDLER DISABLED"
470 PAUSE 5

TYPE: RUN <RETURN>

RESULT: After a pause of five seconds, the sprite handler is disabled.

8.3.10 INFO

FORMAT: A=INFO(spr)

or: PRINT INFO(spr)

PURPOSE: To determine the status of a selected sprite.

The BASIC variable INFO is initially set to zero. If the sprite collides with a boundary, then INFO is set to one. It is cleared as soon as it is read. The single parameter, spr, is the sprite number.

NOTE

In order to be able to read the BASIC variable INFO, you must first set it in the x and y flags, which are part of the SPRITE command, see Section 8.3.1.

EXAMPLE: To return the status of sprite 0:

```

ENTER:  10 PRINT CHR$(147)
        20 INIT
        30 MOB OFF 0
        40 MOB OFF 1
        50 DESIGN 1,240*64
        60 @.....
        70 @.BBBBB.....
        80 @.BBBBBB.....
        90 @.BBBBBB.....
        100 @BBBBBBB.....
        110 @BBBB...BBBBB
        120 @BBB...BBBB.
        130 @BBB...BBB..
        140 @BBB...BB...
        150 @BBB.....
        160 @BBB...CC...
        170 @BBB...CCC..
        180 @BBB...CCCC.
        190 @BBBB...CCCCC
        200 @BBBBBBB.....
        210 @.BBBBBB.....
        220 @.BBBBBB.....
        230 @.BBBBBB.....
        240 @.....
        250 @.....
        260 @.....
        270 MOB SET 0,240,2,0,1
        280 CMOB 6,2
        290 RLOCMOB 0,100,100,0,1
        300 SPRITE 0,2,2,1,23,49,323,231,6,10,7
        310 PRINT "SPRITE INFO";INFO(0)
        320 GOTO310

```

TYPE: RUN <RETURN>

RESULT: The sprite's status is returned. Normally it is 0, but changes to 1 when the sprite collides with a boundary.

8.3.11 SPRX

FORMAT: A=SPRX(spr)

or: PRINT SPRX(spr)

PURPOSE: To determine the x position of a selected sprite.

The SPRX command returns the x position of a selected sprite. The parameter spr is the sprite number.

NOTE

Even though sprite movement may actually be taking place on a low-resolution or multi-colour screen, all sprite coordinates are returned as if on a high-resolution screen, i.e. x coordinates are on a scale of 0-320, and y coordinates are on a scale of 0-200. However, in most situations the scaling deviates slightly from the standard high-resolution screen limits, i.e. for an unexpanded high-resolution sprite, the x coordinates are on a scale of 23-323, and the y coordinates are on a scale of 49-231.

EXAMPLE: To return the x and y positions of the sprite:

```
ENTER: 10 PRINT CHR$(147)
        20 INIT
        30 MOB OFF 0
        40 MOB OFF 1
        50 DESIGN 1,240*64
        60 @.....
        70 @..BBBBB.....
        80 @.BBBBBB.....
        90 @.BBBBBB.....
        100 @BBBBBBB.....
        110 @BBBBB...BBBBB
        120 @BBB...BBBB.
        130 @BBB...BBB..
        140 @BBB...BB...
        150 @BBB.....
        160 @BBB...CC...
        170 @BBB...CCC..
        180 @BBB...CCCC.
        190 @BBBB...CCCCC
        200 @BBBBBBB.....
```

```
210 @.BBBBBB.....
220 @.BBBBBB.....
230 @..BBBBBB.....
240 @.....
250 @.....
260 @.....
270 MOB SET 0,240,2,0,1
280 CMOB 6,2
290 RLOCMOB 0,100,100,0,1
300 SPRITE 0,2,2,1,23,49,323,231,6,10,7
310 PRINT "X=";SPRX(0),"Y=";SPRY(0)
320 GOTO310
```

TYPE: RUN <RETURN>

RESULT: As the sprite bounces around the screen, its x and y positions are displayed.

8.3.12 SPRY

FORMAT: A=SPRY(spr)

or: PRINT SPRY(spr)

PURPOSE: To determine the y position of a selected sprite.

The SPRY command returns the y position of a selected sprite. The parameter spr is the sprite number.

NOTE

Even though sprite movement may actually be taking place on a low-resolution or multi-colour screen, all sprite coordinates are returned as if on a high-resolution screen, i.e. x coordinates are on a scale of 0-320, and y coordinates are on a scale of 0-200. However, in most situations the scaling deviates slightly from the standard high-resolution screen limits, i.e. for an unexpanded high-resolution sprite, the x coordinates are on a scale of 23-323, and the y coordinates are on a scale of 49-231.

EXAMPLE: See example program in Section 8.3.12

8.3.13 SPR LOC

FORMAT: A=SPR LOC(spr)

or: PRINT SPR LOC(spr)

PURPOSE: To determine the start location of the sprite data for a selected sprite.

The SPR LOC command returns the start location of the sprite data for the selected sprite. The parameter spr is the sprite number, i.e. if sprite 0 is set to block 32, then PRINT SPR LOC(0) returns 2048.

EXAMPLE: To return the start location of the sprite data in memory:

```

ENTER:      10 PRINT CHR$(147)
            20 INIT
            30 MOB OFF 0
            40 MOB OFF 1
            50 DESIGN 1,240*64
            60 @.....
            70 @..BBBBB.....
            80 @.BBBBBB.....
            90 @.BBBBBB.....
            100 @BBBBBBB.....
            110 @BBBB...BBBBB
            120 @BBB...BBBB.
            130 @BBB...BBB..
            140 @BBB...BB...
            150 @BBB.....
            160 @BBB...CC...
            170 @BBB...CCC..
            180 @BBB...CCCC.
            190 @BBBB...CCCCC
            200 @BBBBBBB.....
            210 @.BBBBBB.....
            220 @.BBBBBB.....
            230 @..BBBBB.....
            240 @.....
            250 @.....
            260 @.....
            270 MOB SET 0,240,2,0,1
            280 CMOB 6,2
            290 RLOCMOB 0,100,100,0,1
            300 SPRITE 0,2,2,1,23,49,323,231,6,10,7
            310 PRINT "SPRITE BLOCK=";SPR LOC(0)
            320 PAUSE 5
    
```

TYPE: RUN <RETURN>

DISPLAY: SPRITE BLOCK= 240

8.3.14 INVERT

FORMAT: INVERT A

or: INVERT 8192

or: INVERT \$2000

PURPOSE: To invert a sprite.

