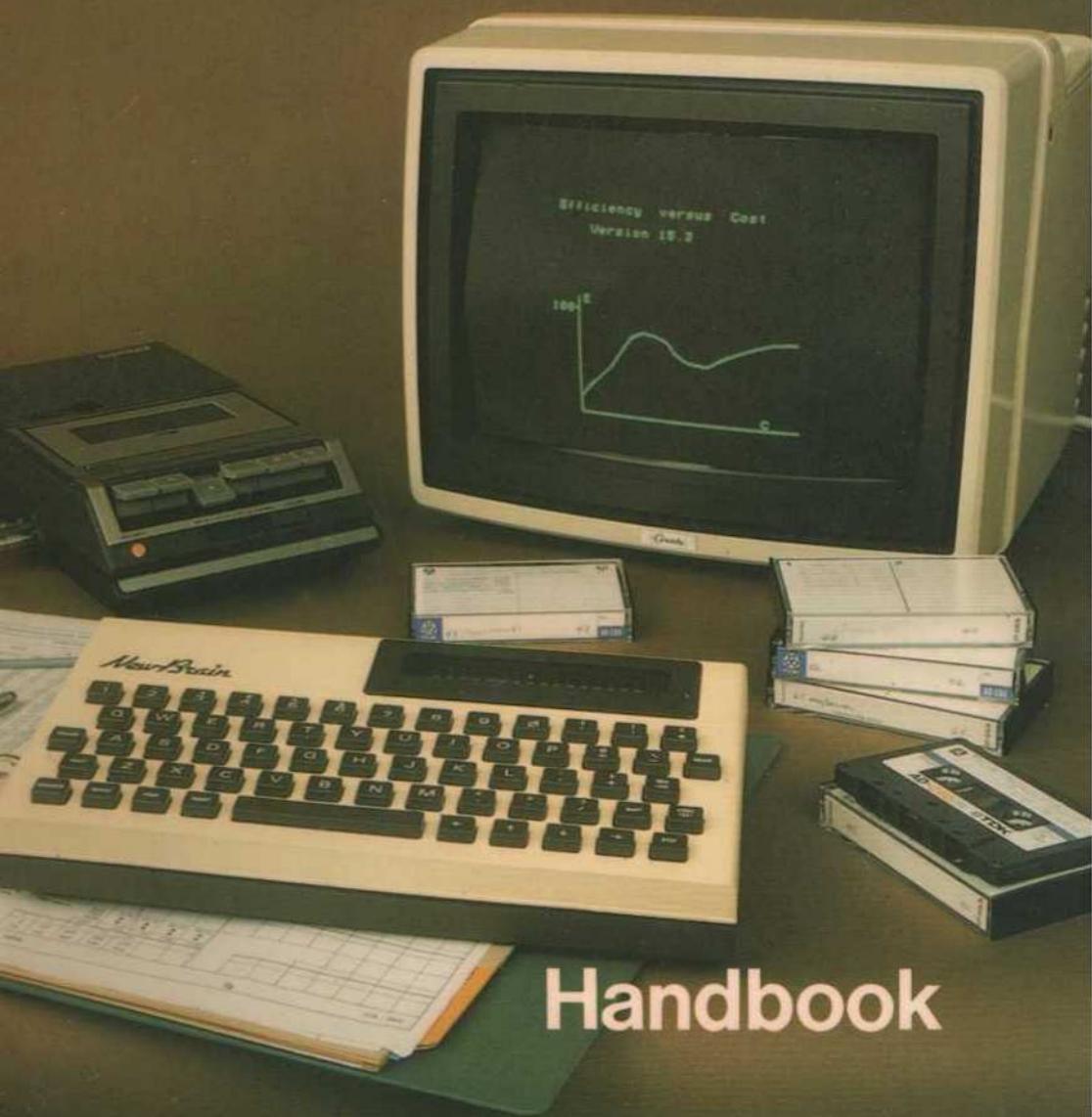


# *NewBrain*



**Handbook**



## **PREFACE**

This handbook is a guide to the use of the NewBrain. It contains a full description of the essential features of the NewBrain and NewBrain BASIC.

Other documents for the NewBrain include the NewBrain Beginners Guide and the NewBrain Software Technical Manual.

**Grundy Business Systems  
Science Park  
Cambridge  
U.K.**

## Contents, page (i), Chapter 2, Section 2:

DELETE "CONDITIONS/"

## Chapter 4, Section 5:

for "ERROR . . . 52"  
read "ERROR AND REPORT . . . 51"

Page 6, last line:

for "blanks"  
read 'flashes through the same message and  
shows a flashing underline as its cursor.  
The user may now choose to select an eighty  
column display by typing

OPEN#0, 4, "L"  
followed by NEWLINE (see pp 169 . . . 171).'

Page 10, line 8:

for "page 40"  
read "page 24"

second column, line 14:

for "edigint"  
read "editing"

Page 11, second column, line 16:

for "NOW IS THE TIME TO COME"  
read "NOW IS NOT THE TIME TO COME"

Page 19, line 2:

after "6.3 Save"  
insert "Before recording ensure the cassette is  
wound past the end of the leader tape."

Page 20, line 3:

for "OPEN#12, 2"  
read "OPEN OUT#12, 2"

line 30:

for "pass"  
read "press"

Page 23, line 6:

for "NUMERIC CONDITIONS"  
read "NUMERIC CONSTANTS"

Page 25, second column, last line:

for "0.000099"  
read "0.000099"

Page 26, second column, last line:

for "(see 66)"  
read "(see page 66)"

Page 31, line 19:

for "'a'<"an"<"and<and"'  
read "'a'<"an"<"and<'ant'"

second column, lines 5-6:

for "order preceding + or - leave or  
change sign"  
read "order.  
preceding + or -      leave or change sign"

second column, line 16:

for "4 \* 6 + 6 - 2 = 24"  
read "4 \* 5 + 6 - 2 = 24"

Page 36, line 11:

for "Z22  
read "Z2                          Z23"  
                                        Z3"

Page 45, line 22:

for '20 INPUT ("NUMBER")N'  
read '20 INPUT ("GIVE ME A NUMBER")N'

Page 50, second column, line 10:

for "lines 10 and 999"  
read "lines 10, 20 and 999"

Page 51, second column, line 17:

for  
read "120 GOTO 20"  
"120 RESUME 20"

second column, line 26:

for  
read "120 END"  
"120 REPORT"

Page 52, line 2:

DELETE "REPORT"

Page 58, second column, line 8:

for  
read "400 PRINT A,B,C,D,E,F,G,H,J"  
"400 PRINT A,B,C,D,E,F,G,H,I,J"

second column, line 22:

for  
read "40 RESTORE: NEXT"  
"40 RESTORE: NEXT I"

Page 61, line 11:

for  
read "system"  
"stream"

Page 65, line 5:

for  
read "Chapter 7"  
"Chapter 8"

Page 68, second column, line 4:

DELETE the 3 lines commencing  
"1 of 10 under 9....."

second column, line 23:

for  
read "2 for cube root"  
"3 for cube root"

Page 86:

INSERT new line 30

"TURNBY (x) gives unpredictable results  
when plotting in radians instead of degrees."

Page 87, second column, line 19:

DELETE "(1,0.5)"

Page 89, line 10:

for  
read "sub-heading is too....."  
"sub-heading is too long....."

Page 92, line 10:	<u>for</u> <u>read</u>	"READ or VERIFY" "read (e.g. LOAD, INPUT, VERIFY)"
Page 94, second column, line 25:		
	<u>for</u> <u>read</u>	"interrupted" "intercepted"
Page 105, line 2:	<u>after</u> <u>INSERT</u>	"109 System error." "e.g. Attempt to input from a printer."
Page 110, line 15:	<u>for</u> <u>read</u>	"5575" "5374"
Page 125, line 12:	<u>for</u> <u>read</u>	"24" "25"
Page 132, line 9: line 10:	<u>for</u> <u>for</u>	"black" <u>read</u> "white" "white" <u>read</u> "black"
Page 142, line 2: line 23: line 24:	<u>for</u> <u>read</u> <u>for</u> <u>read</u>	"district" "distinct" "225 - 250" "225 - 254" "192 - 223" "193 - 222"
Page 153, line 16:	<u>for</u> <u>read</u>	"5461" "5374"
Page 161, line 26:	<u>for</u> <u>read</u>	"compromise" "comprise"
Page 170, line 16:	<u>DELETE</u>	"records....."
Page 175, last line:	<u>for</u> <u>read</u>	' OPEN#4, 0, "L254" : OPEN#1,11, "#4w229" ' ' OPEN#4, 0, "L150" : OPEN#1,11, "#4w220" '
Page 176, first line:	<u>for</u> <u>read</u>	"254" "150"

## **HOW TO USE THIS MANUAL**

The experienced computer user will find that the Appendices, containing the comprehensive information on the NewBrain, and Chapter 1, Introduction, will serve the majority of his needs. The other chapters provide illustrations of information given concisely in these Appendices. Those unfamiliar with computers or with BASIC should read the handbook through and refer to both the Chapters and the Appendix for reference. As the user gains familiarity with the NewBrain Computer, he will find that the Appendices provide sufficient reference material for normal needs.

## CONTENTS

	page
<b>CHAPTER 1 — INTRODUCTION</b>	
1 CONNECTING UP .....	2
2 SWITCHING ON .....	6
3 THE KEYBOARD .....	7
4 THE DISPLAY .....	9
5 OPERATING SYSTEM .....	16
6 USING CASSETTE RECORDER .....	18
7 USING PRINTER .....	21
 <b>CHAPTER 2 — BASIC DEFINITIONS</b>	
1 INTRODUCTION .....	24
2 NUMERIC CONDITIONS/CONSTANTS .....	25
3 VARIABLES .....	27
4 ARRAY VARIABLES .....	28
5 EXPRESSIONS .....	28
6 ERROR MESSAGES .....	32
 <b>CHAPTER 3 — SIMPLE BASIC</b>	
1 ASSIGNMENT — LET .....	34
2 PRINT .....	35
3 TAB .....	36
4 INPUT .....	37
5 LIST .....	38
6 RUN, END AND GOTO .....	39
7 STOP AND CONTINUE .....	40
8 REM .....	41
 <b>CHAPTER 4 — CONTROL</b>	
1 FOR — NEXT .....	44
2 IF — THEN .....	46
3 GOSUB .....	48
4 ON — GOTO AND ON — GOSUB .....	50
5 ON ERROR .....	52
6 ON BREAK .....	52

## CONTENTS (continued)

	page
<b>CHAPTER 5 – DATA STRUCTURES</b>	
1 ARRAYS .....	54
2 DIM AND CLEAR .....	56
3 OPTION BASE .....	57
4 DATA, READ AND RESTORE .....	57
 <b>CHAPTER 6 – FURTHER INPUT AND OUTPUT</b>	
1 OPEN AND CLOSE .....	60
2 STREAM NUMBERS .....	61
3 LINPUT .....	61
4 PUT AND GET .....	62
5 SAVE, VERIFY, LOAD AND LIST .....	63
 <b>CHAPTER 7 – INTRINSIC FUNCTIONS</b>	
1 PI .....	66
2 TRIGONOMETRIC FUNCTIONS .....	66
3 LOGARITHMS .....	67
4 POWERS .....	68
5 ARITHMETIC .....	69
6 RANDOM NUMBERS .....	71
7 USER DEFINE FUNCTIONS .....	73
 <b>CHAPTER 8 – STRING HANDLING</b>	
1 CONCATENATION .....	76
2 LEN .....	76
3 LEFT\$ .....	77
4 MID\$ .....	77
5 RIGHT\$ .....	78
6 INSTR .....	78

## **CONTENTS (continued)**

	page
<b>CHAPTER 9 — CONVERSION</b>	
1	CHARACTER/ASCII .....
2	STRING/NUMERIC .....
3	TEST STRING FOR NUMBER .....
	<b>80</b>
	<b>82</b>
	<b>82</b>
<b>CHAPTER 10 — GRAPHICS</b>	
1	SPECIAL SCREEN CHARACTERS .....
2	HIGH-RESOLUTION DISPLAY .....
3	THE GRAPHICS "PEN" .....
4	THE PLOT COMMANDS .....
5	THE PEN FUNCTION .....
6	GRAPHICS STREAM DEFAULTS .....
	<b>84</b>
	<b>85</b>
	<b>85</b>
	<b>86</b>
	<b>89</b>
	<b>90</b>
<b>CHAPTER 11 — HELP IN AN EMERGENCY</b>	
1	INTERRUPTION — STOP .....
2	CHANGING THE DISPLAY .....
3	FREEING THE KEYBOARD .....
4	RELEASING MEMORY .....
5	WHEN TO SWITCH OFF .....
	<b>92</b>
	<b>93</b>
	<b>93</b>
	<b>94</b>
	<b>94</b>
<b>APPENDICES</b>	
1	ERROR NUMBERS .....
2	BASIC TECHNICAL SPECIFICATION .....
3	SCREEN EDITOR TECHNICAL SPECIFICATION .....
4	BASIC RESERVED WORDS .....
5	LINE AND SCREEN DISPLAY CHARACTER SETS .....
6	BASIC STATEMENT KEYWORDS .....
7	DEVICE DRIVER SUMMARY .....
8	CALL STATEMENT AND O/S ROUTINES .....
9	HARDWARE SPECIFICATION .....
	<b>97</b>
	<b>107</b>
	<b>123</b>
	<b>133</b>
	<b>139</b>
	<b>149</b>
	<b>167</b>
	<b>177</b>
	<b>187</b>
<b>INDEX</b>	<b>191</b>



## CHAPTER 1

### INTRODUCTION

The first chapter of the NewBrain handbook explains briefly how to connect up the computer, and then describes how the computer is used from the keyboard and display (the console). The chapter continues with a description of the commands used to edit the displays, and concludes with a brief guide to the operating system, cassette recorders and printers.

New concepts are introduced by CAPITAL LETTERS and explained in the relevant section. Any visible display from the NewBrain is shown in GREEN and text typed by the user is shown in BROWN.

1. CONNECTING UP
  - 1.1 Power Supply
  - 1.2 Television or monitor
  - 1.3 Cassette recorder(s)
2. SWITCHING ON
3. THE KEYBOARD
4. THE DISPLAY
  - 4.1 Screen Display
  - 4.2 Editing the Screen Display
  - 4.3 Cursor Control Commands
  - 4.4 Screen Editing
  - 4.5 Line Display (Model AD Only)
5. THE OPERATING SYSTEM
  - 5.1 Overview
  - 5.2 The Input-Output System
  - 5.3 Extension
6. USING CASSETTE RECORDERS
  - 6.1 Connection
  - 6.2 Load
  - 6.3 Save
  - 6.4 Verify
  - 6.5 The Tape 2 Socket
7. USING A PRINTER

# CHAPTER 1 – INTRODUCTION

## 1 CONNECTING UP

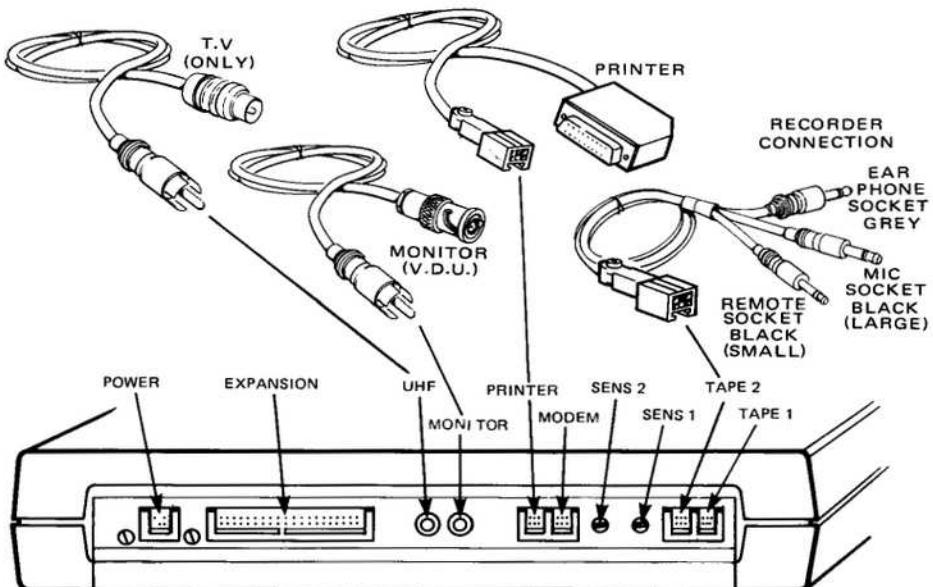
The NewBrain computer is complete with

- a power supply
- a cassette connecting lead
- a T.V. connecting lead
- handbook

Optional extra connecting leads include

- monitor lead
- printer lead
- second cassette lead

### TV & MONITOR CONNECTIONS



## CHAPTER 1 – INTRODUCTION

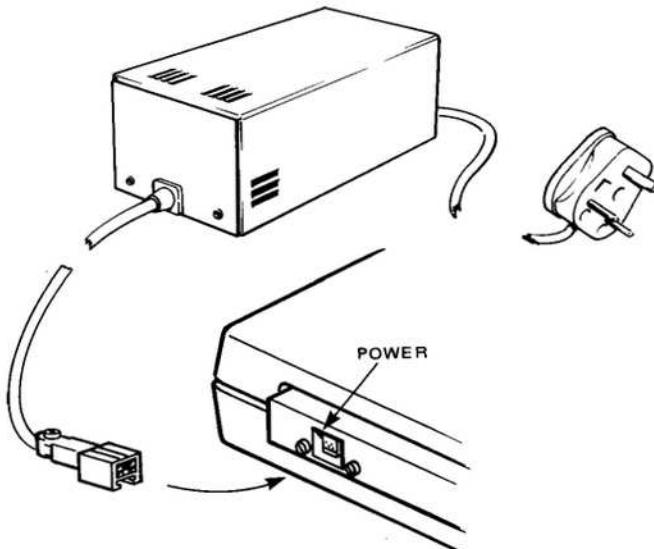
### 1 CONNECTING UP

The printer, cassette, power and modem connections are best made by first engaging the lugs on the cable connector with the flange on the machine connector as illustrated.