The INVERT command turns a sprite upside-down. The number following the INVERT command must be the address of the first byte of the sprite data, the same address as in DESIGN, see Section 8.2.2 of your SIMONS' BASIC MANUAL.

EXAMPLE: To invert the sprite while it is in motion:

```

ENTER:        10 PRINT CHR$(147)
              20 INIT
              30 MOB OFF 0
              40 MOB OFF 1
              50 DESIGN 1,240*64
              60 @.....
              70 @.....
              80 @.....
              90 @.....
             100 @.....
             110 @...CCCCCCDD
             120 @...CCCCCCDD
             130 @.CCCCCCDDDD
             140 @.CCCCCCDDDD
             150 @BBBBBBBBDDDD
             160 @BBBBBBBBDDDD
             170 @BBBBBBBBDDDD
             180 @BBBBBBBBDDDD
             190 @BBBBBBBBDDDD
             200 @BBBBBBBBDDDD
             210 @BBBBBBBBDDDD
             220 @BBBBBBBBDDDD
             230 @BBBBBBBBDDDD.
             240 @BBBBBBBBDDDD..
             250 @BBBBBBBBDDDD...
             260 @BBBBBBBBDDDD....
             270 MOB SET 0,240,2,0,1
             280 CMOB 6,5
             290 RLOCMOB 0,100,100,0,1
             300 SPRITE 0,2,2,1,23,49,323,231,6,10,7
             310 INVERT SPR LOC(0)*64
             320 PAUSE 3
             330 GOTO310

```

TYPE: RUN <RETURN>

RESULT: The sprite bounces around the screen, inverting every three seconds.

8.3.15 REVERSE

FORMAT: REVERSE A

or: REVERSE 8192

or: REVERSE \$2000

PURPOSE: To reverse a sprite.

The REVERSE command reverses a sprite. The number following the REVERSE command must be the address of the first byte of sprite data, the same address as in DESIGN, see Section 8.2.2 of your SIMONS' BASIC MANUAL. The reversal procedure uses an exclusive-or, i.e., the following happens:

HIGH-RESOLUTION SPRITES

MULTI-COLOUR SPRITES

dot on - dot off
dot off - dot on

colour0 - colour3
colour1 - colour2
colour2 - colour1
colour3 - colour0

EXAMPLE: To reverse the sprite:

ENTER: 310 REVERSE SPR LOC(0)*64

TYPE: RUN <RETURN>

RESULT: The sprite bounces around the screen, reversing every three seconds.

8.3.16 SHOW

FORMAT: SHOW addr

or: SHOW addr,line

or: SHOW addr,line,inc

or: SHOW addr,line,inc,mcs

PURPOSE: To show the sprite data at a particular address, or to set up a sprite design grid.

The SHOW command shows the sprite data at the specified address. The first parameter, *addr* is the address of the first byte of sprite data, the same address as in DESIGN, see Section 8.2.2 of your SIMONS' BASIC MANUAL. The sprite data is taken from memory, and displayed using "@", ".", and "B". Spaces are left for you to insert line numbers. Using the second format, where the parameter line is the start line number, the sprite data is displayed with line numbers starting from line in increments of ten. The third format allows you to specify the increment (*inc*), as well as the start line number. If, in the fourth format, the parameter, *mcs*, is anything other than a zero, then the sprite is displayed as a multi-colour one.

EXAMPLE: To show the sprite data:

```
ENTER:      10 PRINT CHR$(147)
            20 INIT
            30 MOB OFF 0
            40 MOB OFF 1
            50 DESIGN 1,240*64
            60 @.....
            70 @.....
            80 @.....
            90 @.....
           100 @.....
           110 @....CCCCCCCCD
           120 @...CCCCCCCCDD
           130 @..CCCCCCCCDDD
           140 @.CCCCCCCCDDDD
           150 @BBBBBBBBBDDDD
           160 @BBBBBBBBBDDDD
           170 @BBBBBBBBBDDDD
           180 @BBBBBBBBBDDDD
           190 @BBBBBBBBBDDDD
           200 @BBBBBBBBBDDDD
           210 @BBBBBBBBBDDDD
           220 @BBBBBBBBBDDDD
           230 @BBBBBBBBBDDDD.
           240 @BBBBBBBBBDDDD..
           250 @BBBBBBBBBDDDD...
           260 @BBBBBBBBBDDDD....
           270 MOB SET 0,240,2,0,1
           280 CMOB 6,5
           290 RLOCMOB 0,180,100,0,1
           300 SHOW SPR LOC(0)*64,1000,10,1
```

TYPE: RUN <RETURN>

RESULT: The multi-colour sprite, and the data used to create it are displayed.

8.3.17 ON DETECT

FORMAT: ON DETECT comm,line

PURPOSE: To check for sprite/sprite, and/or sprite/text collisions.

The ON DETECT command provides continuous detection for sprite/sprite, and/or sprite/text collisions. The first parameter, comm, is the command. This must be a number between zero and three, and specifies the collision type to be checked for. The second parameter, line, is used with command numbers 1, 2, and 3. It specifies the line number that the program jumps to if a collision is detected, just like a GOTO, or GOSUB. The commands are as follows:

- ON DETECT 0 - disables any active ON DETECT.
- ON DETECT 1,line - initializes sprite/sprite collision detection, and specifies a line number that the program will jump to in the event of a collision.
- ON DETECT 2,line - initializes sprite/text collision detection, and specifies a line number that the program will jump to in the event of a collision.
- ON DETECT 3,line1,line2 - initializes sprite/sprite and sprite/text collision detection. If a sprite/sprite collision occurs, then the program will jump to line 1, and if a sprite/text collision occurs, then the program will jump to line 2.

EXAMPLE: To set two sprites into motion, and transfer the sprite data onto the multi-colour screen every time they collide:

SIMONS' BASIC EXTENSION

```
ENTER:      10 HIRES 15,0
            20 MULTI 6,2,5
            30 COLOUR 5,15
            40 SC=$C000
            50 INIT
            60 MOB OFF 0
            70 MOB OFF 1
            80 DESIGN 1,SC+40*64
            90 @.....
           100 @.....
           110 @.....
           120 @.....
           130 @.....
           140 @....CCCCCCCC
           150 @...CCCCCCCC00
           160 @..CCCCCCCC000
           170 @.CCCCCCCC0000
           180 @BBBBBBBB0000
           190 @BBBBBBBB0000
           200 @BBBBBBBB0000
           210 @BBBBBBBB0000
           220 @BBBBBBBB0000
           230 @BBBBBBBB0000
           240 @BBBBBBBB0000
           250 @BBBBBBBB0000
           260 @BBBBBBBB000.
           270 @BBBBBBBB00..
           280 @BBBBBBBB0...
           290 @BBBBBBBB....
           300 MOB SET 0,40,2,0,1
           310 MOB SET 1,40,2,0,1
           320 CMOB 6,5
           330 RLOCMOB0,40,123,0,1
           340 RLOCMOB1,280,123,0,1
           350 SPRITE 0,-3,1,1,23,49,323,231,4,8,7
           360 SPRITE 1,1,-3,1,23,49,323,231,4,8,7
           370 ON DETECT1,1000:DETECT0
           380 GOTO380
           1000 X=(SPRX(0)-23)AND510
           1010 Y=(SPRY(0)-49)AND254
           1020 A=SPR LOC(0)*64+SC
           1030 TRANSFER 1,1,X,Y,A
           1040 CONTINUE
```

TYPE: RUN <RETURN>

RESULT: Every time the two sprites collide, the sprite data is transferred onto the multi-colour screen.

8.3.18 CONTINUE

FORMAT: CONTINUE

PURPOSE: To return program execution to where it was before a collision was detected.

The CONTINUE command returns program execution back to where it was before a collision was detected, just like a RETURN at the end of a sub-routine. CONTINUE should be used at the end of sprite collision routines.

EXAMPLE: To return program execution after a sprite collision routine - See Section 8.3.17 line 1040

RESULT: After the sprite data is transferred onto the multi-colour screen, program execution resumes.

8.3.19 TRANSFER

FORMAT: TRANSFER expx,expy,x,y,addr

PURPOSE: To transfer sprite data to high-resolution data.

The TRANSFER command transfers sprite data to high-resolution data. The first two parameters, expx, and expy, are the x and y expansion factors; a "1" denotes normal size, a "2" denotes double size, etc. The next two parameters, x and y, are the top left-hand coordinates of the area to which the sprite data is to be transferred. X and y must always be coordinates for a high-resolution screen (0-320, and 0-200), even if you are in multi-colour mode. The final parameter, addr, is the start address of the sprite data, the same as that used in DESIGN, see Section 8.2.2 of your SIMONS' BASIC MANUAL.

NOTE

If you are transferring multi-colour sprites to a multi-colour screen, then the colours are converted as follows:

SPRITE CODE	PLOT TYPE
A	0
B	1
C	2
D	3

EXAMPLE: To transfer sprite data onto the multi-colour screen - See Section 8.3.17 line 1030.

RESULT: Every time the two sprites collide, the sprite data is transferred onto the multi-colour screen.

8.3.20 CREATE

FORMAT: CREATE x,y,addr

PURPOSE: To create sprite data from high-resolution data.

The CREATE command creates sprite data from high-resolution data. The first two parameters, x and y, are top left-hand coordinates of the area from which the sprite data is to be created. The final parameter, addr, is the start address of the sprite data, this is where the sprite data will be written to when the CREATE command is executed.

NOTE

Even though the sprite data is written into memory when the CREATE command is executed, you will still have to use the MOB SET command (see Section 8.2.5 of your SIMONS' BASIC MANUAL) if you wish use the sprite.

EXAMPLE: To create sprite data from high resolution data:

```
ENTER: 10 HIRES 15,0
        20 CIRCLE 110,110,10,10,1
        30 PAINT 110,110,1
        40 CREATE 100,100,$C000+40*64
        50 MOB OFF 0
        60 RLOCMOB 0,123,149,0,1
        70 MOB SET 0,40,15,0,0
        80 INIT
        90 SPRITE 0,2,2,1,23,49,323,231,4,8,7
        100 GOTO100
```

TYPE: RUN <RETURN>

RESULT: A circular sprite is created by using the CIRCLE and PAINT commands.

8.4 NOTES ON VECTOR DRIVEN SPRITES

The following commands will stop all vector driven sprites from moving, because they need to access the character ROM:

```
MEM
FONT1
FONT2
TEXT
LABEL
```

SECTION NINE

MUSIC COMMANDS

9.1 INTRODUCTION

This section contains additional music commands; FILTER, MODE, PULSE, and BEEP.

The FILTER command enables you to set up a value for the filter, and MODE sets up the filter mode and resonance. The PULSE command sets up pulse width.