#### 1.1 Power Supply

The mains lead from the power supply is fitted with a mains plug to connect to the domestic electricity supply. The plug is fitted with a 3 amp fuse. The low-voltage supply from the power supply unit is connected to the NewBrain by means of a moulded socket into the plug marked POWER in the rear of the computer.

The embossed legend TOP on the connector must be uppermost with the computer in its normal position. The power socket is designed to prevent incorrect insertion into the power plug, or insertion into the wrong plug, by small polarising inserts.

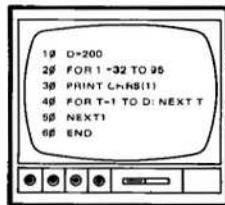


## CHAPTER 1 – INTRODUCTION

### 1 CONNECTING UP

#### 1.2 Television or Monitor

MODEL A NewBrain may be connected either to a U.H.F. television, or to a video monitor. The connection is made from the video output socket to a monitor, or the U.H.F. output socket to a television, using the miniature screened cable provided. The T.V. must be tuned to channel 36. Correct tuning is indicated by absence of background noise on the picture. The contrast and brightness controls should be set low, and then advanced, until a suitable picture is displayed. The best picture is usually obtained with low rather than high settings of the contrast control. The same guidelines apply to monitor adjustment. If desired, both a monitor and a television may be connected, although the picture quality may be slightly degraded.



MODEL AD has in addition a self contained display. It may therefore be operated as a stand alone console, or may be connected to a video monitor, or to a television. If the self-contained display only is to be used it must be activated in the manner described on Page 14.



## **CHAPTER 1 – INTRODUCTION**

### **1 CONNECTING UP**

#### **1.3 Cassette Recorder(s)**

One or two cassette recorders may be connected to a NewBrain for program and data storage. If a cassette recorder is being purchased specifically for use with the computer then it should have a remote microphone socket, which will enable the recorder to be stopped and started automatically by the computer. A cassette recorder with a counter is beneficial as this permits efficient indexing and retrieval of stored programs. Cheap, low-quality recorders are to be avoided, since the demands of digital recording are more severe than those of audio recording. Similarly, good quality cassettes should be used, of not more than 30 minutes per side, to ensure adequate tape thickness. C12 cassettes (6 minutes per side) are available from many computer shops, and permit all but the longest programs to be recorded on one side of the cassette.

The cassette lead from the computer is terminated by three miniature jack plugs. These should be connected as follows:

- |               |                                   |
|---------------|-----------------------------------|
| BLACK (large) | — MIC or MICROPHONE SOCKET        |
| GREY          | — EAR, EARPHONE or MONITOR SOCKET |
| BLACK         | — REMOTE SOCKET                   |

The moulded connector should be inserted into either TAPE 1 or TAPE 2 sockets on the rear of the NewBrain with the moulded legend "TOP" uppermost. The operation of the cassette recorder is described in 6.1 to 6.5.

The volume control should be set at maximum and any tone control set in the flat or inoperative state. Only when all else fails should the output level from the NewBrain be adjusted by the sensitivity control (SENS 1 and SENS 2). Remember to keep the tape head clean.

#### **PRINTER**

A printer with a serial interface may be connected. Both connecting cables and suitable printers are available from Grundy Business Systems Ltd. Other printers may of course be used, but a different cable may be needed and switch settings within the printer may need to be changed.

## **CHAPTER 1 – INTRODUCTION**

### **2 SWITCHING ON**

NewBrain computers will automatically go through an initialisation routine, checking all the memory, when powered up. The routine lasts for about 10 seconds after which the computer is ready to start running programs.

MODEL A may briefly display a checker board pattern on the screen, followed by a pause of 10 seconds and then the message

NEWBRAIN BASIC  
READY  
■

where ■ is the flashing cursor.

MODEL AD in addition displays apparently random characters on its self-contained line display for the full period of the initialisation, after which the line display blanks.

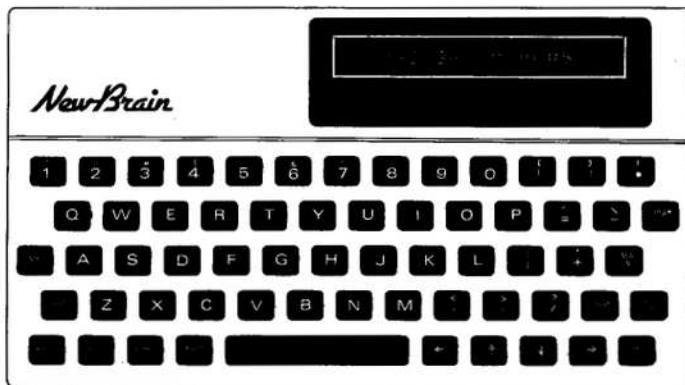
## CHAPTER 1 – INTRODUCTION

### 3 THE KEYBOARD

The NewBrain keyboard is laid out in the same pattern as a typewriter keyboard, with some additional keys whose functions are explained in this section. See figure below. Initially the keyboard will produce lower case letters, and the lower of the characters on key tops embossed with two legends.

Example:

PRESS A	=	a
PRESS B	=	b
PRESS C	=	c
PRESS 1	=	1
PRESS ;	=	;



## CHAPTER 1 – INTRODUCTION

### 3 THE KEYBOARD

The SHIFT Key behaves as a typewriter shift, producing UPPER CASE letters, and the upper of the characters on the keytops. NewBrain BASIC will accept letters input from the keyboard in either UPPER or lower case. In this handbook, when a SHIFTed character is to be typed in, it will be indicated as SH/a. Thus, for example:

PRESS SH/A	=	A
PRESS SH/B	=	B
PRESS SH/C	=	C
PRESS SH/1	=	!
PRESS SH/;	=	:

The alphabetic characters (a-z) may be shifted to upper case (A-Z) by typing CONTROL/1 (henceforth CTRL/1), giving a function similar to the SHIFT-LOCK on a typewriter. All non-alphabetic characters will remain unshifted, thus 1 will still appear as 1 unless SH/1 is pressed.

PRESS CTRL/1 (no visual display)

PRESS A	=	A
PRESS B	=	B
PRESS C	=	C
PRESS 1	=	1
PRESS ;	=	;

CTRL/Ø cancels the effect of CTRL/1, i.e. removes the shift-lock.

Note that pressing SH/† causes an effect (ATTRIBUTE ON— see page 129) used in advanced input/output. In the context of simple input/output SH/† causes the display and any input from the user to become unintelligible to the computer. To recover from this condition press SH/ESCAPE.

## CHAPTER 1 – INTRODUCTION

### 4 THE DISPLAY

As noted in Section 1.2 a variety of means of display may be used by NewBrain computers model A and AD. Model A uses either a monitor connected to the video output, or a conventional television connected to the U.H.F. output. To avoid ambiguity, video or U.H.F. displays will be referred to as SCREEN displays. Model AD outputs data either to the SCREEN or to a 16 character fluorescent display, integral with the console, or to both simultaneously. The 16 character display will be referred to as a LINE display. This section describes how the screen and line displays are controlled and explains how the contents of the displays may be changed or EDITED by means of the additional keys on the NewBrain keyboard.

#### 4.1 Screen Display

The screen display initially consists of 24 lines, each forty characters long. The position on the screen of the next character to be displayed is shown by a flashing CURSOR.

The cursor is initially a flashing block (■) indicating that unless the next character is a CONTROL CHARACTER any text on the line that the cursor is presently on will be cleared, and the line can be overprinted.

CONTROL CHARACTERS are non-printing features of the display. Some of the characters used to edit a display, discussed in this section, are control characters. A full list of control characters

### 4 THE DISPLAY

is in appendix 3. If a non-control character, for example an ALPHANUMERIC (A-Z, 0-9) is pressed, the character chosen will be displayed, and the cursor will be advanced one step to the right, and adopt its normal form, a flashing underbar (—). Only one character will appear even if pressure on the key is maintained. To obtain multiple entries of a character the REPEAT key is used along with the chosen character (RPT/A). If you repeat a character often enough, it will print to the end of the first line, and continue on subsequent lines.

Each subsequent line is a CONTINUATION of the first, and is identified by a non-flashing block in the leftmost position.

READY

A A A A — — — A A A A  
■ A A A — — — A A A A  
■ A A etc.

Continue repeating a key, until it is close to the end of the line, then step it to the last position in the line and you will notice that the cursor has reverted to a flashing block. This is an indication that the next character to be printed will be on the following line, thus creating a continuation line. A line in NewBrain BASIC can be one 'screen' in length, that is one initial line plus continuation lines.

## CHAPTER 1 – INTRODUCTION

### 4 THE DISPLAY

#### 4.2 Editing the screen display

Editing features are provided in NewBrain Basic so that mistakes can be corrected and features can be added or deleted without affecting the remainder of the PROGRAM. (PROGRAM is defined on page 40). The NewBrain editor is a SCREEN EDITOR, that is changes may be made to characters displayed anywhere on the screen. The line on which the cursor is placed at any time is called the CURRENT line. (When editing a PROGRAM the changes to the current line of the program must be followed by NEWLINE, which will transfer the modified line into memory. If NEWLINE is not pressed, whilst the cursor is on the current line, then the SCREEN display will be edited, but the program will remain unchanged.)

Prior to editing, the cursor must be moved to the position on the display where the edit is to take place. A number of cursor control commands are available to facilitate cursor movement. Many of the cursor control commands and the editing commands which follow are auto repeating, that is they will execute the required action when the appropriate key is pressed, and after a short delay, will repeat the action until the key is released. In the tables which follow, such commands are identified by (R):

### 4 THE DISPLAY

#### 4.3 Cursor Control Commands

→ (R) steps cursor to the right

← (R) steps cursor to the left

↑ (R) steps cursor up

↓ (R) steps cursor down

CONTROL/→ moves cursor to the right end of current line

CONTROL/← moves cursor to the left end of current line

HOME moves cursor to the top left corner of the screen (HOME position)

All the above may be used to position the cursor prior to edigint either a program or the display. PROGRAM EDITS change the content of prgram memory, whereas SCREEN EDITS alter the display but have no effect on program content. NEWLINE is used to enter the current line into the computer.

#### The screen editing commands

SHIFT →

SHIFT ←

INSERT

SHIFT HOME

SHIFT INSERT

CONTROL HOME

SHIFT ↓

GRAPHICS ↑

GRAPHICS ↓

## CHAPTER 1 – INTRODUCTION

### 4 THE DISPLAY

#### 4.4 Screen Editing

##### SHIFT →R

Deletes character(s) above the cursor and moves remainder of the line to the left to close the gap left by the deleted character(s), i.e. deletes characters above and to the right of the cursor.

Example: Type in

NOW IS THE TIME FOR ALL GOOD  
MEN TO COME TO THE AID OF  
THE PARTY

(Do not enter NEWLINE). Use the ← key to place the cursor in the space between TO and THE. Pressing SH/→ deletes the words THE AID OF, resulting in

NOW IS THE TIME FOR ALL GOOD  
MEN TO COME TO THE PARTY

##### SHIFT ←(R)

Deletes character(s) preceding the cursor and moves remainder of the line to the left to fill the gap left by the deleted character(s), i.e. deletes characters to the left of the cursor.

Example: Place the cursor on the space between MEN and TO. Pressing SH/← delete

FOR ALL GOOD MEN

leaving

NOW IS THE TIME TO COME TO  
THE PARTY

### 4 THE DISPLAY

#### 4.4 Screen Editing

##### INSERT

Subsequently typed characters are inserted immediately before the cursor. Insertion is terminated by the cursor control commands → ← ↑ ↓ HOME or by NEWLINE. If the cursor control commands are used to terminate the insert, then NEWLINE must be pressed whilst the cursor is anywhere on the relevant line for the insertion to be effective.

Example: Move the cursor to the space between IS and THE. Press INSERT, SPACE, NOT. The line now reads

NOW IS THE TIME TO COME  
TO THE PARTY

##### SHIFT HOME

Clears the screen, and HOMES the CURSOR.

The keys discussed above allow considerable freedom to type the lines of a program; the following keys provide facilities to move the lines on the page of the screen. As the use of these keys is not immediately obvious, the example should be carefully worked through on the NewBrain, using a monitor or television:

Type SH/HOME to clear the screen, then enter

FOR I = 1 TO 5: PRINT I↑2: NEXT I

## CHAPTER 1 – INTRODUCTION

### 4 THE DISPLAY

#### 4.4 Screen Editing

When NEWLINE is pressed, this command will be obeyed by the computer, which will print the square of each number from 1 to 5 on lines 2 to 6 of the screen.

Now press the HOME key then the INSERT key and type 10.

This will insert a LINE NUMBER in front of the series of commands on the screen, so that BASIC will recognise it as part of a program. That means that when NEWLINE is pressed, the line will be saved in memory. So, press NEWLINE. The cursor will then appear on the next line, just to the left of the number 1.

The cursor now should be a solid flashing block. This is to tell you that typing any character now will clear the line the cursor is on, allowing you to overwrite that line. To check that, type the single key R and see how the number 1 disappears. The small flashing underline form the cursor now has, tells you that the line will not be cleared when the next character is typed. Now type UN to complete the word RUN, and enter the command with NEWLINE.

The same list of numbers is now displayed, but on lines 3 to 7, and below them an error message to tell you that your program has no end statement:

### 4 THE DISPLAY

#### 4.4 Screen Editing

10 for I = 1 to 5: PRINT I↑2: NEXT I

RUN

1  
4  
9  
16  
25

ERROR 3 AT 10:3

This display will be useful in trying out the next control codes.

#### SHIFT INSERT

Moves the current line (i.e. the line on which the cursor is placed) and lower lines downwards leaving a blank line. A line which is scrolled off the bottom of the screen is lost.

Type HOME and then ↓. The cursor will then be resting on the letter R of RUN on the second line. Now type SH/INSERT. The word RUN and all the lines below will shift down one, leaving a blank line for you to use. On this line, type

20 END

and press NEWLINE to enter this as another line of your program. The cursor will again appear as a flashing block, resting on the R of RUN. Type → once to move the cursor to the U, then press NEWLINE, and the program will be executed again.

## CHAPTER 1 – INTRODUCTION

### 4 THE DISPLAY

#### 4.4 Screen Editing

##### CONTROL HOME

Deletes the current line, leaving a blank line.

The cursor is now a flashing block, resting on the E of the error message given last time. There was no error this time; but to prove that, press CTRL/HOME which will delete the error message, and type RUN again, using NEWLINE to enter the command.

The display will now show the 2 line program at the top, then RUN and its results, a blank line, then RUN and the results again.

##### SHIFT ↓ (R)

Deletes the current line and scrolls the remaining lines upwards to fill the gap.

Press ↑ and hold it down while the cursor moves up to the word RUN, halfway up the screen. If you overshoot, use ↓ to come back down. Then press SH/↓ and the word RUN will disappear. Press SH/↓ again and hold it down and all the lines below will also disappear.