BEEP is for those applications that only require simple sound.

9.2 WHAT IS A FILTER?

A filter is a vital part of any synthesizer. Just like the envelope generator can shape the ADSR of each note, the filter can actually control the sound make-up, or frequency spectrum of each note.

Every sound or waveform is made up of many sine waves. The frequency of the sound is called the fundamental, and the actual make-up of the sound is denoted by its harmonics. This is why a C played on a piano “sounds” different from a C played on a violin. The fundamental is the same, but the harmonic structure, or frequency spectrum is different. Thus, being able to filter out various parts of the frequency spectrum gives us incredible control over the scope of sounds that we can produce.

This technique is known as subtractive synthesis, starting off with a full frequency spectrum, and taking away the parts that we don't want. The opposite of subtractive synthesis is additive synthesis, starting off with just the fundamental frequency, and adding the harmonics.

9.3 MUSIC COMMANDS

9.3.1 FILTER

FORMAT: FILTER value

PURPOSE: To set up a filter value.

The FILTER command sets up a value for the filter cutoff frequency. The single parameter, value, is any number in the range 0-2047.

EXAMPLE: To use FILTER in conjunction with MODE to set up a filter cutoff frequency:

ENTER: 10 VOL 15
20 WAVE 1,01010000
30 ENVELOPE 1,10,10,8,1
40 FILTER \$031A
50 MODE 15,3,7,9
60 MUSIC 9,"<SHIFT CLR/HOME>1<F1>C5<F1>D5<F1>E5
<F1>F5<F1>G5<F1>A5<F1>B5<F1>C6<F1>"
70 PLAY 1
80 GOTO 70

TYPE: RUN <RETURN>

RESULT: A filtered, ascending scale is played.

9.3.2 MODE

FORMAT: MODE resonance,voice,type,volume

PURPOSE: To set up the filter mode, resonance, and volume.

The MODE command sets up filter mode, resonance, and volume for single or multiple voices. The first parameter, resonance, is any number in the range 0-15, and specifies how strongly the band of frequencies at the cutoff point are emphasized. The next parameter, voice, selects which voice, or voices are put through the filter. The voice number is a "1" for voice 1, a "2" for voice 2, and a "4" for voice 3. Multiple voices are specified by adding together the numbers of the voices that you wish to filter. The next parameter, type, denotes the filter type, a "1" specifies the low pass filter (all frequencies or harmonics below the cutoff frequency are allowed to pass through), a "2" specifies the high pass filter (all frequencies or harmonics above the cutoff frequency are allowed to pass through), and a "4" specifies the band pass filter (all frequencies or harmonics around the cutoff frequency are allowed to pass through). The final parameter, volume, is a number in the range 0-15, and controls the output volume. It is the same as VOL (see Section 11.2.1 of your SIMONS' BASIC MANUAL).

WARNING
IF MODE HAS BEEN USED TO SET UP THE
VOLUME, THEN VOL MUST NOT BE USED
AFTER IT.

EXAMPLE: To use MODE in conjunction with FILTER to set up the filter mode, resonance, and volume:

ENTER: 10 VOL 15
 20 WAVE 1,01010000
 30 ENVELOPE 1,10,10,8,1
 40 FILTER \$031A
 50 MODE 15,3,7,9
 60 MUSIC 9,"<SHIFT CLR/HOME>1<F1>C5<F1>D5<F1>E5
 <F1>F5<F1>G5<F1>A5<F1>B5<F1>C6<F1>"
 70 PLAY 1
 80 GOTO 70

TYPE: RUN <RETURN>

RESULT: A filtered, ascending scale is played.

9.3.3 PULSE

FORMAT: PULSE voice,value.

PURPOSE: To set up pulse width.

The PULSE command sets up pulse width for a voice, or number of voices. The first parameter, voice, is the voice number, a "1", "2", or "3". To set up pulse width for multiple voices, just add the voice numbers together. The second parameter, value, is any number in the range 0-4095, and specifies the actual width of the pulse, 2048 being a perfect square wave. In order to work out the precise pulse width, use the formula on page 462 of the PROGRAMMER'S REFERENCE GUIDE.

EXAMPLE: To vary the pulse width for voice 1:

ENTER: 10 VOL 15
 20 WAVE 1,01000000
 30 ENVELOPE 1,5,10,8,1
 40 MUSIC 9,"<SHIFT CLR/HOME>1<F1>C5<F1>D5<F1>E5
 <F1>F5<F1>G5<F1>A5<F1>B5<F1>C6<F1>"
 50 FOR I=0TO4096 STEP 128
 60 PULSE 1,I
 70 PLAY 1
 80 NEXT

TYPE: RUN <RETURN>

RESULT: An ascending scale with varying pulse width is played.

9.3.4 BEEP

FORMAT: BEEP

PURPOSE: To produce a "beep".

The BEEP command has no parameters, and simply produces a "beep" from voice 1.

EXAMPLE: To produce a beep:

TYPE: BEEP <RETURN>

RESULT: A beep is produced from voice 1.

SECTION TEN

EXAMPLE PROGRAMS

10.1 INTRODUCTION

This section contains three programs that demonstrate what may be achieved when using SIMONS' BASIC and SIMONS' BASIC EXTENSION commands. Simply type each program in and RUN it.

10.2 PROGRAM 1 - GRAPH PLOTTER

This program allows you to define formulae, and plot graphs derived from these formulae in high-resolution mode.

```

1000 DIM K(255),EQ$(27)
1010 K$="0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ+-*/<.>"
1020 K$=K$+CHR$(20)+CHR$(13)
1030 FOR I=1 TO LEN(K$)
1040 K(ASC(MID$(K$,I,1)))=1
1050 NEXT
1060 HIRES 15,0
1070 LABEL 10,10,"ENTER EQUATION ",1,1,2,1,10,0,0
1080 LABEL 10,30,"USE VARIABLES 'X' AND 'Y'",1,1,2,1,10,0,0
1090 LABEL 10,50,"Y =",1,1,2,1,10,0,0
1100 SX=50:SY=50:EXEC GET.CHAR
1110 EXEC EVAL.STRING
1120 EQ$=EV$
1130 HIRES 15,0
1140 LABEL 10,10,"ENTER RANGE",1,1,2,1,10,0,0
1150 LABEL 10,30,"MIN X",1,1,2,1,10,0,0
1160 SX=70:SY=30:EXEC GET.CHAR
1170 EXEC EVAL.STRING
1180 LX=VAL(EV$)
1190 LABEL 10,50,"MAX X",1,1,2,1,10,0,0
1200 SX=70:SY=50:EXEC GET.CHAR
1210 EXEC EVAL.STRING
1220 HX=VAL(EV$)
1230 IF HX<=LX THEN 1130
1240 LABEL 10,70,"MIN Y",1,1,2,1,10,0,0
1250 SX=70:SY=70:EXEC GET.CHAR
1260 EXEC EVAL.STRING
1270 LY=VAL(EV$)
1280 LABEL 10,90,"MAX Y",1,1,2,1,10,0,0
1290 SX=70:SY=90:EXEC GET.CHAR
1300 EXEC EVAL.STRING

```

SIMONS' BASIC EXTENSION

```

1310 HY=VAL(EV$)
1320 IF HY<=LY THEN1130
1330 HIRES 15,0
1340 LABEL 10,10,"ENTER STEP VALUE",1,1,2,1,10,0,0
1350 SX=180:SY=10:EXEC GET.CHAR
1360 EXEC EVAL.STRING
1370 SP=VAL(EV$)
1380 IF SP>(HX-LX)/10 THEN1330
1390 MX=(LX<0)*LX:MY=(LY<0)*LY
1400 SCALE MX+HX,MY+HY
1410 HIRES 15,0
1420 HLIN 199-SCY(MY),0,320,1
1430 VLIN SCX(MX),0,200,1
1440 X=LX:A$="Y="+EQ$+":BACK"
1450 EVAL A$
1460 IF Y>HY THEN PLOT 0,0,0:GOTO1490
1470 IF Y<LY THEN PLOT 0,199,0:GOTO1490
1480 PLOT 0,200-SCY(Y+MY),0
1490 FOR X=LX TO HX STEP SP
1500 EVAL A$
1510 IF Y<LY THEN PLOT SCX(X+MX),199,0:GOTO1540
1520 IF Y>HY THEN PLOT SCX(X+MX),0,0:GOTO1540
1530 DRAW TO SCX(X+MX),200-SCY(Y+MY),1
1540 NEXT
1550 LABEL68,192,"PRESS SPACE TO CONTINUE",1,1,1,1,8,0,0
1560 GET E$:IF E$<>" " THEN1560
1570 RUN
1580 PROC GET.CHAR
1590 D=SX
1600 LABEL D,SY,"_",1,1,2,1,10,0,0
1610 LABEL D,SY,"_",0,1,2,1,10,0,0
1620 FORI=1TO10
1630 GETE$:IFE$<>" "THEN1660
1640 NEXT
1650 GOTO1600
1660 E = ASC(E$)
1670 IF K(E) = 0 THEN1600
1680 IF E=20 THEN D=D+10*(D>SX):GOTO1600
1690 IF E=13 AND D>SX THEN1770
1700 IF E=13 THEN1600
1710 LABEL D,SY,"",0,1,2,1,10,0,0
1720 LABEL D,SY,E$,1,1,2,1,10,0,0
1730 EQ$((D-SX)/10)=E$
1740 D=D+10
1750 IF D>310 THEN D=310
1760 GOTO1600
1770 END PROC
1780 PROC EVAL.STRING
1790 EV$=""
1800 FOR I=0TO(320-SX)/10
1810 IF EQ$(I)=" "THEN1840
1820 EV$=EV$+EQ$(I)
1830 EQ$(I)=" "
1840 NEXT
1850 END PROC

```

10.2 PROGRAM 2 - DOGGY

This program draws a famous dog on the multi-colour graphics screen.

```

1000 HIRES0,1
1010 MULTI 5,2,11
1020 COLOUR 6,3
1030 BLOCK 0,100,160,200,1
1040 VLIN 48,160,199,3
1050 VLIN 119,160,199,3
1060 BLOCK 49,161,118,199,2
1070 LINE 135,152,131,160,3
1080 DRAW TO 34,160,3 TO 30,152 TO 135,152
1090 DRAW TO 125,69,3 TO 118,69
1100 HLIN 192,48,119,3
1110 HLIN 176,48,119,3
1120 HLIN 165,48,119,3
1130 HLIN 97,59,127,3
1140 HLIN 97,36,51,3
1150 LINE 48,69,38,69,3
1160 DRAW TO 31,153,3
1170 HLIN 125,34,131,3
1180 PLOT 57,72,3
1190 DRAW TO 58,109,3
1200 ARC 54,110,90,300,9,4,6,3
1210 DRAW TO 51,90,3 TO 54,71
1220 LINE 48,70,55,71,3
1230 DRAW TO 74,67,3 TO 88,69 TO 89,72
1240 DRAW TO 90,73,3
1250 LINE 91,68,91,73,3
1260 DRAW TO 95,73,3
1270 LINE 79,60,93,60,3
1280 DRAW TO 98,66,3 TO 98,70
1290 LINE 99,68,116,71,3
1300 DRAW TO 119,68,3 TO 118,64 TO 113,35
1310 DRAW TO 107,36,3 TO 107,48 TO 110,57
1320 DRAW TO 100,55,3
1330 LINE 109,56,105,52,3
1340 DRAW TO 105,35,3 TO 107,35
1350 HLIN 39,110,114,3
1360 HLIN 43,111,114,3
1370 HLIN 69,98,107,3
1380 HLIN 69,82,88,3
1390 HLIN 69,68,79,3
1400 VLIN 70,47,67,3
1410 VLIN 73,47,67,3
1420 LINE 103,54,102,54,3
1430 DRAW TO 101,49,3 TO 100,42 TO 98,38
1440 DRAW TO 91,30,3 TO 85,29 TO 82,32
1450 DRAW TO 79,36,3 TO 78,39 TO 77,43
1460 DRAW TO 76,45,3 TO 70,47 TO 69,24
1470 DRAW TO 65,19,3 TO 59,19 TO 56,20
1480 DRAW TO 52,23,3 TO 50,27 TO 50,41
1490 DRAW TO 52,46,3 TO 52,49 TO 46,55
1500 DRAW TO 44,57,3 TO 44,64 TO 47,70