##### GRAPHICS ↑

One “line” of text may in fact occupy more than one row of characters on the screen. This is shown by a CONTINUATION MARK at the start of the second and subsequent rows. To see this, type

### 4 THE DISPLAY

#### 4.4 Screen Editing

HOME then → and hold the → down until the cursor reaches the first colon, before PRINT.

Next press the space bar, which will overwrite the colon, then INSERT and hold down RPT/SPACE (don't worry if the P of PRINT seems to disappear) until the words “PRINT I↑2: NEXT I” have all come well onto the second row. Release the space bar, and a continuation mark will be visible at the start of the second row:

```
10 FOR I=1 TO 5
    ■ PRINT I↑2: NEXT I
```

GR/↑ functions to split a continued line into two lines. The row on which the cursor appears becomes the first row of the newly formed line.

In the example, with the cursor still on the P of PRINT, press GR/↑. The cursor will then appear at the start of the line, and the continuation mark will disappear. Now type 12 and NEWLINE to enter line 12 into memory:

```
10 FOR I=1 TO 5
12     ■ PRINT I↑2: NEXT I
20 END
```

Note at this stage that line 10 is still in the computer's memory as originally entered. It may be quickly altered by pressing HOME and NEWLINE.

## CHAPTER 1 – INTRODUCTION

### 4 THE DISPLAY

#### 4.4 Screen Editing

If the example has been followed up to this point, it should be fairly easy to again use INSERT and GR/↓ to break up line 12 into two lines, say

```
12 PRINT I↑2  
13 NEXT I
```

Remember to enter **both** altered lines with NEWLINE, however. You may now enter LIST and NEWLINE to display the program

```
10 FOR I=1 TO 5  
12      PRINT I↑2  
13 NEXT I  
20 END
```

#### GRAPHICS/↓

This joins two lines of text on the screen into a single line, by making the current line a continuation of the line above and introducing one continuation mark.

Example: place the cursor on the line number 20 then press SH/→ twice to delete the number 20. Next type GR/↓ introducing a continuation mark, and hold down SH/← until the word END has nearly reached NEXT I on the preceding line. Finally, type INSERT, then a colon, then NEWLINE. You will then have formed one line out of the two lines shown on the screen, numbered 13 and 20.

### 4 THE DISPLAY

#### 4.5 Line Display (Model AD Only)

The NewBrain AD directs output to either a LINE or SCREEN display, or to both line and screen. When first switched on the AD model will default to the combined display. Line display is selected by the command OPEN#0,3 (see page 4) whereupon the screen will blank, and a flashing cursor will appear in the left-most position of the line display. Screen display is selected by the command OPEN#0,0. Output can be directed to both displays by the command OPEN #0,4.

The line display is a "window", 16 characters wide, onto the current line. The window may be moved across the line by the cursor control keys → and ←. Whatever the position of the cursor in the display, it first moves to the right or left end of the window, and then pushes the window along the line, stopping when either end of the line is reached. When OPEN#0,4 has been used, editing is exactly as described above for the screen display, and the window moves from one line to another. When OPEN#0,3 is used however, the editor is now a line editor, that is it will only accept those commands whose area of operation is confined within one line. Thus, the following commands will not work if the line display only is being used.

# CHAPTER 1 – INTRODUCTION

## 4 THE DISPLAY

### 4.5 Line Display (Model AD Only)

↑ and ↓

SH/INSERT (insert a blank line)

SH/↓ (delete the current line)

GR/↓ (insert continuation mark)

HOME will return the cursor to the leftmost position in the line.

Note that all editing commands are effective if both displays are in use.

The line display will present output from a program, or LIST a program, one line at a time. To continue the output, or the LISTing, press NEWLINE. Thus the program:

```
1 PRINT 1
2 PRINT 2
3 PRINT 3
4 END
```

will print first 1, then 2, then 3, as NEWLINE is repeatedly pressed. Similarly, if LISTed the display will show lines 1, 2, etc. as NEWLINE is repeatedly pressed. A program line which is to be edited using the line display only, must be listed, thus:

LIST 40

to present line 40 for editing.

The line display can represent all the characters shown on the NewBrain keyboard. Alphabetic characters are always upper case. The limitations of

## 4 THE DISPLAY

### Line Display (Model AD Only)

the line display cause the representations of a few of the less-frequently used characters to be somewhat stylised. The user should therefore make himself familiar with the display characters. With the exception of the pound sign (£), the complete character set may be displayed by the following simple program, which also demonstrates the operation of the LINE display. This program may be entered with the screen selected, but should be RUN with only the line display selected (OPEN#0,3).

```
10 D=200
20 FOR I=32 TO 95
30 PRINT CHR$(I);
40 FOR T=I TO D:NEXT T
50 NEXT I
60 END
```

When the program is RUN the character set will be displayed, with a small time delay between characters set by line 10. Note how the window presented by the display moves as the characters reach the end of the display. When the program has finished the window moves back to the beginning of the output line, and the cursor appears in the leftmost position. The character set may be examined by using the cursor control keys → and ← to move the window across the output line. The characters displayed are:

```
SPACE ! " # $ % & ' ( ) * + , - . /
0 to 9
: ; < = > ? A to Z [ \ ] ↑
```

## CHAPTER 1 – INTRODUCTION

### 4 THE DISPLAY

#### 4.5 Line Display (Model AD Only)

and a final character !^ which is used to indicate that the character sent to the display cannot be represented, for example a graphics symbol. The line display character set is illustrated in Appendix 5.

### 5 THE OPERATING SYSTEM

#### 5.1 Overview

The remaining sections of this chapter provide the experienced computer user with an introduction to various features which are specific to the NewBrain, and should be read in conjunction with the Appendices. The reader who is not familiar with BASIC should read the Introduction to BASIC in Chapter 2–9, working through the examples with a NewBrain and a TV screen wherever possible, before returning to finish this chapter.

The NewBrain operating software consists of three parts which are able to function almost independently from one another. These parts are the OPERATING SYSTEM itself, the DEVICE DRIVERS, and the BASIC COMPILER. The operating system includes the control of input and output (the INPUT–OUTPUT SYSTEM), all the memory checking and other routines required on power-up, and a RESTART mechanism described below, as well as a powerful mathematics package. The device drivers handle input and output to all peripherals, including the screen display and keyboard, and the BASIC COMPILER is automatically given control of the NewBrain after the computer is switched on.

## CHAPTER 1 – INTRODUCTION

### 5 THE OPERATING SYSTEM

#### 5.2 The Input-Output System

In order to move data between the USER Program (normally BASIC) and the various peripherals, a set of DEVICE DRIVERS is provided. Each of these is designed to "look the same" to the operating system, and therefore to BASIC, while meeting the specialised needs of each corresponding peripheral device. The OPEN statement is used to set up a numbered data STREAM as a channel from the program itself to a peripheral. After the OPEN statement has been executed, the numbered stream remains associated with the device type given in the OPEN, e.g.

OPEN#3,8

opens stream number 3 as a printer stream. Thereafter any output to stream 3 would be directed to the printer connections at the back of the NewBrain. This association remains in force until the stream is closed. The available device drivers are summarised in Appendix 7, together with examples of the OPEN statement for each device type.

The Z80 processor provides a number of hardware PORTS for the input and output channels to various peripherals. Certain of these ports are reserved for specific NewBrain functions, such as TV control to maintain the screen display, and the Enable and Status registers which are related to other aspects of the hardware. A BASIC program may use the device type 7 to direct a stream to use

### 5 THE OPERATING SYSTEM

#### 5.2 The Input-Output System

any selected port for byte-oriented input or output, normally using the USER INPUT parallel port (21 for model A, 20 for model AD) and the USER OUTPUT parallel port (number 3). These can provide a means of accessing digital instruments directly from the NewBrain when it is fitted with a suitable expansion box, using statements like

```
OPEN#20, 7, 20  
GET# 20, x
```

#### 5.3 Extension

The NewBrain operating system is designed to make it practicable to extend the facilities available at any time. This will be illustrated by reference to the device drivers. A table of the standard device drivers is provided, and the starting address of this table is written into a particular memory location. Either firmware or an extra package contained in a ROM expansion box may extend this table by copying it into a suitable area of memory, with any desired alterations or additions, then inserting the address of this copy of the table into the correct location. This method is suitable for providing an extra type of device driver, or for making changes to an existing device driver. In either case, the code for the new driver or for the alteration must also be provided.

A similar method of extension makes use

## CHAPTER 1 – INTRODUCTION

### 5 THE OPERATING SYSTEM

#### 5.3 Extension

of a RESTART MECHANISM in the processor. The various functions provided in the Operating System and in BASIC are accessed when required through a table of routines, which can be altered or extended in the same way. This means that additional features can be provided in BASIC, or added to the operating system, simply by plugging an expansion box into the NewBrain before switching on. Some of the operating system routines are described in Appendix 8.

### 6 USING CASSETTE RECORDERS

#### 6.1 Connection

There are two sockets on the NewBrain for cassette recorder leads, labelled TAPE 1 and TAPE 2. Either may be used for programs or data according to preference; however, the BASIC LOAD and SAVE commands use TAPE 1 as a default, so it is usual to connect a cassette recorder to this socket for loading programs.

#### 6.2 Load

In order to load a BASIC program from tape, first plug in the tape recorder to the TAPE 1 socket. Insert the tape, wound forward to the correct position, and depress the PLAY button on the tape recorder. Next enter LOAD on the NewBrain. The tape will be scanned for a file header, and if the file has a title that title will be displayed on the screen. The computer will read the entire program file into memory, showing a solid cursor when the file has been completely read in.

The NewBrain may be interrupted when reading from a tape by pressing the asterisk key “\*”; at all other times, it may be interrupted by pressing the STOP key.

#### 6.3 Save

To save a BASIC program on tape, plug in the tape recorder as described above, and depress the RECORD (or RECORD and PLAY) switch(es).

## CHAPTER 1 – INTRODUCTION

### 6 USING CASSETTE RECORDERS

#### 6.3 Save

The program may be saved with a title, by entering e.g.

SAVE "accts program"

or without a title, by entering simply

SAVE

The computer will then output the BASIC program from its memory to the tape, displaying a solid cursor when this has been completed. The program remains in memory, and may be checked with the VERIFY command, or a further copy put on the next part of the tape by entering SAVE (and a title if wanted) again.

#### 6.4 Verify

To check that a program has been saved correctly, after saving as many copies as desired, press the STOP key on the tape recorder and then the REWIND. No tape motion will result at this stage. Then enter VERIFY on the NewBrain keyboard. The tape will wind back, and must be stopped just before the first copy of the program (or at the beginning of the tape). Then press the PLAY button on the tape recorder. The NewBrain will display the title of the file on the tape if there is one, then read it through, comparing it with the program in its memory. If the saved copy matches the program in memory, "VERIFIED" will

### 6 USING CASSETTE RECORDERS

#### 6.4 Verify

be displayed, otherwise an error message (e.g. ERROR 91) will be displayed. The VERIFY command may be stopped by using the asterisk key “\*\*”, as for LOAD above.

Note that the program in memory is not changed as a result of VERIFY. The VERIFY command is often useful as a way of simply releasing the cassette recorder from the NewBrain's control: enter VERIFY, then use the cassette recorder controls to position the tape to a desired point for any operation, e.g. LOAD. After pressing the STOP button on the cassette recorder, press the asterisk on the NewBrain to cancel the VERIFY command. This method is preferable to continually unplugging one end or the other of the cassette lead!

#### 6.5 The Tape 2 Socket

The SAVE command outputs the BASIC: program in its space-saving form. It may be specified with a stream number, e.g.

SAVE # 12

but in this case the stream must be open. This form of the command must not contain a file title, and although the stream may in fact be of any type, the most useful application is to save a program on a tape recorder that is plugged in to the TAPE 2 socket, as e.g.

## CHAPTER 1 – INTRODUCTION

### 6 USING CASSETTE RECORDERS

#### 6.5 OPEN #12,2 Tape 2 Socket

SAVE # 12  
CLOSE # 12

The first command in this example selects a stream called 12 of type 2, which is the device type that handles the TAPE 2 outlet. The tape will move at this stage, to write the file header, and the cassette recorder must therefore be set to RECORD. The next command saves the current BASIC program on the tape via stream 12, so the cassette recorder must remain set to RECORD.

If a short BASIC program is saved in this way, it may happen that no tape motion is seen after entering SAVE. This will not cause any problem, as the last block in any tape handling process is not written to the tape until a CLOSE is executed. If the entire program is less than one block, then it will be written out in response to the CLOSE command.

OPEN # 12,2

The program may also be verified on TAPE 2. Enter ~~VERIF~~ then press REWIND on the tape recorder to bring the tape back to the start of the file. When the tape is correctly positioned, pass PLAY and the NewBrain will then read the file header and display the file title. ~~VERIF~~ may then enter

### 6 USING CASSETTE RECORDERS

#### 6.5 The Tape 2 Socket

CLOSE # 12

to check the program in memory against the saved copy, and ~~VERIF~~ when the verification is completed. The tape recorder may then be switched off and disconnected.

## CHAPTER 1 – INTRODUCTION

### 7 USING A PRINTER

Any “byte-serial” printer which uses the RS232 standard interface may be connected to the NewBrain to print documents, list programs or provide hard-copy output from a program. First a printer stream must be opened, e.g.

OPEN # 18,8

then the printer may be switched on and the printer lead connected to the back of the NewBrain. Any output statement may then be used, with output directed to the selected stream:

LIST # 18

prints a copy of the entire program

LIST # 18, 200--

prints the program from line number 200 to the end

PRINT # 18, "Amount payable .....  
£", P [5.2]

prints the legend given in quotes followed by the value of P.

The formatting specification used in this example is particularly valuable for printed output, allowing the digits of numbers to be correctly aligned with one another.



## **CHAPTER 2**

### **BASIC DEFINITIONS**

This chapter provides introductory definitions of BASIC terms essential to an understanding of the remainder of the handbook.

- 1 INTRODUCTION**
- 2 NUMERIC CONDITIONS**
- 3 VARIABLES**
- 4 ARRAY VARIABLES**
- 5 EXPRESSIONS**
- 6 ERROR MESSAGES**

## CHAPTER 2 – BASIC DEFINITIONS

### 1 INTRODUCTION

### 1 INTRODUCTION

NewBrain BASIC will accept input lines with or without line numbers. Lines which are entered with line numbers are not executed, but are added to the current program in memory. The program is not executed until a suitable command is entered, at which time control of the computer passes to the program. Lines without the numbers are commands – they are executed immediately NEWLINE is pressed. NewBrain BASIC will accept commands (be in COMMAND MODE) unless control has already passed to the computer because it is executing either another command, or a program. Command Mode is re-entered when:-

- an END or STOP or LOAD statement occurs
- an ERROR occurs
- the STOP key is pressed
- the current command is completed, unless it transfers control to a program.

A BASIC PROGRAM is a sequence of numbered lines. Each line consists of a number of BASIC STATEMENTS separated by colons, thus:-

NNNN BASIC STATEMENT: BASIC STATEMENT: ---

where NNNN is the line number. Line numbers must lie in the range 1 to 65,535. Program lines are executed in numerical order, commencing with the lowest

numbered lines. It is good practice to use an increment of say 10 between successive line numbers to allow for later insertions, thus:

```
10 LINE 1
20 LINE 2
30 LINE 3
40 LINE 4
```

A BASIC STATEMENT consists of a KEYWORD followed by a list of parameters in a format which is specific to each keyword. The keyword identifies the operation to be performed by the computer. The parameter(s) refer either literally or symbolically to the data to be processed. A parameter may often be an EXPRESSION.

## CHAPTER 2 – BASIC DEFINITIONS

### 2 NUMERIC CONSTANTS

NewBrain BASIC accepts integers, floating point real numbers, or strings as constants. Some examples of acceptable numeric constants are:-

276  
3.141592  
0.0716  
1.234E05

Numbers input from the console or numeric constants in a program may have any number of digits up to the length of a line, but are stored internally to a precision of 10 or more significant figures, and are output, by default, rounded to 8 significant figures. Thus the command:-

PRINT 3.141592653589

produces the output:

3.1415927

Numbers are printed in INTEGER, FLOATING POINT, or SCIENTIFIC notation. Integers are whole numbers, with no fractional or decimal component, e.g.

0,102, +4, -36, 1000

Floating point numbers have a decimal point, the position of which may vary from number to number (hence floating point), thus

3.732, -0.358, 6352.961

### 2 NUMERIC CONSTANTS

In scientific notation numbers are represented as a fixed point number, the MANTISSA, and an EXPONENT, which indicates how many powers of ten the mantissa should be multiplied by. The format of a scientific number is:

SN.NNNNNNNNESMM

where S is the sign (always printed in the exponent, omitted, if positive, in the mantissa), NNN--- are the digits of the mantissa, printed in fixed point format, one digit before the decimal point, E is an abbreviation of Exponent, and MM are the digits of the exponent. The following are equivalent numbers

1.2345E9 123.45E7 0.0012345E12

where the change of position of the decimal point is compensated by the change of the exponent. NewBrain BASIC will print the number as:

1.2345E+09

Numbers outside the range 99 999 999 to 0.00001 are printed in scientific notation.

Number	NewBrain BASIC output
100 000 000	1 E+08
99 999 999	99999999
0.00001	.00001
0.000099	9.9E-06

## CHAPTER 2 – BASIC DEFINITIONS

### 2 NUMERIC CONSTANTS

Numbers are stored internally in the range  $-10^{150}$  to  $+10^{150}$ . Numbers are output however, in the range  $-10^9$  to  $+10^9$ . Thus the command

```
PRINT 1E60 * 1E90/1E80
```

(where \* is used to mean multiply, see page 28 correctly yields the result  
1E+70.

An attempt to input a number outside the range  $-1E99$  to  $1E99$  will result in an error message. An attempt to output a number outside the range  $-1E99$  to  $1E99$  will result in a display of \*\*\*\*\*. Thus:

```
PRINT 1E60* 1E90/1E80
```

is acceptable, whereas

```
PRINT 1E150/1E80
```

is not.

Note that numbers in scientific notation must have a mantissa (1E60 is acceptable, E60 will produce an ERROR message).

The format of a number output by the computer is thus as follows:

1. If the number is positive, a space is output, followed by the number. If the number is negative, a minus sign is output, followed by the number.

### 2 NUMERIC CONSTANTS

2. If the absolute value of a number is an integer in the range 0 to 99 999 999 it is printed as an integer.

3. If the absolute value of a number is greater than or equal to 0.0001 and less than or equal to 99 999 999 it is printed in fixed point notation with no exponent.

4. If the number is outside the ranges in 2 and 3 above, it is printed in scientific notation (with an exponent). In scientific notation non-significant zeros are suppressed in the mantissa, but two digits are always printed in the exponent.

A space is printed after a number in all formats. Thus the input:

```
T=1E15 Y=-.56.78; D=.0342  
PRINT T;Y;D
```

yields the result.

```
1E+15 -56.78 .0342
```

Certain BASIC reserved words (q.v.) are constants, for example the mathematical constant  $\pi$ , which in BASIC is written PI. (See 66 )

## CHAPTER 2 – BASIC DEFINITIONS

### 3 VARIABLES

Numbers can be represented symbolically by letters, called variables. The value may be set explicitly by the programmer,

LET A = 22.35

or may be assigned as a result of calculations in a program

LET C = 2\*PI\*R\*R

Before a variable is assigned a value, it has the value zero. Variable names can be one or two characters long. The first character must be a letter (A–Z), the second may be a letter or number (A–Z, 0–9). The letter(s) may be upper or lower case; NewBrain BASIC does not distinguish between the two. Examples of valid variable names are A, A3, AC.

A variable may also represent a STRING which is any sequence of letters or other characters. String variables may use any one or two-character name (alphabetic followed by alphanumeric as for a variable) followed by a dollar sign \$.

Thus string variables may be labelled A\$, A3\$, AC\$, etc. The string itself is delimited by quotation marks at the beginning and end. Strings are assigned in the same way as numeric variables,

### 3 VARIABLES

A\$ = "Hello"  
A3\$ = "NewBrain"  
AC\$ = "What is your name?"  
D\$ = "72/36?"

and output in the same way,

PRINT AC\$

Press NEWLINE

What is your name?

Strings may be of any length from 0 to 32767 characters long.

N.B. Variables cannot be given the following names, since each is a BASIC RESERVED WORD (see Appendix 4).

TO	TO\$
ON	ONS\$
OR	ORS\$
IF	IFS\$
PI	PIS\$
FN	FNS\$

## CHAPTER 2 – BASIC DEFINITIONS

### 4 ARRAY VARIABLES

An array is a table of values, stored with a common name, where each element of the table is identified by the array subscript. Thus A(3) represents the third element of the array called A. F(4,2) is the element in the fourth column, second row of the array F. The array A above is one-dimensional, whereas F is two-dimensional. Arrays may not have more than two dimensions and the maximum number of elements in an array is 5374. The contents of an array may be numbers, or strings, but string arrays must have a string name, e.g.:

C\$ (4), A3\$ (5,3), BB\$ (17)

In a string array each element is a single string, e.g. C\$ (2) = "Goodbye".

A name used for an array may also be used for a different scalar (i.e. single element) variable of the same type, thus C3 (4) and C3 may be used concurrently, as can BB\$ and BB\$ (12). Arrays must be DIMENSIONED before use (q.v.), which sets all elements of an array to zero.

### 5 EXPRESSIONS

An expression consists of any or all of the previously defined items, that is

Constant	e.g.	42
String	e.g.	"Hello"
Variable	e.g.	A
Array Element	e.g.	C (4, B+2)

plus the intrinsic functions of BASIC, for example SQR (SQSquare Root, see Chapter 7), linked by a number of OPERATORS.

#### 5.1 Arithmetic Expressions

– use the arithmetic operators

+	plus	3+2 = 5
-	minus	3-2 = 1
/	divide	3/2 = 1½
*	multiply	3*2 = 6
↑	raise to the power	3↑2 = 9

and in addition

preceding + leaves the sign of the following number or variable unchanged:  $x = +y$   
preceding - changes the sign of the following number or variable:  $x = -y$

Note that the up-arrow ↑ used for raising to a power is SH/+ and not the cursor control up-arrow.

## CHAPTER 2 – BASIC DEFINITIONS

### 5 EXPRESSIONS

#### 5.2 Logical Expressions

There are two types of operator in logical expressions, the logical operators AND, OR and NOT, and the RELATIONAL operators. These latter consist of

=	equals
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
<>	less than or greater than (not equal)

When used in an expression the relation is evaluated and the result stored as either TRUE if the relationship is valid or FALSE if invalid. These may be linked by the logical operators AND, OR and NOT. In order to describe these a few notes on BINARY ARITHMETIC are required.

The familiar decimal arithmetic uses the position of a digit in a number to indicate the power of 10 corresponding to the digit, which may be 0 to 9. For example

$$\begin{aligned}234_{10} &= 4 \times 10^0 + 3 \times 10^1 + 2 \times 10^2 \\&= 4 \times 1 + 3 \times 10 + 2 \times 100\end{aligned}$$

where the subscript  $_{10}$  indicates a decimal number. Computers however perform arithmetic using binary numbers, that is numbers consisting of BInary digiTs (BITS), 0 or 1. Thus

### 5 EXPRESSIONS

#### 5.2 Logical Expressions

$$\begin{aligned}1101_2 &= 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 \\&\quad + 1 \times 2^3 \\&= 1 \times 1 + 0 \times 2 + 1 \times 4 \\&\quad + 1 \times 8 \\&= 13_{10}\end{aligned}$$

where  $1101_2$  is a 4-BIT binary number. The rightmost bit, bit 0 is the least-significant bit, the leftmost bit, bit 3 is the most-significant bit. In practice arithmetic within the NewBrain is performed using 8-bit numbers. Bit 7 is often used as a SIGN-BIT, indicating a positive number if bit 7 = 0, a negative number if bit 7 = 1.

#### NOT

The NOT operation changes each bit in a number from 0 to 1, or from 1 to 0. This may be shown by a TRUTH TABLE which lists all the possible short conditions, in this case B = 0 or B = 1, and the result following the operation.

B	NOT B
0	1
1	0

Therefore

$$\begin{aligned}\text{NOT } 0000\ 1011 &(11_{10}) \\&= 1111\ 0100 (-12_{10})\end{aligned}$$

If N is a decimal number then

$$\text{NOT } (N) = -(N+1)$$

e.g.

$$\text{NOT } B = -7 \text{ and NOT } -7 = B$$

## CHAPTER 2 – BASIC DEFINITIONS

### 5 EXPRESSIONS

#### 5.2 Logical Expressions

Results of the relational operations, e.g. IF A = B, are stored as -1 if TRUE (in this case A equal to B) or 0 if FALSE (A not equal to B). Thus the command

PRINT TRUE, FALSE

produces the result,

-1      0

Note in particular that

NOT -1 = 0, NOT 0 = -1

or

NOT TRUE = FALSE, NOT FALSE  
= TRUE

AND

AND operates according to the Truth Table

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

setting a bit in the result to 1 only if the corresponding bits in the input words are both 1, i.e.

(A AND B) = 1 if A = 1 AND B = 1

### 5 EXPRESSIONS

#### 5.2 Logical Expressions

For example

AND	0 0 1 1	1 1 1 1	(63 <sub>10</sub> )
=	0 0 0 1	1 0 1 1	(27 <sub>10</sub> )
		1 0 1 1	(27 <sub>10</sub> )

and

AND	1 1 1 1	1 1 1 1	(-1 = TRUE)
=	0 0 0 0	0 0 0 0	(0 = FALSE)
	0 0 0 0	0 0 0 0	FALSE

i.e. TRUE AND FALSE = FALSE

OR

OR has the Truth Table

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

that is (A OR B) = 1 if A = 1 or B = 1

OR	0 0 1 1	1 1 1 1	63 <sub>10</sub>
=	0 0 0 1	1 0 1 1	27 <sub>10</sub>
	0 0 1 1	1 1 1 1	63 <sub>10</sub>

OR	1 1 1 1	1 1 1 1	TRUE
=	0 0 0 0	0 0 0 0	FALSE
	1 1 1 1	1 1 1 1	TRUE

i.e. TRUE OR FALSE = TRUE

## CHAPTER 2 – BASIC DEFINITIONS

### 5 EXPRESSIONS

#### 5.3 String Expressions

The operation of joining two strings is called CONCATENATION. Two operators may be used to concatenate strings, + or &.

Thus "GOOD" + "BYE"  
= "GOOD" & "BYE"  
= "GOODBYE"

Strings may also be tested by the relational operators. Effectively the ASCII code (q.v.) for each character is compared on a character by character basis. Thus this context

"a" = "a"  
"a" < "b"  
"A" < "a" (i.e. Upper case < Lower case)  
"a" < "a a"

Thus "a" < "an" < "and < ant"

NewBrain BASIC provides functions to handle parts of strings, described in Chapter 7.

### 5 EXPRESSIONS

#### 5.4 Precedence

Where an expression contains more than one operator these are evaluated in the order preceding + or - leave or change sign.

↑ raising to a power  
\* / multiplication and division  
+ - &  
< < = = > > = <>  
NOT  
AND  
OR

The precedence of operators may be changed by inserting parentheses, thus

4 \* 6 + 6 - 2 = 24  
4 \* (5 + 6 - 2) = 36  
4 \* (5 + 6) - 2 = 42

NESTED parentheses are always evaluated from the innermost parentheses outwards.

## CHAPTER 2 – BASIC DEFINITIONS

### 6 ERROR MESSAGES

### 6 ERROR MESSAGES

NewBrain BASIC provides a comprehensive range of error codes to assist the user in identifying faults in programs. Error messages take the form

**ERROR XXX AT NN**

where XXX is the error code number, and NN is the line number at which the error occurred. If the error occurred in a multi-statement line, then the error message takes the form

**ERROR XXX AT NN: P**

where P indicates the first invalid statement on the line. Thus

**10 PRINT A, B: GOTO 100**

produces the message

**ERROR 29 AT 10: 2**

if line 100 is non-existent when the program is executed.

One error is detected on entry, i.e. following NEWLINE. The line

**99999 PRINT A (NEWLINE)**

immediately produces

**ERROR 4**

indicating an illegal line number.

Other errors are detected when a program is run, or when a command is entered for immediate execution. A comprehensive list of error code numbers is contained in Appendix 1.

## **CHAPTER 3**

### **SIMPLE BASIC**

This chapter outlines the elementary commands necessary to get a program "up and running", including the assignment statement (LET), input from the keyboard, output to the display, and the commands to start and stop execution of a program.

- 1. ASSIGNMENT – LET**
- 2. PRINT**
- 3. TAB**
- 4. INPUT**
- 5. LIST**
  - 5.1 List – Screen Display**
  - 5.2 List – Line Display**
- 6. RUN, END AND GOTO**
- 7. STOP AND CONTINUE**
- 8. REM**

## CHAPTER 3 – SIMPLE BASIC

### 1 ASSIGNMENT -- LET

Variables are set to a desired value using =. Thus, to assign the value 26.35 to a variable called G the statement

```
LET G = 26.35
```

is used. Similarly, one may assign a value to a string variable

```
LET Y$ = "YES"
```

The assignment may consist of equating a variable to an expression, thus

```
LET C = 2*PI*R  
LET P = C*A*(1-N/R).  
LET D = D + 1
```

In BASIC this last example means "take the value stored in the box with the label D (four, say), add one to it (making five) and put this back in the box labelled D". Thus D becomes D+1. Variables, like D, may change their value during the course of the program.

Example: Print the surface area of a cylinder

```
10 LET R = 2  
20 LET H = 3  
30 LET A = PI * R * R  
40 LET C = 2 * PI * R  
50 LET S = H * C  
60 LET S = S + 2 * A  
70 PRINT S  
80 END
```

### 1 ASSIGNMENT – LET

ASSIGNMENT is so frequently used that the keyword LET may be omitted.

```
10 R = 2  
20 H = 3  
30 A = PI * R * R  
40 C = 2 * PI * R  
50 S = H * C  
60 S = S + 2 * A  
70 PRINT S  
80 END
```

## CHAPTER 3 – SIMPLE BASIC

### 2 PRINT

This is one of the most versatile statements in BASIC. It is used to output data from the computer, normally to the screen or line display. PRINT may be followed by a number, a numeric variable, a string, a string variable, or an expression, as in the examples which follow.

PRINT 32.76 produces the result  
32.76

A = 4: PRINT A produces the result  
4

The value of the variable is printed, not its name. Where the item to be printed is a number, its format may be completely controlled by using a FORMATTING SPECIFICATION (Appendix 2 § 4.1.4). A different formatting specification may be used for each item. Thus,

X = 1612.24: PRINT X [4.2], X [5.1]

yields

1612.24 1612.2

The symbol ? may be used as a synonym for PRINT, as in

? X [1.3E] giving

1.612E+03

and

? "Hello" giving

### 2 PRINT

Hello

The text within the quotation marks is printed, exactly as typed in. Similarly for a string variable:

A\$ = "string var": PRINT A\$

results in

String var

If an expression follows a print statement then the expression is evaluated and the value is printed.

PRINT 32 \* 5/8  
20

A print statement may have any combination of parameters, so long as the list will fit on a line (including continuation lines). The parameters in a print statement must be separated by semi-colons or commas. The effect of the semi-colons or commas is to format the printing. A semi-colon instructs the display device not to advance the cursor before printing the next character:-

PRINT "GOOD"; "BYE"  
GOODBYE

A comma will cause the next character to be printed at the start of the next ZONE. The page upon which characters are PRINTed is divided into PRINT-ZONES. The print-zones on the screen are 10

## CHAPTER 3 -- SIMPLE BASIC

### 2 PRINT

characters wide. Thus

```
PRINT "GOOD", "BYE"
```

results in

```
GOOD     BYE
```

where the B of BYE is placed on the 11th position from the left edge of the display.

```
PRINT "Z1", "Z2", "Z3"
```

will indicate the start of the print zones on the screen.

```
Z1      Z22     Z23
```

The output from a PRINT statement can be directed to a PRINT ZONE by preceding it with the appropriate number of commas. Thus PRINT , , , 21 causes 21 to be positioned in the fourth print zone. Note that the first character of a positive number is a blank (in place of the understood positive sign), so the first digit of a positive number is placed in the second character position in the print zone.

### 3 TAB

TAB may be used to place the output at any character position along the line. TAB is used only in PRINT statements. It has the form TAB(n) which places the cursor n positions from the start of a line.

```
PRINT TAB (12), "SUM"  
                  SUM
```

Note that TAB, like other parameters to the PRINT statement must be separate from subsequent parameters by a semi-colon or a comma. However, if a comma is used, the cursor is immediately advanced to the start of the next print-zone.