```

SIMONS' BASIC EXTENSION

```
1510 LINE 52,50,54,54,3
1520 LINE 50,60,40,53,3
1530 CIRCLE 61,16,4,3,3
1540 LINE 96,72,98,69,3
1550 PAINT 50,156,2
1560 PAINT 50,146,2
1570 PAINT 50,120,2
1580 PAINT 50,98,2:PAINT 100,98,2
1590 PAINT 50,90,2:PAINT 100,90,2
1600 PAINT 55,80,3:PAINT 55,110,3
1610 PAINT 60,16,3
1620 PAINT 71,60,2
1630 LOW COL 1,2,11
1640 PAINT 60,50,1:PAINT 80,40,1
1650 PAINT 106,50,1
1660 LOW COL 7,2,11
1670 ARC 0,0,90,180,10,20,20,1
1680 PAINT 0,0,1
1690 FOR T=90 TO 180 STEP 10
1700 ANGL 0,0,T,40,40,1
1710 NEXT
1720 DESIGN 0,$CC00
1730 @.....
1740 @.....BBB.....
1750 @.....BBBBBB.....
1760 @.....BBBBBBB..BB.....
1770 @...BBBBBBBBBBBBBBBB.....
1780 @..BBBBBBBBBBBBBBBBBBB...
1790 @.BBBBBBBBBBBBBBBBBBBBBB..
1800 @BBBBBBBBBBBBBBBBBBBBBBBBB.
1810 @BBBBBBBBBBBBBBBBBBBBBBBBBB.
1820 @BBBBBBBBBBBBBBBBBBBBBBBBBB.
1830 @BBBBBBBBBBBBBBBBBBBBBBBBBB.
1840 @.BBBBBBBBBBBBBBBBBBBBBBBBB.
1850 @...BBBBBBBBBBBBBBBBBBBBBB...
1860 @.....
1870 @.....
1880 @.....
1890 @.....
1900 @.....
1910 @.....
1920 @.....
1930 @.....
1940 MOB SET 1,48,1,1,0
1950 MOB SET 2,48,1,1,0
1960 MOB SET 3,48,1,1,0
1970 MOB SET 4,48,1,1,0
1980 MOB SET 5,48,1,1,0
1990 MOB SET 6,48,1,1,0
2000 MMOB 1,100,50,100,50,3,0
2010 MMOB 2,120,110,120,110,1,0
2020 MMOB 3,300,80,300,88,1,0
2030 MMOB 4,70,80,70,80,1,0
2040 MMOB 5,200,50,200,47,1,0
2050 MMOB 6,150,80,150,80,3,0
2060 INIT
2070 FORT=1T06:READ X%
2080 SPRITE T,X%,0,2,0,0,0,0,0,0,0
2090 NEXT
2100 START
2110 GOTO2110
2120 DATA 1,-1,2,-2,1,-1
```

10.3 PROGRAM 3 - ROAD RACER

A winding road, and a fast car. How long can you last?

```

1000 DIM K(32),J(8),PX(7),PY(7)
1010 K(10)=-1:K(18)=1:KY=197
1020 READ ARR J
1030 DATA 0,0,1,1,1,0,-1,-1,-1
1040 READ ARR PX
1050 READ ARR PY
1060 DATA 0,25,45,35,230,245,260,250
1070 DATA 0,80,160,240,25,85,145,205
1080 DEF FND(I)=(RND(1)*8)-4
1090 DEF FNE(I)=INT(RND(1)*I)+1
1100 FOR I=0 TO RND(TI)*10
1110 Q=RND(TI) : NEXT
1120 DESIGN 1,240*64
1130 @.....
1140 @.....
1150 @.....
1160 @...DDDDD...
1170 @...DCDCD...
1180 @.....C.....
1190 @.BB.CCC.BB..
1200 @.BBDCCCDBB..
1210 @.BB.CCC.BB..
1220 @....CDC.....
1230 @....CDC.....
1240 @....CDC.....
1250 @....CDC.....
1260 @...CCCC....
1270 @BB.CCCCC.BB.
1280 @BB.CCCCC.BB.
1290 @BBDCCCCCDBB.
1300 @BB.CCDDC.BB.
1310 @BB...D...BB.
1320 @...DDDDD...
1330 @...DCDCD...
1340 DESIGN 1,241*64
1350 @...CCCC....
1360 @..CCCCCCC...
1370 @.CCCCCCCCC..
1380 @..CCCCCCC...
1390 @...CCCCCCCC.
1400 @.CCBCCCCCCCC