Multiple tabs may be used, thus

```
PRINT TAB(10); "TEN"; TAB(20); "TWENTY"  
          TEN        TWENTY
```

but if the tab counts are such as to make one piece of text overwrite another, then a new line is forced:-

```
PRINT TAB(10); "TEN"; TAB(12); "TWELVE"  
          TEN        TWELVE
```

A new line is also forced if an attempt is made to print a number or string towards the end of a line, where the spaces remaining at the end of the line are not sufficient to contain all the characters making up the number or string. PRINT PI will output PI to 7 decimal places. With a leading space and a decimal point the output will thus occupy ten character

## CHAPTER 3 – SIMPLE BASIC

### 3 TAB

positions. So, when the screen is forty characters wide,

```
PRINT TAB(30); PI
```

will print 3.1415927 tabbed 30 spaces out in the line following the print statement, whereas PRINT TAB(31), PI results in a blank line, followed by PI, printed on the next line, in the normal position:

```
PRINT TAB(31); PI
```

```
3.1415927
```

The same result will be obtained for any tab from 31 to 40. Tabs greater than 40 are "reduced modulo the line length", that is the tab, for example 218, is divided by the line length (40) and the remainder (18) is taken as the tab value. Thus TAB(218) is equivalent to TAB(18). The maximum number permitted in a tab statement is 65,535 (equivalent to TAB(15)), the minimum is 1.

TAB and , are only effective when printing to the screen. For other devices alternative means must be used for print formatting.

### 4 INPUT

INPUT is used to collect data from an input device, normally the keyboard. Its function is to request a number or a string which must then be assigned to a variable.

Example:

INPUT A	— numeric
INPUT A\$	— string
INPUT A, B, C, A\$	— mixed

When this simple form is used a prompt (question mark, space) is issued to the display, thus

```
INPUT A  
?—
```

The computer waits for the requested data to be entered, followed by NEWLINE, thus

```
INPUT A$  
? FRED
```

Where more than one item is to be input (third example above) they must be separated by a comma,

```
INPUT A, B, C, A$  
? 34, 56, 3.87, FRED
```

If either too many or too few items are input, an error message will be displayed. If a program is being executed, the prompt will then be repeated.

## CHAPTER 3 – SIMPLE BASIC

### 4 INPUT

Where the input is a string, it may not of course contain a NEWLINE character and it need not be input with its enclosing quotes. However, if the string contains embedded quotes, for example She said "Help" then the string must have closing quotation marks and the embedded quotation marks must be duplicated, thus

```
INPUT D$  
? "She said" "Help" "
```

```
PRINT D$  
She said "Help"
```

The prompt (question mark, space) passed to the console may be usefully replaced by a prompt expression, in the form

```
INPUT (PROMPT EXPRESSION)  
VARIABLE(S)
```

for example

```
INPUT ("BLACK OR WHITE") PCS  
BLACK OR WHITE —
```

Again, if an incorrect response is given to the prompt, then the user is re-prompted with the prompt expression. If no prompt whatsoever is desired, the NULL STRING " " is used

```
INPUT (" ") A
```

### 5 LIST

#### 5.1 LIST – Screen Display

LIST is used to output the program to an output device (screen, printer, etc). By default, LISTing takes place on the screen.

LIST has two parameters separated by a minus sign, thus LIST 10–100. The parameters represent the start and end points of the LIST. If either parameter is omitted the start or end of the program is assumed.

```
LIST 10 lists line 10  
LIST 10–100 lists lines 10 to 100  
inclusive  
LIST –100 lists up to and including  
line 100  
LIST 100— lists from line 100 onwards  
LIST — lists complete program  
LIST lists complete program
```

The first parameter must be less than the second, thus LIST 200–100 has no effect.

## CHAPTER 3 – SIMPLE BASIC

### 5 LIST

#### 5.2 List – Line Display

When the line display alone is in use, LIST operates as above, except that only one line of a program is displayed at a time. Subsequent lines are displayed by pressing NEWLINE repeatedly. When it is desired to edit a program line (say line 50) then LIST 50 must be entered. If

LIST 10–100

were used, then the computer would not return to command mode (when editing could commence) until the LIST command has completed, i.e. lines 10 to 100 had all been displayed.

### 6 RUN, END AND GOTO

RUN is used to start execution of a program stored in memory. Prior to execution, all variables are cleared to zero. Thus the simple program

```
10 PRINT a, b, c  
20 a = 4: b = 3: c = 2  
30 PRINT a, b, c  
40 END
```

always produces the results

0	0	0
4	3	2

each time it is RUN, i.e. a, b, c (set to 4, 3, 2 in line 20) are reset to zero each time RUN is entered.

Notice that the program finishes with

```
40 END
```

This permits more than one program to be stored in memory at the same time. RUN will always commence execution at the lowest numbered line available, and execution will cease when END is encountered. A second or subsequent program may be executed by the command GOTO XXXX where XXXX is the starting line number of the program to be executed. If the program above is extended to read as follows:-

```
10 PRINT a, b, c  
20 a = 4: b = 3: c = 2  
30 PRINT a, b, c  
40 END
```

## CHAPTER 3 – SIMPLE BASIC

### 6 RUN, END AND GOTO

```
100 PRINT d, e, f  
110 d = 5: e = 6: f = 7  
120 PRINT d, e, f  
130 END
```

RUN will cause execution of lines 10 to 40, whereas GOTO 100 will cause execution of lines 100 to 130. Note that unlike RUN, GOTO will not clear variables, thus the first GOTO 100 results in

0	0	0
5	6	7

whereas subsequent GOTO 100 commands yield

5	6	7
5	6	7

GOTO may be used within a program to alter the sequence in which BASIC executes the lines of a program. The following trivial program will always loop back to line 10 from line 30

```
10 INPUT ("YOUR NAME? ")  
    AS  
20 PRINT TAB (7); "HELLO ";  
    AS  
30 GOTO 10  
40 PRINT "THIS LINE WILL"  
      NEVER BE PRINTED"  
50 END
```

and the only way to stop the program (apart from switching off) is to press STOP, followed by NEWLINE.

### 7 STOP AND CONTINUE

STOP may be pressed at any time. The computer will halt and a message

STOPPED

or

STOPPED AT NNNN: P

will be displayed, where NNNN is the line number which is about to be executed. Execution of a program may be restarted by the command CONTINUE, or the abbreviated command CONT . From the example above

```
YOUR NAME? FRED  
HELLO FRED  
YOUR NAME? (STOP, NEWLINE  
pressed)
```

STOPPED AT 20

■

Now CONTINUE the program

CONT

```
HELLO  
YOUR NAME? —  
etc.
```

Note that the program continued, on line 20, by printing "HELLO" only, since the input statement prior to the STOP, was not followed by an input string (or, the input string was the NULL STRING).

STOP may be used in a program, particularly during testing and fault finding, to

## CHAPTER 3 – SIMPLE BASIC

### 7 STOP AND CONTINUE

halt a program, so that variables may be examined. Program execution may then be continued. Line 30, above, may be edited, to produce the following:-

```
10 INPUT "YOUR NAME? ") A$  
20 PRINT TAB (7); "HELLO "; A$  
30 STOP  
40 PRINT "THIS LINE WILL  
NEVER BE PRINTED"  
50 END
```

This will produce the output

```
RUN  
YOUR NAME? BABBAGE  
HELLO BABBAGE  
  
STOPPED AT 30
```

The only variable available in this example is A\$

```
? A$  
BABBAGE  
■
```

after which CONT produces the output  
THIS LINE WILL NEVER BE PRINTED.

### 8 REM

When a BASIC program consists of more than a few lines, it can be difficult to see at once what it does. The keyword REM is used to introduce a REMark, i.e. a line in the program which will be ignored by BASIC. As an example, see lines 190 and 195 in the program on page 49 It is good practice to use REM:

- a) at the beginning of a program listing, to explain the purpose of a program,
- b) at the beginning of any sub-routine (see Chapter 4) to explain the purpose of the sub-routine.
- c) anywhere where clarification (especially at some future date) will be helpful.



## **CHAPTER 4**

### **CONTROL**

With the exception of GOTO, the commands in the last chapter allowed programs to execute line by line, in numerical order. The CONTROL commands allow the order of execution to be determined either by the programmer during program development, or by the program itself.

1. FOR .... NEXT
2. IF .... THEN
3. GOSUB
4. ON .... GOTO and ON .... GOSUB
5. ON ERROR and REPORT
6. ON BREAK

## CHAPTER 4 – CONTROL

### 1 FOR – NEXT

Thus

```
10 FOR I = 1 TO 8
20 FOR J = 1 TO 10
30 A (I, J) = 10 * I * J
40 NEXT J
50 NEXT I
60 END
```

The inner loop will be executed  $10 * 8 = 80$  times. The loops may not overlap, thus

```
10 FOR A = 1 TO 6
20 T = A * 4
30 FOR B = 1 TO 4
40 P = T * B
50 NEXT A
60 NEXT B
```

is NOT allowed.

The result of the correct example above may be seen by running it and then executing the program below –

```
100 FOR I = 1 TO 8
110 FOR J = 1 TO 10
120 PRINT A (I, J);
130 NEXT J
140 NEXT I
150 END
```

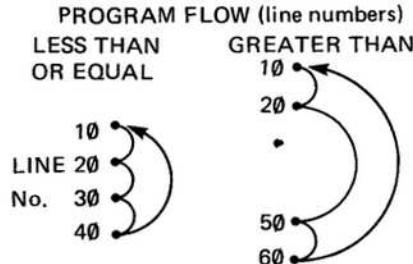
### 2 IF – THEN

The IF-THEN statement causes execution of alternative parts of a program depending on the CONDITION within the IF-THEN statement. All the relational operations ( $<$ ,  $>$ ,  $\leq$ ,  $\geq$ ,  $=$ ,  $\neq$ ) and the logical operations (AND, OR, NOT) may be used in the CONDITION.

20 IF C = 4 \* A THEN 100 will go to line 100 only if the condition is met (i.e.  $C = 4 * A$ ). The program

```
10 INPUT N
20 IF N >= 5 THEN 50
30 PRINT "LESS THAN OR = 5"
40 GOTO 10
50 PRINT "GREATER THAN 5"
60 GOTO 10
70 END
```

will print a message LESS THAN OR = 5 or GREATER THAN 5 depending on the value of the number keyed in, and then return to request another number. At line 20, if the number is greater than 5, execution continues at line 50, otherwise it proceeds to the next line.



## CHAPTER 4 – CONTROL

### 2 IF – THEN

The condition must be a logical expression (see page 29) and may be as complex as required:

```
70 IF (A * A + B * B - C * C) <
    0.02 THEN 200
or
200 INPUT ("ANOTHER GAME?
    YES OR NO . . .") A$
210 IF A$ = "YES" THEN 10
220 PRINT "GOODBYE"
230 END
```

Where IF statements are used to re-direct program execution, GOTO may be used in place of THEN,

```
IF X < 1 THEN 100
```

is the same as

```
IF X < 1 GOTO 100
```

(IF X < 1 THEN GOTO 100 is also acceptable.)

IF statements may also be followed by any keyword. The example above may be re-written.

```
10 INPUT N
20 IF N > 5 PRINT "GREATER
    THAN 5": GOTO 10
30 PRINT "LESS THAN 5"
40 GOTO 10
```

### 2 IF – THEN

When the statement following the condition is a LET, then either LET or THEN must be used, e.g.

IF A = 1 B = 2 is NOT allowed  
but  
IF A = 1 THEN B = 2 and  
IF A = 1 LET B = 2 are allowed

## CHAPTER 4 – CONTROL

### 3 GOSUB

In many computer applications there are routine tasks which are required to be performed intermittently during the execution of a program. In games programs, for example, there is frequently a part of the program used to re-evaluate the players score after each move is made. In the business environment for example, in stock control, a routine will be used to re-calculate the stock of items held after each transaction is completed. In each case the sequence of instructions required does not appear several times in the program listing, but only once, as a SUB-ROUTINE. When required, control is passed to the sub-routine by a GOSUB statement. The sub-routine executes its operations, and returns control to the body of the program by a RETURN statement. Control will return to the program line following the GOSUB. In the program symbolically represented below

```
10 LINE 1
20 LINE 2
30 GOSUB 1000
40 LINE 3
.....
1000 SUBROUTINE LINE 1
1010 RETURN
```

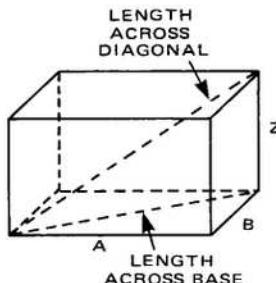
the computer will execute the line numbers in the order

```
10 20 30 1000 1010 40 ...
```

As an example, here is a program which

### 3 GOSUB

calculates the length across the base of a rectangular box, and the length across the diagonal.



The program uses a small subroutine which uses Pythagoras's theorem (the square of the hypotenuse equals the sum of the squares of the other two sides) to evaluate the unknown dimension. The sub-routine first calculates the length across the base, using A and B as input, returning the length as C, then uses the length across the base and the depth as input, returning the diagonal length.

The sub-routine is

```
200 C = SQR (A * A + B * B)
210 PRINT
220 RETURN
```

SQR in line 200 is equivalent to SQURE Root. The PRINT statement in line 210 is included in the sub-routine to improve legibility in the complete program. The RETURN statement, which must be the

## CHAPTER 4 – CONTROL

### 3 GOSUB

last statement in any sub-routine, may be abbreviated RET if desired.

The complete program follows

```
10 INPUT ("LENGTH, BREADTH,  
HEIGHT?") A, B, Z  
20 IF A = 0 OR B = 0 THEN 999  
30 GOSUB 200  
40 PRINT "LENGTH ACROSS  
BASE ="; C  
50 A = C: B = Z  
60 GOSUB 200  
70 PRINT "LENGTH ACROSS  
DIAGONAL ="; C  
80 PRINT PRINT: GOTO 10  
190 REM: SUB-ROUTINE TO  
CALCULATE  
195 REM SQUARE ROOT OF SUM  
OF SQUARES  
200 C = SQR (A * A + B * B)  
210 PRINT  
220 RETURN  
999 END
```

- Line 10 Requests the dimensions of the box, with a suitable prompt.
- Line 20 Allows the user to escape from the program by input of a dimension of zero.
- Line 30 Passes control to the sub-routine.
- Line 40 On RETURN from the sub-routine prints the base length.
- Line 50 Re-assigns the variables to comply with the requirements of the sub-routine (it expects A and B as inputs, and outputs C).
- Line 60 Calls the sub-routine for the

### 3 GOSUB

second time.

Line 70 On return, prints the diagonal length.

Line 80 Inserts blank lines to make the presentation tidier and returns control to the start of the program.

Note that the sub-routine is placed at the end of the program, but could in fact be placed anywhere.

RUN

LENGTH, BREADTH, HEIGHT? 3, 4, 12

DISTANCE ACROSS BASE = 5

DISTANCE ACROSS DIAGONAL = 13

.....

Subroutines may be nested, that is one sub-routine may call another.

## CHAPTER 4 – CONTROL

### 4 ON – GOTO and ON – GOSUB

Program control may be transferred to different lines, as a function of the value of an expression, using the statement

ON EXPRESSION GOTO LINE No.  
LINE No., etc.

NewBrain BASIC evaluates the expression, rounds it to the nearest integer, then goes to the line with the first number if the answer was one, the second number if the answer was two, and so on. In this example, because A = 2.7, which is rounded to 3, the programs jumps to line 300, and prints 300.

```
10  A = 2.7
20  ON A GOTO 100, 200, 300,
     400, 500
30  PRINT "SHOULDN'T GET
     HERE": GOTO 999
100 PRINT "100": GOTO 999
200 PRINT "200": GOTO 999
300 PRINT "300": GOTO 999
400 PRINT "400": GOTO 999
500 PRINT "500"
999 END
```

Note that if the decimal component of the value of the expression is greater than or equal to .5000000000 approximately, the number is rounded up, otherwise the number is rounded down. Because BASIC does arithmetic using Binary numbers, and then converts the results to decimal numbers, the rounding is not exact. In the example, numbers greater than about 2.4999999980 will be

### 4 ON – GOTO and ON – GOSUB

rounded to 3, otherwise they will be rounded down. Additionally, care should be taken to ensure that the value of the expression is neither zero nor greater than the number of options (that is greater than 5 in the example above) otherwise an error message results. The program may be modified, to test differing numbers by altering lines 10 and 999:

```
10  INPUT ("NO.      ") N
20  ON N GOTO 100, 200, 300
     400, 500
30  PRINT "SHOULDN'T GET
     HERE": GOTO 999
100 PRINT "100": GOTO 999
200 PRINT "200": GOTO 999
300 PRINT "300": GOTO 999
400 PRINT "400": GOTO 999
500 PRINT "500"
999 GOTO 10
```

Following an ON ... GOTO, execution will always continue at one of the line numbers on the statement, or else an error message will be generated. Thus, in this example, line 30 will never be executed. Where it is desired to switch execution to different points in a program, and then continue on a common path, ON ... GOSUB is used.

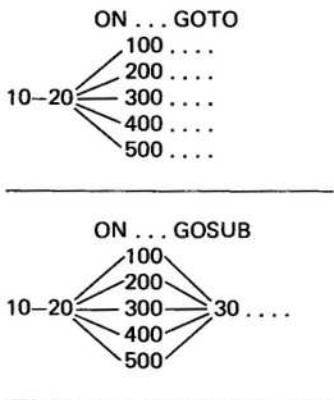
ON GOSUB is very similar to ON GOTO except that a sub-routine is executed following the ON statement, and returns to the statement following the ON statement. The example from ON-GOTO may use GOSUB, as below:

## CHAPTER 4 – CONTROL

### 4 ON – GOTO and ON – GOSUB

```
10 INPUT ("NO.      ") N
20 ON N GOSUB 100, 200, 300,
    400, 500
30 PRINT "GOT HERE FROM
        SUBROUTINE": GOTO 10
100 PRINT "100": RETURN
200 PRINT "200": RETURN
300 PRINT "300": RETURN
400 PRINT "400": RETURN
500 PRINT "500": RETURN
999 END
```

The flow of the programs may be pictured thus



LINE 30, which was never executed in ON . . . GOTO, is always executed in ON . . . GOSUB.

### 5 ON ERROR AND REPORT

During program development errors are trapped by the NewBrain operating system and error messages displayed. It is possible to redirect the error handling routines to within the user program, using ON ERROR

```
10 ON ERROR GOTO 100
20 INPUT ("NO. ?") A
30 PRINT "RECIPROCAL OF ";
        A; "IS"; 1/A
40 INPUT ("ANOTHER NO. ?") B$
50 IF B$ = "Y" GOTO 20
60 END
100 PRINT "not valid"
110 PRINT "TRY AGAIN"
120 GOTO 20
```

In this program when an attempt is made to calculate the reciprocal of zero, input in response to line 20, instead of outputting an error message the program jumps to line 100, resulting in the output

RECIPROCAL OF 0 IS NOT VALID  
TRY AGAIN

If line 120 is replaced by

```
120 END
```

then the user may identify the error in the normal fashion, by the command REPORT which prints the latest error message and ends execution.

## CHAPTER 4 – CONTROL

### 5 ON ERROR

```
REPORT  
ERROR 2 AT 30
```

■

(Note that if an error has not occurred, it is an error to call REPORT.)

After an ON ERROR transfer execution may be continued at the line at which the error occurred by RESUME, or at an alternative line by RESUME line number. Two system functions are available for use by an error handling routine, ERRLIN and ERRNO.

ERRLIN returns the line number in which the error occurred, ERRNO returns the error number.

ON ERROR GOTO 0 cancels the trapping of errors.

### 6 ON BREAK

The STOP key is used to break into a program and halt execution. The ON BREAK statements allows the STOP key to redirect the program in the same manner as ON ERROR. The example above may be re-written:-

```
10  ON ERROR GOTO 100  
15  ON BREAK GOTO 200  
20  INPUT ("NO.?") A  
30  PRINT "RECIPROCAL OF";  
    A;"IS"; 1/A  
60  GOTO 20  
100 PRINT " NOT VALID"  
110 PRINT "TRY AGAIN"  
120 GOTO 20  
200 PRINT "TERMINATED"  
210 PRINT "GOODBYE"  
220 END
```

To stop the program the STOP key is pressed, and the message

```
TERMINATED  
GOODBYE
```

■

appears.

The ERRLIN and ERRNO functions may be used with ON BREAK; ERRNO returns the system interrupt number 0 (for the STOP key) ERRLIN, RESUME and REPORT act as in ON ERROR. ON BREAK GOTO 0 cancels the trapping of the STOP key.

## **CHAPTER 5**

### **DATA STRUCTURES**

**Data within a program may be stored either item by item, in DATA statements or in ordered arrays. This chapter discusses the creation and manipulation of arrays, and the use of data statements.**

- 1. ARRAYS**
- 2. DIMENSION (DIM and CLEAR)**
- 3. OPTION BASE**
- 4. DATA, READ and RESTORE**

## CHAPTER 5 – DATA STRUCTURES

### 1 ARRAYS

An array is a means of storing items of data, under a common name, with each item identified by a SUBSCRIPT to the array name. If the array was called A then the ELEMENTS of the array are called A(1), A(2), A(3) etc., where 1, 2, 3 ... are the subscripts. An array of this sort may be thought of as a list.

A(1)	2.2
A(2)	6.8
A(3)	7.1
A(4)	9.3
.	
.	
.	
A(10)	17.65

where the number of elements, in this case 10, is the DIMENSION of the array. Thus A is said to be a ONE-DIMENSIONAL array, of dimension 10. Arrays may also be TWO-DIMENSIONAL, corresponding to a table,

C =	1	2	3
T(1,C)	2.2		
T(2,C)		17.4	-0.15
T(3,C)	6.3		
T(4,C)			

### 1 ARRAYS

where

T(1, 1) =	2.2
T(3, 1) =	6.3
T(2, 2) =	17.4
T(2, 3) =	-0.15
etc.	

Here T is a two-dimensional array, of dimensions 4 and 3, i.e. 4 rows and 3 columns. Each element of the array may be manipulated in the same way as any numeric variable, thus

```
10 LET R = 2
20 LET A (2, 1) = PI * R * R
30 PRINT A (2, 1)
40 END
```

(RESULT: 12.566371)

or

```
10 LET R = 2
20 LET A (2, 1) = PI * R * R
30 LET A (2, 2) = PI
40 LET A (3, 3) = A (2, 1)/A (2, 2)
50 PRINT A (3, 3)
60 END
```

(RESULT: 4)

The arrays above are NUMERIC ARRAYS, that is each element contains a number. STRING ARRAYS are similar, except that each element contains a string. String arrays, like string variables, are identified by the \$ suffix in the array name. Thus

## CHAPTER 5 – DATA STRUCTURES

### 1 ARRAYS

```
10 A$(1) = "THIS "
20 A$(2) = "IS A "
30 A$(3) = "STRING "
40 A$(4) = "ARRAY "
50 FOR C = 1 TO 4: PRINT
    A$(C); :NEXT C
60 END
```

produces

THIS IS A STRING ARRAY

A variable may have the same name as an array in a program, thus A\$ and A\$( ), and A and A( ), are distinguished by BASIC. The following has a variable A, a string A\$, and a string ARRAY A\$( ).

```
10 A$(1) = "THIS "
20 A$(2) = "IS A "
30 A$(3) = "STRING "
40 A$(4) = "ARRAY "
50 FOR A = 1 TO 4: PRINT
    A$(A); :NEXT A
60 PRINT
70 A$ = "THIS IS A STRING"
80 PRINT A$
90 END
```

The following program will fill a two dimensional array with numbers which represent the array subscripts, that is A(2, 1) = 21, A(4, 5) = 45 etc., and then print out the array. The dimensions of the array are requested from the user, and must not exceed 9 rows by 7 columns, in order to retain the format of the display.

### 1 ARRAYS

```
10 INPUT ("ROWS, COLUMNS ?")
20 R, C
30 FOR I = 1 TO R
40 FOR J = 1 TO C
50 A(I, J) = 10 * I + J
60 NEXT J
70 NEXT I
70 PRINT TAB (7); "C = ";
80 FOR I = 1 TO C: PRINT
    I; " "; :NEXT I
90 PRINT: PRINT
100 FOR I = 1 TO R
105 ?"A ("; I; ", C) = ";
110 FOR J = 1 TO C
120 PRINT A(I, J);
130 NEXT J
135 PRINT
140 NEXT I
150 END
```

The output looks like this:-

ROWS, COLUMNS? 6, 5

C =	1	2	3	4	5
A( 1 , C)=	11	12	13	14	15
A( 2 , C)=	21	22	23	24	25
A( 3 , C)=	31	32	33	34	35
A( 4 , C)=	41	42	43	44	45
A( 5 , C)=	51	52	53	54	55
A( 6 , C)=	61	62	63	64	65

Arrays may not have more than two dimensions, and the number of elements in an array is limited by the amount of memory space available, up to a maximum of 5374 elements. The dimension(s) of arrays which are greater than 10 or 10 by 10 elements should be declared by a DIMENSION STATEMENT.

## CHAPTER 5 – DATA STRUCTURES

### 2 DIM and CLEAR

DIM, short for dimension, is an instruction to the computer to reserve sufficient memory space to store the contents of the arrays in the DIM statement. Several arrays may be dimensioned by one DIM statement,

**DIM A (20), C (6, 17), CS (12)**

DIM statements may contain algebraic expressions, which are evaluated, and rounded to the nearest integer number:-

```
10 N = 6.6
20 DIM A (N+10)
```

results in A having dimension 17.

DIM statements should be placed early in a program, before the first use of any array elements, or SUBSCRIPTED VARIABLES (A(3), B(17,2), C(4,1) etc). If any element of an array is encountered before the relevant DIM statement, the array is assumed to have dimension 10, or 10 by 10. Once an array has been dimensioned by a DIM statement or by default, it may not be re-dimensioned, unless it has been CLEARED. RUN automatically CLEARS.

CLEAR reverses the effect of a DIM statement, in that it releases the memory used for array variables, and it additionally resets all variables, thus subsequent references to variables will produce the value 0 for numbers, and null for strings.

### 2 DIM and CLEAR

Thus the program

```
10 C (4, 5) = 45: F = 22
20 D$ = "ANSWER"
30 GOSUB 100
40 CLEAR
50 GOSUB 100
60 END
100 PRINT C (4, 5), F, D$
110 RET
```

produces the results

45	22	ANSWER
0	0	

Specific variables, may be cleared, thus

**CLEAR F** (clear a variable)

or

**CLEAR C( )** (clear an array)

or a list may be cleared

**CLEAR C( ), D\$, A, L\$( )**

## CHAPTER 5 – DATA STRUCTURES

### 3 OPTION BASE

The first element of an array A may be considered as A(0) or A(1), depending on the programmer, or the version of BASIC in use. If A(0) is permitted then an array of dimension 8, A(8), has 9 elements A(0) to A(8). NewBrain BASIC permits the programmer to select which alternative he wishes by the statement

**OPTION BASE 0**

or

**OPTION BASE 1**

If the base is 0 the first element of a one-dimensional array will be A(0), the first element of a two-dimensional array will be B(0,0). If the base is 1, then the first elements are A(1) and B(1, 1). OPTION BASE may only be followed by 0 or 1, and must be declared before any dimension statement, or use of arrays. If OPTION BASE is omitted, base zero is assumed.

### 4 DATA, READ and RESTORE

A DATA statement contains numeric or string constants which are to be used in a program, and which are assigned to variables by a READ statement. The constants in a DATA statement are read, one at a time, and assigned to the variables in the list following the READ statement.

Thus

**10 DATA 12, 2.7, HELLO, 5, SUM**

may be read by

**100 READ A, C, E\$, L, M\$**

assigning 12 to A, 2.7 to C, HELLO to E\$ etc. Numeric and string constants may be mixed in both DATA and READ statements, but they must match, or an error will result.

**10 DATA 3.4, RADIUS**

**20 READ C\$, Y**

will assign 3.4 to C\$, but will not assign RADIUS to Y, and will produce an error message.

Data may be placed in as many DATA statements as desired and placed anywhere within a program. A DATA statement, if part of a multi-statement line, must be the last statement of the line. BASIC assembles all the data items in the statements into a list, and maintains a pointer into the list, advancing the pointer each time an item is READ.

## CHAPTER 5 – DATA STRUCTURES

### 4 DATA, READ and RESTORE

```
10 DATA 23.5, 45.2, 54.4.  
      545, 3.28  
20 DATA 2.2, 1.76, 1.0819, 454  
100 READ A, F, I, G  
200 READ Y, K, L, M, D, X
```

After execution of line 100, A = 23.5, F = 45, I = 2.54, G = 4.545 and the pointer will indicate 3.28 as the next data item to be read, as Y in line 200. There are more variables in the READ statements than there are data items in the DATA statements, however, so X cannot be assigned a value in this instance, and an error message will result.

The pointer in the data list can be reset to either the beginning of the complete list, or to the first item in any data statement by RESTORE.

RESTORE sets the pointer to the start of the list  
RESTORE 20 sets the pointer to the first item in line 20.

If line 20 were not a DATA statement, then the pointer would be set to the first item in a DATA statement following line 20.

```
10 DATA 1, 2, 3  
20 DATA 4, 5, 6  
30 DATA 7, 8, 9  
100 READ A, B, C, D  
101 REM SETS A=1, B=2, C=3, D=4  
200 RESTORE 20  
201 REM SETS POINTER TO 4
```

### 4 DATA, READ and RESTORE

```
210 READ E, F, G  
211 REM SETS E=4, F=5, G=6  
300 RESTORE  
301 REM SETS POINTER TO 1  
310 READ H, I, J  
311 REM SETS H=1, I=2, J=3  
400 PRINT A, B, C, D, E, F, G, H, J  
999 END
```

RUN	1	2	3	4
	4	5	6	1
	2	3		

The program above would set A = 1, B = 2, C = 3, . . . I = 9, and fail to assign J, were it not for the RESTORE statements.

This example

```
10 DIM A$(20)  
20 DATA NEWBRAIN  
30 FOR I=1 to 20; READ A$(I)  
40 RESTORE : NEXT
```

sets each element of the string array to "NEWBRAIN". In the absence of the RESTORE statement an error would occur, as the data would be used up after the first READ.

## **CHAPTER 6**

### **FURTHER INPUT AND OUTPUT**

For many applications, the PRINT and INPUT commands described in Chapter 2 will prove sufficient. More advanced input and output handling is available, either by selecting a stream other than the console, or by using the commands described in this chapter and in Chapter 10.

- 1. OPEN and CLOSE**
- 2. STREAM NUMBERS**
- 3. LINPUT**
- 4. PUT and GET**
- 5. SAVE, VERIFY, LOAD and LIST**

## CHAPTER 6 – FURTHER INPUT AND OUTPUT

### 1 OPEN and CLOSE

A BASIC program may use up to 255 numbered STREAMS in addition to the console. A STREAM is a data route from the computer to an input/output device. Each of these may be assigned to any one of the device types listed in Appendix 7. The OPEN statement is used to make this assignment, e.g.

```
OPEN # 2, 3  
OPEN OUT # 7, 2, "accounts"
```

In the first example stream number 2 is opened with device type 3, i.e. the line (or v.f.) display. Input and output commands which make use of stream 2 are then allowable, e.g.

```
PRINT # 2, "press any key";
```

In the second example, stream number 7 is opened with device type 2, i.e. the cassette recorder plugged into the TAPE 2 socket. Output commands which make use of stream 7 are then allowable.

After the OPEN command has been executed, all input or output operations which use the same stream number are directed to the selected peripheral. This connection may be broken by means of the CLOSE command:

```
CLOSE # 7
```

Since the CLOSE command does not give an error, even when the given stream is not in fact open, it is easy to close all

### 1 OPEN and CLOSE

streams with a command of the form

```
FOR I = 1 to 255: CLOSE #I : NEXT I
```

Note that stream 0 may not be explicitly closed. If stream 0 is opened at any time, BASIC first closes the console stream then re-opens it. This is to ensure that there is always a console stream to handle input commands!

## CHAPTER 6 – FURTHER INPUT AND OUTPUT

### 2 STREAM NUMBERS

All the input and output statements are acceptable with or without a stream number. If a stream number is given, the form of the command is as in

**INPUT # 5, e\$**

i.e. the BASIC keyword is followed by "#", the stream number, and the remaining parameters with a comma "," after the stream number if it does not end the statement. The default system if a stream number is not used is 0, i.e. the console stream, except in the case of the PLOT command described in Chapter 10.

The use of a prompt with the INPUT and LINPUT statements is not allowed when a stream number is given after the keyword.