```

SIMONS' BASIC EXTENSION

```

1410 @CCCCBCCCBCC
1420 @.CCCCBBBCCC.
1430 @.CBCCBCCCBC.
1440 @C.CBBBCBBCC.
1450 @.CCCCBBCC...
1460 @CCCCCBCC.C..
1470 @..BCCBCCCBC.
1480 @.CCBCBCBBCCC
1490 @CCCCBBBCCCC.
1500 @.CCCBBCCCC..
1510 @...BCBBBCCCC
1520 @....BBCCBB..
1530 @.....BBBCC..
1540 @.....B.....
1550 @.....B.....
1560 IFPEEK(16000)=128 THEN1590
1570 POKE16000,128
1580 EXEC DEF.EXPL
1590 HS$="000000"
1600 L(0)=101:L(1)=101:L(2)=84:L(3)=71
1610 L(4)=66:L(5)=72:L(6)=89:L(7)=103
1620 SL=15*8:D=07*8:SR=SL+D
1630 S=54272:SD=500
1640 SX=160:SY=210
1650 MN=46:MX=200
1660 SC=SCREEN
1670 VOL 0
1680 EXEC MOBS.OFF
1690 PRINT CHR$(147) :COLOUR 2,12
1700 PRINT AT(12,10) "■KEYBOARD [F1]"
1710 PRINT AT(12,12) "   OR"
1720 PRINT AT(12,14) "JOYSTICK [F7]"
1730 A = INKEY
1740 IF A<>1 AND A<>7 THEN1730
1750 IF A=7 THEN1820
1760 IP=1
1770 PRINT AT(11,16) "USE KEYS:-"
1780 PRINT AT(13,17) "■A■ TO MOVE LEFT"
1790 PRINT AT(13,18) "■D■ TO MOVE RIGHT"
1800 PAUSE 3
1810 GOTO1850
1820 IP=2
1830 PRINT AT(8,16) "PLUG JOYSTICK IN PORT 2"
1840 PAUSE 3
1850 COLOUR 5,13
1860 PRINT CHR$(147)
1870 FORI=24 TO 0 STEP-1
1880 PRINT AT(14,I)" |";SPC(6)" |"
1890 PRINT AT(14,I)" |";SPC(6)" |"
1900 NEXT

```

```

1910 PRINT AT(34,1)"SCORE"
1920 PRINT AT(34,5)"HIGH"
1930 PRINT AT(34,6)"SCORE"
1940 PRINT AT(34,7) RIGHT$(HS$,4)
1950 POKE53265,16
1960 RLOCMOB 0,SX,SY,0,1
1970 INIT : ON DETECT 2,2350:DETECT1
1980 FORI=1 TO 7
1990 RLOCMOB I,PX(I),PY(I),0,1
2000 SPRITE I,0,2,3,0,0,0,0,0,0
2010 MOB SET I,241,5,0,1
2020 NEXT
2030 MOB SET 0,240,1,0,1
2040 CMOB 0,2
2050 START 1,2,3,4,5,6,7
2060 VOL15
2070 WAVE 1,00001000:WAVE 1,01000001
2080 ENVELOPE 1,3,8,8,8
2090 PULSE 1,2048
2100 DOKE S,500
2110 TI$="000000"
2120 X=FND(0)
2130 FORI=1 TO FNE(5)+2
2140 SL=SL+X:SR=SL+D
2150 IF SL<MN THENSL=MN:SR=SL+D
2160 IF SR>MX THENSR=MX:SL=SR-D
2170 L=INT(SL/8):R=INT(SR/8):R1=40-R-2
2180 DOWNB 0,0,20,25:DOWNB 0,20,10,25
2190 POKE SC+L,L(SL AND7)
2200 POKE SC+R,L(SR AND7)
2210 T=T+1
2220 IF T/50=INT(T/50) THEN EXEC INC.LEVEL
2230 ON IP GOTO2240,2270
2240 K=PEEK(KY)AND31
2250 DX=K(K)
2260 GOTO2290
2270 J=JOY AND15
2280 DX=J(J)
2290 XX=SPRX(0)
2300 SX=SX+DX
2310 SPRITE 0,DX,0,2,0,0,0,0,0,1
2320 PRINT AT(34,2) RIGHT$(TI$,4)
2330 NEXT
2340 GOTO2120
2350 CLEAR
2360 IF TI$>HS$ THEN HS$=TI$
2370 FOR I=1 TO 60
2380 WAVE 1,10000001
2390 DOKE S,4000
2400 MOB SET 0,241+IAND3,1,0,1

```


SIMONS' BASIC EXTENSION

```
2410 VOL 15-(I/4)
2420 NEXT
2430 EXEC MOBS.OFF
2440 GOTO1600
2450 PROC DEF.EXPLN
2460 FORI=242*64 TO 244*64
2470 POKE I,2↑INT(RND(1)*8)
2480 NEXT
2490 END PROC
2500 PROC INC.LEVEL
2510 D=D+(D>30)
2520 SY=SY+(SY>120)*3
2530 RLOCMOB 0,SPRX(0),SY,0,1
2540 SD=SD-(SD<1400)*40
2550 DOKE S,SD
2560 END PROC
2570 PROC MOBS.OFF
2580 FORI = 0 TO 7
2590 MOB OFF I
2600 NEXT I
2610 END PROC
```

APPENDIX A

ERROR MESSAGES

In the course of using SIMONS' BASIC with SIMONS' BASIC EXTENSION, an error message may appear. These messages are unique to SIMONS' BASIC and SIMONS' BASIC EXTENSION, and along with the HELP command (see section 2.5), make debugging much easier, and faster.

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 MOB IN LINE n

A parameter within the MOB design stage is outside the range defined. The line number of the error (n), 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.

? NULL STRING ERROR

An attempt has been made to pass an empty string to another command.

GLOSSARY

A list of terms used in this manual.

ADDRESS

Each memory location has its own identification number. This number is called the address.

BASIC INTERPRETER

Since binary is all that the processor really understands, a BASIC INTERPRETER (i.e. SIMONS' BASIC) is a program that translates the commands that you type, into strings of binary numbers that the computer will be able to understand, and act on.

BINARY

Number system to the base 2, the only number system that the processor really understands. Applications of binary in computers result from simple two-state devices, i.e. on, or off.

BIT

A single binary number, either a 0, or a 1. The word bit comes from combining Binary with digIT.

BUFFER

A temporary storage area in the computer's memory.

BUG

A mistake in the program that causes it to "crash out".

BYTE

An eight-bit binary number. The largest decimal number that a byte can represent is 255.

DEBUGGING

Going through the program, and taking out all the bugs.

DEVICE NUMBER

Because data can be sent to a number of devices external to the computer (e.g. screen, printer, disk drive), each device has its own identification code, so that the computer can distinguish between them.

DIRECT MODE

In DIRECT MODE you may type a command (without a line number), and that command is executed as soon as you press <RETURN>. Whereas in PROGRAM MODE, none of the commands are executed until the program is RUN.

FLAG

A flag is a pointer in memory, that helps the computer keep track of what's going on. If a condition occurs, then the flag is hoisted up the flag-pole, and the computer can act accordingly.

FLOATING-POINT LIMITS

Floating-point constants allow you to work with figures of up to nine digits, representing values between -999999999 to +999999999. Numbers smaller than 0.01, or larger than 999999999 are printed in scientific notation.

HEXADECIMAL

Another major number system (the other is binary) used by computers. Hexadecimal is to the base 16, and is closely related to binary.

INTEGER LIMITS

Integer values must be within the range -32768 to +32767.

KERNAL

The operating system of the COMMODORE 64. All input, output, and memory management is controlled by the KERNAL.

NYBBLE

Half a byte, or a four-bit binary number.

PIXEL

The smallest addressable location on the screen. Everything that your computer displays, from text to high-resolution graphics, is made up of pixels.

PROGRAM CRASH

An unwanted halt in program execution, usually resultant in an error message.

RAM

Random Access Memory; this is free memory available to the user. The contents of RAM are not retained when the computer is switched off.

ROM

Read Only Memory; ROM is not available to the user, it contains the computer's operating system, and is retained when the computer is switched off.

SECONDARY ADDRESS

The secondary address is a supplementary command used when transferring data between the computer, and one of its peripheral devices.

SPRITE UPDATE

Calculations relating to the current position of the sprite, and its new position. The frequency with which these calculations, and actual sprite movements take place are determined by the parameter sp, in the SPRITE command.

INDEX

	Page
ADD ALL	3-9
ADD ARR	3-5
ALTER	2-3
BACK	4-6
BARRIER	8-6
BCKFLASH	7-11
BEEP	9-4
BIN\$	4-2
CALCX	4-3
CALCY	4-4
CCOL	7-7
CHAIN	2-5
CHANGE	8-7
CLEAR	8-8
CONTINUE	8-18
Converting from decimal/binary to hexadecimal	4-2
Converting from decimal/hexadecimal to binary	4-2
COPY ARR	3-11
CREATE	8-20
DEEK	5-2
DEG	4-1
DELETE	2-2
DESIGN	7-9
DIV ALL	3-10
DIV ARR	3-8
Doggy program	10-3
DOKE	5-1
DRAW TO	6-6
DS\$	2-8
ECOL	7-6
ELEMENTS	3-14
EVAL	4-5
FILTER	9-2
FONT	7-4
FORCE	2-7
GRAD	4-1
Graph plotter program	10-1
GRID	6-1

SIMONS' BASIC EXTENSION

	Page
HELP	2-4
HEX\$	4-2
High-resolution graphics	6-1
HIMEM	5-2
HLIN	6-7
HLOAD	6-4
HSAVE	6-3
INFO	8-10
INIT	8-9
INPUT ARR	3-11
INVERT	8-13
LABEL	6-5
Loading SIMONS' BASIC EXTENSION	1-3
LOMEM	5-3
LOW RES	7-1
MAX	3-15
MCOL	7-8
MIN	3-14
MODE	9-2
MUL ALL	3-10
MUL ARR	3-7
Multi-colour graphics	6-1
Music commands	9-1
NORMAL	8-9
ON DETECT	8-17
PREPARE	7-3
PRINT ARR	3-4
PROTECT	2-6
PULL	7-4
PULSE	9-3
PUSH	7-3
READ ARR	3-12
RENUMBER	2-1
REVERSE	8-15
Road racer program	10-5
ROTATE	7-10

	Page
SCALE	6-8
SCHR	7-8
SCOL	7-7
SCRATCH	3-12
SCREEN	5-3
SCX	6-9
SCY	6-10
SET ARR	3-3
SHOW	8-15
SORT	3-16
SPRITE	8-2
SPR LOC	8-12
SPRX	8-11
SPRY	8-12
START	8-7
SUB ALL	3-9
SUB ARR	3-6
SUM	3-13
TICK	6-2
TRANSFER	8-19
UPPER	7-5
VLIN	6-7
XVEC	8-5
YVEC	8-6
ZER ARR	3-5
★	7-12



**© COMMODORE BUSINESS MACHINES
(UK) LIMITED**

All rights reserved. No part of this program or accompanying literature may be duplicated, copied, transmitted or otherwise reproduced without the express written consent of the Publishers.

**COMMODORE BUSINESS MACHINES
(UK) LIMITED,**

675 Ajax Avenue, Slough Trading Estate, Slough,
Berks. SL1 4BG, England.

MADE IN ENGLAND

The logo features a stylized 'C' symbol on the left, composed of a thick blue arc and a thinner blue arc. To the right of the symbol, the word 'commodore' is written in a bold, lowercase, sans-serif font. Below 'commodore', the word 'COMPUTER' is written in a larger, all-caps, sans-serif font.