### 3 LINPUT

The LINPUT command is very similar to INPUT when the latter is used to supply a value to a string variable. All input from the selected device is collected until a NEWLINE is received. This means in particular that any quotation marks are placed in the string variable exactly as typed, thus

```
LINPUT D$ : PRINT D$  
?She said "Help"  
She said "Help"
```

When a program is written to use the line display as the console, LINPUT is often used with a prompt of less than 16 characters, e.g.

```
LINPUT ("Enter part no:") p$
```

Note that if the prompt were given as "Enter part number:", the display would actually show only the last 15 characters, as the window is moved far enough along the current line to bring the cursor position onto the right hand end of the window:-

ter part number:\_\_

## CHAPTER 6 – FURTHER INPUT AND OUTPUT

### 4 PUT and GET

In many control applications it is necessary to output single bytes to a stream. This is done using the PUT statement. One or more bytes may be given in the PUT-list, and a stream number may be specified:

```
PUT 22, 8, 10  
PUT # 8, 13
```

In the first example, the console stream is directed to place the cursor at character position 8 on row 10 of the screen (see Appendix 3). In the second, a NEWLINE control code is delivered to stream number 8. If stream 8 is a printer stream, this will result in a single paper feed.

The PUT-list consists of one or more bytes, i.e. numbers in the range 0–255. Each of these may be a constant, an expression, or a string; however, if a string is used, only the first character is considered. If PUT is used to output a control code, the effect of that code depends on the peripheral device used; for the Screen Editor control codes, see Appendix 3. Certain control codes require additional bytes to further define the action to be taken. If the byte is not a control code, it is equivalent to its corresponding character, as shown in Appendix 5. Thus the following two examples produce the same result:

```
PRINT "Hello";  
PUT 72, 101, 108, 108, 111
```

Note the Screen Editor control code 27

### 4 PUT and GET

(ESCAPE) has the effect of ensuring that the next byte is not treated as a control code. Thus PUT 27, 12 prints character number 12, whereas PUT 12 moves the cursor to the home position.

The command GET may be used to obtain a single byte from an input stream. This can be useful in various control applications, and when a stream is of type 7 (Z80 port) or type 9 (modem):

```
OPEN # 1, 9  
GET # 1, X : PUT X
```

Many real-time games will operate by opening a keyboard input stream (device type 5 or 6) and using GET to pick up single keystrokes.

## **CHAPTER 6 – FURTHER INPUT AND OUTPUT**

### **5 SAVE VERIFY LOAD and LIST**

The BASIC program entered by the user is held in memory in "entokened" form. This means that many of the BASIC reserved words are replaced by a one-byte TOKEN in order to save space. The SAVE and LOAD commands allow the program to be output to any stream (usually a backup store) or read in, in this "entokened" form. The LIST command allows the ordinary, or expanded, form to be output to any stream, usually the screen or a printer. Examples of all these commands are given in Chapter 1.



## CHAPTER 7

### INTRINSIC FUNCTIONS

NewBrain BASIC provides a range of predefined functions. This chapter discusses the mathematical, utility, and random number functions, and concludes with user-defined functions. String handling functions are treated in Chapter 8.

1. PI	PI
2. TRIGONOMETRIC FUNCTIONS	SINE (SIN) COSINE (COS) TANGENT (TAN) ARCSINE (ASN) ARC-COSINE (ACS) ARC-TANGENT (ATN)
3. LOGARITHMS	
3.1	Natural Logarithm (LOG)
3.2	Natural Anti-Logarithm (EXP)
4. POWERS	
4.1	Square Root (SQR)
4.2	Raising to a Power ( $\uparrow$ )
5. ARITHMETIC	
5.1	Integer Part (INT)
5.2	Absolute Part (ABS)
5.3	Sign (SGN)
6. RANDOM NUMBERS	RND/RANDOMIZE
7. USER DEFINED FUNCTIONS	DEF FN

## CHAPTER 7 – INTRINSIC FUNCTIONS

### 1 PI

The ratio of the circumferences of a circle to its diameter, represented by the Greek letter  $\pi$  (pronounced PI), and often approximated by 22/7, is provided by NewBrain BASIC to an accuracy of 10 significant figures.

```
PRINT PI [1.9]
```

```
3.141592654
```

### 2 TRIGONOMETRIC FUNCTIONS

NewBrain BASIC provides

SIN(X) — The SINE of (X)  
COS(X) — The COSINE of (X)  
TAN(X) — The TANGENT of (X)  
ASN(X) — The angle whose sine is X  
or ARCSINE (X)  
ACS(X) — The angle whose cosine is  
X or ARC-COSINE (X)  
ATN(X) — The angle whose tangent is  
X or ARC-TANGENT (X)

The angles used with trigonometric functions are always expressed in RADIANS.

There are  $\pi$  radians in 180 degrees, thus N degrees = N \*  $\pi/180$  RADIANS, i.e.  $90^\circ = \pi/2$ ,  $60^\circ = \pi/3$  etc. If angles greater than 360 degrees,  $2\pi$  radians, are used as the argument to trigonometric functions, they are reduced modulo  $360^\circ$  (or  $2\pi$  radians), that is the angle is divided by 360 or  $2\pi$ , and the remainder used as the argument. Thus

$$\text{Sin } 450^\circ = \text{Sine } 90^\circ$$

or in BASIC format

$$\text{SIN } (5 * \text{PI}/2) = \text{SIN } (\text{PI}/2)$$

Trigonometric functions may be manipulated in the same way as variables. The following program plots a simple graph of the sine and cosine functions.

```
10 GOSUB 200
20 FOR D = 0 TO 360 STEP 20
```

## CHAPTER 7 – INTRINSIC FUNCTIONS

### 2 TRIGONOMETRIC FUNCTIONS

```
30 R = D * PI/180
40 IF SIN (R) > COS (R) THEN 100
50 PRINT D; TAB (22 + 15 * SIN
(R)); "S";
60 PRINT TAB (22 + 15 * COS
(R)); "C"
70 GOTO 120
100 PRINT D; TAB (22 + 15 * COS
(R)); "C";
110 PRINT TAB (22 + 15 * SIN
(R)); "S"
120 NEXT D
130 GOSUB 200
140 END
200 PRINT "DEG"; TAB (6); "-1";
210 PRINT TAB (13); "-.5";
220 PRINT TAB (22); "0";
230 PRINT TAB (29); "+.5";
240 PRINT TAB (37); "+1"
250 RETURN
```

The sine curve is plotted using S, the cosine curve using C.

### 3 LOGARITHMS

#### 3.1 Log

LOG (X) provides the natural logarithm (to the base e, e = 2.71828183) of the argument X thus

```
? LOG (10)
2.3025851
? LOG (2.71828183)
1
? LOG (1)
0
```

LOG will not accept an argument which is zero, or negative

#### 3.2 EXP

EXP (X) produces the value of "e to the power X" thus EXP(2) = e \* e, EXP(3) = e \* e \* e etc. EXP (X) is the inverse function of LOG, i.e. the natural antilogarithm, thus

```
PRINT EXP (LOG(X))
```

produces the result X as does

```
PRINT LOG (EXP(X))
```

Reversing the examples in LOG above

```
? EXP(0)
1
? EXP(1)
2.7182818
? EXP(2.30258509)
10
```

## CHAPTER 7 – INTRINSIC FUNCTIONS

### 4 POWERS

#### 4.1 SQR

SQR (X) yields the SQuare Root of (X).

```
? SQR(25)  
5  
? SQR(62500)  
250
```

The square root of a negative number is not a real number, and produces an error message.

#### 4.2 ↑

↑ is read as "TO THE POWER OF" thus  $2 \uparrow 3$  is read as "TWO TO THE POWER OF THREE" which equals  $2 * 2 * 2$  or 8. This short program illustrates the function of ↑.

```
10 INPUT ("NUMBER ? ") N  
20 A$ = "TO THE POWER OF"  
30 FOR I = 1 TO 10  
40 PRINT N; A$; I; "="; N ↑ I  
50 NEXT I  
60 PRINT: GOTO 10
```

producing a table of powers of the input number, up to a power of 10, thus

```
NUMBER ? 2  
2 TO THE POWER OF 1 = 2  
2 TO THE POWER OF 2 = 4  
2 TO THE POWER OF 3 = 8  
2 TO THE POWER OF 4 = 16  
2 TO THE POWER OF 5 = 32  
2 TO THE POWER OF 6 = 64  
2 TO THE POWER OF 7 = 128  
2 TO THE POWER OF 8 = 256
```

### 4 POWERS

```
2 TO THE POWER OF 9 = 512  
2 TO THE POWER OF 10 = 1024  
NUMBER ? - 1 of 10 under 9  
1 of 1024 under  
1 of 512
```

where each line is N (in this case 2) times the previous line. The program may be modified to work up to any power, by inserting line 15 and modifying line 30:

```
10 INPUT ("NUMBER ? ") N  
15 INPUT ("GREATEST POWER  
? ") P  
20 A$ = "TO THE POWER OF"  
30 FOR I = 1 TO P  
40 PRINT N; A$; I; "="; N ↑ I  
50 NEXT I  
60 PRINT: GOTO 10
```

↑ may be used to evaluate roots of a number by using the form

PRINT N ↑ (1/R)

where R is the root required, i.e. 2 for square root, 3 for cube root, etc. The following program will evaluate the roots of a number N, up to the Rth root.

```
10 A$ = "ROOT OF"  
20 INPUT ("NUMBER ? ") N  
30 INPUT ("GREATEST ROOT ?  
") R  
40 FOR I = 2 TO R  
50 IF I > 3 PRINT " "; I;  
"TH ";: GOTO 70  
60 ON I GOSUB 100, 110, 120  
70 PRINT A$; N; "="; N ↑ (1/I)  
80 NEXT I
```

## CHAPTER 7 – INTRINSIC FUNCTIONS

### 4 POWERS

```
100 PRINT: GOTO 10
110 PRINT "SQUARE "; :RET
120 PRINT "CUBE"; :RET
```

LINES 50 and 60 are used to preserve the standard nomenclature, i.e. "SQUARE ROOT", "CUBE ROOT" instead of 2nd Root, 3rd Root. The variable I in LINE 60 should only take the values 2, and 3, switching to lines 110 and 120 respectively, therefore an arbitrary line number is placed in the list of arguments corresponding to the switch value 1. Note that the line corresponding to the number must exist, however, otherwise an error will be flagged. For "NUMBER" = 64, and "GREATEST ROOT" = 7, the output is

```
SQUARE ROOT OF 64 = 8
CUBE ROOT OF 64 = 4
4TH ROOT OF 64 = 2.8284271
5TH ROOT OF 64 = 2.2973967
6TH ROOT OF 64 = 2
7TH ROOT OF 64 = 1.8114473
```

### 5 ARITHMETIC

#### 5.1 INT

INT(X) returns the integral part of X, that is the largest integer less than or equal to X. Thus

INT (35.3) = 35
INT (2.1) = 2
INT (-2.3) = -3
INT (-7.1) = -8
INT (1.345E23) = 1.345 E +23
INT (1.234567E5) = 123456

#### 5.2 ABS

ABS(X) strips X of its minus sign, if one exists, and returns the positive value of X.

ABS(42) = 42
ABS(-42) = 42

Formally

ABS(X) = X if $X \geq 0$
= -X if $X < 0$

remember that minus a negative number X (= -4 say) produces a positive number (4).

ABS(X) is used, for example, where the sign of a number is immaterial, or where it is essential to have a positive number, for example prior to extracting a square root:-

```
10 INPUT D
20 Y = SQR (ABS(D))
```

## CHAPTER 7 – INTRINSIC FUNCTIONS

### 5 ARITHMETIC

If it is necessary to strip off the sign, by ABS( ), and perhaps restore it later, then SGN( ) may be used.

#### 5.3 SGN

SGN(X) returns the value

+1 if  $X > 0$  i.e. X positive  
0 if  $X = 0$   
-1 if  $X < 0$  i.e. X negative

PRINT SGN(5.3), SGN(0), SGN(-3.2)  
1            0            -1

In this example, a number is input, stripped of its sign, operated upon, and then has its sign restored.

```
10 INPUT N
20 S = SGN (N)
30 N = ABS (N)
40 ...
50 ...
60 N = S * N
```

As a practical example, using INT, ABS, and SGN the following program simulates a radar "speed trap" which operates by transmitting radio energy at a known frequency say F, which is changed in frequency when it bounces off an approaching, or receding car, increasing when the car is approaching, decreasing when receding. (The change in frequency is called a Doppler shift after the discoverer of the phenomenon.) This

### 5 ARITHMETIC

program asks for an input frequency, converts it to miles per hour, and states whether the "car" was approaching, receding, or stationary.

```
10 REM DOPPLER SHIFT
20 REM K = CONVERSION
   FACTOR FREQ./M.P.H.
30 REM M = SPEED IN M.P.H.
40 F0 = 10.00: REM TRANSMITTED FREQUENCY
50 INPUT ("RECEIVED FREQUENCY") F
60 DF = F - F0
70 S = SGN (DF)
80 M = INT (346.73 * ABS (DF))
90 S = S + 2
100 ON S GOSUB 200, 210, 220
110 PRINT "CAR"; SS; "AT"; M,
   "M.P.H."
120 PRINT: GOTO 50
200 SS = "RECEDING": RET
210 SS = "STATIONARY": RET
220 SS = "APPROACHING": RET
999 END
```

#### NOTES:

LINE 60 Calculates the change in frequency DF.

LINE 70 Strips off the sign of the frequency change.

LINE 80 Converts the change in frequency to M.P.H., ignoring the sign (by using ABS), and taking the integer part of the speed, i.e. the next lowest whole number of M.P.H.

## CHAPTER 7 – INTRINSIC FUNCTIONS

### 5 ARITHMETIC

LINE 90 Converts the sign S from the range  
-1 0 or +1  
to 1 2 3

LINE 100 Uses the new value of S to choose the appropriate word to describe the car's speed.

LINE 110 PRINTs the results.

LINE 120 Returns to the beginning.

### 6 RANDOM NUMBERS

RND is a function which produces a random number each time it is called. Thus

```
10 FOR I = 1 TO 10
20 PRINT RND
30 NEXT I
40 END
```

produces

```
.61393043
.95403863
.076651651
.21306613
.033194093
.59871473
.56201019
.97603686
.13175516
.51549558
```

The numbers lie between 0 and 1, and are not truly random, but PSEUDO-RANDOM in that a very long sequence of seemingly random numbers is produced, but given sufficient time and patience the sequence of numbers would repeat itself. The sequence starts at the same point, each time a program is run, unless the command RANDOMIZE is used, thus

```
5 RANDOMIZE
10 FOR I = 1 TO 10
20 PRINT RND
30 NEXT I
40 END
```

## CHAPTER 7 – INTRINSIC FUNCTIONS

### 6 RANDOM NUMBERS

The following program plots a simple histogram (bar chart) showing the distribution of the random numbers in a series. The length of the series is determined by the response to line 30, 2 generates  $10^2$  numbers, 3 generates  $10^3$  numbers, etc. The more numbers generated, the more even the distribution, but the longer the program runs (about 1 sec for 10 samples, 5 for 100, 25 for 1000, etc.).

Essentially, the program generates a random number, multiplies it by 10, and takes the integer part of the number, which it uses to point to an element of an array which is then incremented by one. Thus if the random number were

.63471925

X 10 = 6.3471925  
INT = 6

Therefore the 6th element of the array is incremented. The program thus divides the range between 0 and 1 into 10 sections (DECILES), which, if the distribution were uniform, would each contain the same number. As the sample range becomes bigger, the numbers become more uniform. Line 110 prints the decile number, followed by the number in the decile, lines 120 onwards print a simple bar chart.

```
10 CLEAR
20 RANDOMIZE
30 INPUT ("SAMPLE RANGE")
```

### 6 RANDOM NUMBERS

```
(POWER OF 10) "P
40 SR = 10 ↑ P
50 FOR I = 1 TO SR
60 R = INT (10 * RND)
70 T (R) = T (R) + 1
80 NEXT I
100 FOR R = 0 TO 9
110 PRINT R; T (R)
120 FOR L = 1 TO T(R)/
(5 * 10 ↑ (P-3))
130 PRINT "*";
140 NEXT L
150 PRINT
160 NEXT R
999 END
```

### RESULTS

SAMPLE RANGE	1	2	3
DECILE			
0	3	14	86
1	2	8	114
2	0	11	97
3	2	7	89
4	0	9	99
5	0	9	107
6	0	10	92
7	0	9	114
8	2	14	108
9	1	9	94

## CHAPTER 7 – INTRINSIC FUNCTIONS

### • 6 RANDOM NUMBERS

### 7 USER DEFINED FUNCTIONS

SAMPLE RANGE	4	5
DECILE		
1036	10127	
980	9893	
1015	10143	
961	10044	
1009	9901	
1007	9892	
957	9999	
1011	10049	
1045	9952	
979	10000	

As well as the intrinsic functions described in the preceding sections, BASIC provides a facility for the user to define his own functions, labelled for example FNB3, FNC\$. A user function is defined in a DEF statement, thus:-

```
DEF FNC(X) = PI * X * X/4
```

Here, FNC calculates the area of a circle of diameter X. Once defined, user functions are employed exactly as intrinsic functions.

```
10 DEF FNC (X) = PI * X * X/4
20 INPUT ("DIAMETER") D
30 A = FNC (D)
40 PRINT "AREA IS"; A
50 END
```

Notice that the variable X used in the definition in line 10, is not used in the remainder of the program. If the input to line 20 were 3 say, then line 30 evaluates FNC as PI\*3\*3/4 substituting the value of D wherever X appeared in the definition. X, which could be any character, is called a dummy variable, and its use does not prevent a true variable with the same name from being used elsewhere.

NewBrain BASIC permits user defined string functions.

```
10 DEF FNFS (XS) = "HELLO "
+DS+", HOW DO YOU DO?"
20 INPUT ("YOUR NAME ?")
DS
```

## CHAPTER 7 – INTRINSIC FUNCTIONS

### 7 USE DEFINED FUNCTIONS

```
30  A$ = FNFS (D$)
40  PRINT A$
50  END
```

produces the output

```
YOUR NAME ? NEWTON
HELLO NEWTON, HOW DO YOU
DO?
```

Any variable that may happen to exist with the same name as that used in a defined function, e.g. X\$ and FNFS(X\$) is not affected by the use of the defined function. A function may be defined which has no parameters thus

```
DEF FNR = PI/180
DEF FNH = SQR (A * A + B * B)
```

where the functions use the variables in the program. A defined function may not have more than one parameter.

A DEF FN statement may appear at any point in a program, it does not have to appear before the function is used. Each time the computer encounters a FN it will search the remainder of the program for the appropriate DEF FN statement. The commands CLEAR and RUN delete all records of defined functions, but commands of the form CLEAR FNA are not permitted.

## **CHAPTER 8**

### **STRING HANDLING**

NewBrain BASIC provides very powerful string handling functions, described in this chapter.

- 1. CONCATENATION**
- 2. LEN**
- 3. LEFT\$**
- 4. MID\$**
- 5. RIGHTS**
- 6. INSTR.**

## CHAPTER 8 – STRING HANDLING

### 1 CONCATENATION

Strings are concatenated, or joined, by either + or &. Thus

```
10 INPUT ("YOUR NAME ? ")  
    NS  
20 AS = "HAPPY BIRTHDAY"  
30 BS = AS +" TO YOU"  
40 CS = " DEAR "+ NS  
50 PRINT BS  
60 PRINT BS  
70 PRINT AS + CS  
80 PRINT BS  
90 END
```

```
RUN  
YOUR NAME? PASCAL  
HAPPY BIRTHDAY TO YOU  
HAPPY BIRTHDAY TO YOU  
HAPPY BIRTHDAY DEAR PASCAL  
HAPPY BIRTHDAY TO YOU
```

### 2 LEN

LEN (AS) returns the length of the string AS. Thus, from the above

```
PRINT LEN(AS); LEN(BS); LEN(CS)  
14 21 12
```

The argument to LEN may be a string variable as above, or a string constant.

```
PRINT LEN ("THIS WAS THEIR  
FINEST HOUR")  
26
```

## CHAPTER 8 – STRING HANDLING

### 3 LEFT\$

LEFT\$(A\$, I) returns the leftmost I characters of A\$, thus, using the strings from the "Happy Birthday Program"

```
10 INPUT ("YOUR NAME? ") N$  
20 A$ = "HAPPY BIRTHDAY"  
30 B$ = A$ + "TO YOU"  
40 C$ = "DEAR" & N$
```

```
PRINT LEFT$(A$,5)  
HAPPY
```

The second argument, 5 above, may be a variable

```
100 FOR I = 1 TO LEN (A$) -1  
110 PRINT LEFT$(A$, I)  
120 FOR T = 1 TO 100: NEXT T  
130 NEXT I  
140 END
```

producing

```
H  
HA  
HAP  
HAPP  
HAPPY  
HAPPY  
HAPPY B  
HAPPY BI  
HAPPY BIR  
HAPPY BIRT  
HAPPY BIRTH  
HAPPY BIRTHD  
HAPPY BIRTHDA  
HAPPY BIRTHDAY
```

### 4 MID\$

MID(A\$, I) returns the rightmost characters from the string A\$, starting with the Ith character

```
PRINT MID$(A$, 5)  
Y BIRTHDAY
```

If I is greater than the length of the string A\$, MID\$ returns the null string.

MID\$(A\$, I, J) returns a string of length J, starting with the Ith character in A\$.

```
PRINT MID$(A$, 7, 5)  
BIRTH
```

If J is greater than the number of characters in A\$ to the right of I, MID\$ returns the string from the Ith character:

```
PRINT MID$(A$, 7, 15)  
BIRTHDAY
```

The example used for LEFT\$, may be modified for MID\$, LINE 110 should be changed to

```
110 PRINT MID$(A$, I, 1);
```

The program prints H, followed by a delay (LINE 120) then A, delay, then P etc. A further variant is to amend line 120 to

```
120 PRINT " ";
```

producing spaced printing

```
H A P P Y     B I R T H D A Y
```

## CHAPTER 8 – STRING HANDLING

### 5 RIGHTS

RIGHT\$(A\$, I) is analogous to LEFT\$(A\$, I), producing the right most I characters of A\$.

```
PRINT RIGHTS (A$, 3)  
DAY
```

If I is equal to or greater than the length of A\$, RIGHTS returns A\$.

### 6 INSTR

INSTR (A\$, B\$, I) searches the string A\$, from the Ith character for an occurrence of B\$. If the third parameter is missing INSTR searches the whole of string A\$.

```
? INSTR (A$, "DAY")  
12
```

```
? INSTR (A$, "BIRTH", 10)  
0
```

In the latter case the result is 0, since "BIRTH" commences before the tenth character of "HAPPY BIRTHDAY". Note however this exception

```
? INSTR ("ABC", "")
```

gives

1

## **CHAPTER 9**

### **CONVERSION**

The chapter describes the functions available to convert ASCII codes to their corresponding characters, and vice versa, and functions to convert between string expressions and numeric expressions. NUM, which tests a string for numeric content, is also discussed.

- 1. CHARACTER/ASCII**
  - 1.1 Character to ASCII – ASC
  - 1.2 ASCII to Character – CHR\$
- 2. STRING/NUMBERS**
  - 2.1 String to Numeric – VAL
  - 2.2 Numeric to String – STR\$
- 3. TEST STRING FOR NUMBER**
  - 3.1 NUM

## CHAPTER 9 – CONVERSION

### 1 CHARACTER/ASCII

#### 1.1 ASC

This chapter discusses intrinsic functions which convert data from one form to another. Since a computer stores all forms of data, for example letters, numbers and punctuation marks, as numbers, it must have a code to convert its internal numbers into their corresponding letters etc. A variety of codes are commonly employed for this purpose, of which the most widely used is the ASCII code. (ASCII stands for American Standard Code for Information Interchange). ASCII code is issued by New-Brain computers both for internal data storage and transfer, and for communication with external devices.

`ASC(A$)` returns the ASCII code for the first character of the string `A$`. If `A$ = "ABLE"`, or `"ANCHOR"`, or `"A"`, then `ASC (A$)` returns the code for `A`.

```
PRINT ASC (A$)  
65
```

Thus `A` is stored internally as 65.

`ASC` may be made to operate on characters other than the first in a string, by using the form `ASC (B$)` where `B$ = (MID$(A$, I, 1))`, for example.

```
10 A$ = "ABCDEFGHIJKLMNP  
QRSTUVWXYZ"  
20 FOR I = 1 TO LEN (A$)  
30 B$ = MID$ (A$, I, 1)  
40 PRINT B$; ASC (B$),  
50 NEXT I  
60 END
```

80

### 1 CHARACTER/ASCII

`B$` need not be defined, explicitly, the following produces the same result

```
10 A$ = "ABCDEFGHIJKLMNO  
PQRSTUVWXYZ"  
20 FOR I = 1 TO LEN (A$)  
30 PRINT MID$ (A$, I, 1);  
    ASC (MD$ (A$, I)),  
40 NEXT I  
50 END
```

A 65	B 66	C 67	D 68
E 69	F 70	G 71	H 72
I 73	J 74	K 75	L 76
M 77	N 78	D 79	P 80
Q 81	R 82	S 83	T 84
U 85	V 86	W 87	X 88
Y 89	Z 90		

It is easier to derive a table of ASCII codes, however, by using the function `CHR$`.

## CHAPTER 9 – CONVERSION

### 1 CHARACTER/ASCII

#### 1.2 CHR\$

CHR\$(N) converts a number, N, into its equivalent CHaracteR.

```
PRINT CHR$(65)  
A
```

Thus CHR\$ is the inverse function to ASC. The ASCII conversion table may be derived by a program of the form –

```
10 FOR I = 1 TO N  
20 PRINT I; CHR$(I),  
30 NEXT I  
40 END
```

However the first 32 codes (0 to 31) are used by the NewBrain as CONTROL CODES, used for editing etc., and 31 in particular is the code for SH/HOME or Clear Screen. Thus

```
PRINT CHR$(31)
```

will clear the screen of any previously displayed characters. (Control codes are discussed more fully in Appendix 3.) The ASCII character set is defined up to CHR\$(127), thereafter NewBrain uses the internal codes for graphics characters. The program is thus

```
10 FOR I = 32 TO 127  
20 PRINT I; CHR$(I),  
30 NEXT I  
40 END
```

### 1 CHARACTER/ASCII

and the output is of the form

32	33	!	34	"	35	#
36	\$	37	%	38	&	39
40	(	41	)	42	*	43
44	,	45	-	46	.	47
48	0	49	1	50	2	51
52	4	53	5	54	6	55
56	3	57	9	58	:	59
60	<	61	=	62	>	63
64	@	65	A	66	B	67
68	D	69	E	70	F	71
72	H	73	I	74	J	75
76	L	77	M	78	N	79
80	P	81	Q	82	R	83
84	T	85	U	86	V	87
88	X	89	Y	90	Z	91
92	\	93	]	94	↑	95
96	'	97	§	98	b	99
100	d	101	a	102	f	103
104	h	105	i	106	j	107
108	l	109	m	110	n	111
112	p	113	q	114	r	115
116	t	117	u	118	v	119
120	x	121	y	122	z	123
124	:	125		126	~	127

## CHAPTER 10 – GRAPHICS

### 1 SPECIAL SCREEN CHARACTERS

Certain types of graphic presentation may be shown on the normal screen display by using the MOSAIC GRAPHICS CHARACTERS in the character sets shown in Appendix 5. These each have a numeric equivalent in the range 0–31 or 128–159. As each of these occupies the entire 8 x 10 character frame, they can be placed side by side to form complete pictures, as is indicated by the name "mosaic". As each character set features different mosaic characters, a sequence such as CTRL/W B must be typed to select a character set.

CTRL/W sets the TV control mode. This is best illustrated by an example. Enter the program

```
10 FOR I = 1 TO 255  
20 PUT 27, I, 26  
30 NEXT I  
40 END
```

and RUN it. The character set will be displayed with a space between each character. The effect of T.V. control mode may be seen by typing CTRL/W followed by A, B, C or D for the lower character ROM set, and H, I, J, K for the upper character ROM set. See Section 5.2 of Appendix 5 for details.

The mosaic characters in the range 129–158 can be typed from the keyboard directly by holding down the GRAPHICS key; the mosaic characters in the range 1–30 may be typed by first

### 1 SPECIAL SCREEN CHARACTERS

typing SH/↑, then holding down the GRAPHICS key while typing A–Z (–)+= Don't forget to type SH/ESCAPE afterwards! All the mosaic characters in a selected character set can also be generated by a suitable BASIC command, such as

PUT132, 152, 131  
or  
PUT27, 16, 27, 12, 27, 23

## CHAPTER 10 – GRAPHICS

### 2 HIGH-RESOLUTION DISPLAY

The screen display is normally in a state described as "low-resolution" or "character-oriented". In this state, each byte in the video area of memory corresponds to a single character in the current character set. It is also possible to arrange matters so that part of the screen is in a "high-resolution" state, wherein the colour (on or off) of each dot may be individually selected. This is achieved from BASIC by opening a stream of type 11, which must be linked to a stream already open of the right type, i.e. 0 (screen only) or 4 (screen and line display). The type 11 screen then shares the memory area originally occupied by the type 0 or type 4 screen. As the high-resolution display requires more memory to cover the same area of the screen, the type 0 or type 4 stream must be opened with a very large depth,

```
OPEN#0,4,"200": OPEN#1,11,"150"
```

For the full syntax of the parameter string used when opening a type 11 stream, see Appendix 7. As the graphics stream is "parasitic" upon the linked stream given in this parameter, it cannot function after that stream has been closed (even by an OPEN#0 implicit close), and must therefore also be closed.

### 3 THE GRAPHICS "PEN"

The PLOT statement is an extension to BASIC to allow the user to handle the graphics stream conveniently. It must be understood in terms of the concept of a "pen" associated with the graphics stream. This pen may be moved about on the screen by the various PLOT commands, and will then generally leave a visible trail. Some of the available commands change the direction or the colour of the pen, others alter particular controlling attributes of the graphics stream.

It is possible for the pen position to be off the screen. The pen may still be moved around, and if it is moved onto or across the screen (with a suitable colour), a visible trail will result.

The PEN function allows certain attributes of the pen at a given time to be determined.

## CHAPTER 10 – GRAPHICS

### 4 THE PLOT COMMANDS

The syntax of a PLOT statement is either PLOT # stream-number, plot-list or

PLOT plot-list

The stream number should be a Graphics stream, otherwise unpredictable effects may be produced. If the stream-number is not given, the first-opened graphics stream is assumed if it is still open.

The plot-list consists of one or more of the following plot items, separated by commas. Each plot item has a three-letter mnemonic; many plot items require one, two or three parameters to be provided in parentheses.

MOVE (x,y) moves the pen to position (x, y), drawing as it goes.  
The pen angle is set to the direction taken from the previous pen position.

MOVEBY (d) moves the pen by a distance d in its current direction, drawing as it goes.

TURN ( $\theta$ ) turns the pen to face direction  $\theta$ .

TURNBY ( $\theta$ ) turns the pen through an angle  $\theta$ .

PLACE (x,y) moves the pen to position (x,y) without drawing.

### 4 THE PLOT COMMANDS

BACKGROUND (b)  
BCK (b)

Sets the background colour to b. Values are  
0 – off ("white")  
1 – on ("black")  
other – undefined.

COLOUR (c)  
COL (c)

Sets the pen colour to c. Values are  
0 – leave alone  
1 – contrast with background  
2 – same as background  
3 – invert  
other – undefined.

WIPE  
WIP

Clears the entire screen to background colour.

DRAW(x,y,c)  
DRW(x,y,c)

Draws a line from the pen position to the point (x,y) in colour c.

DRAWBY (d,c)  
DBY (d,c)

Draws a line of length d in the colour c, in the direction of the current pen angle.

DOT (x,y,c)

Marks a single point at position (x,y) in colour c.

RADIANS  
RAD

Calls for angles to be given and returned in radians until DEGREES is executed.

DEGREES  
DEG

Calls for angles to be given and returned in degrees until RADIANS

## CHAPTER 10 – GRAPHICS

### 4 THE PLOT COMMANDS

	is executed.
RANGE (a,b)	Sets the horizontal and vertical ranges.
CENTRE(x,y)	Sets the position of the plotting origin, relative to the bottom left corner of the screen.
FILL FIL	Fills in the area around the current pen position up to a boundary, or the edge of the screen.
	ARC (D, $\theta$ ) draws a circular arc of length D, turning through an angle $\theta$ .
AXES (a,b) AXE (a,b)	Draws annotated axes crossing at the current pen position. The X-axis is marked up with a spacing a, and the Y-axis is marked up with a spacing b. If a or b is zero, the axis is drawn but not marked.
text	The string is plotted, starting from the current pen position, in a sequence of 8 x 10 frames. The current setting of the MODE is used to determine the surround in which the text is plotted.

### 4 THE PLOT COMMANDS

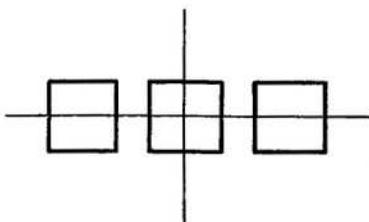
MODE (m) Sets the text plotting mode. Values are as for colour.

Note only ARC, MOVE, MOVEBY and PLACE change the pen position, and only ARC, MOVE, TURN, TURNBY change the pen angle. However, RANGE and CENTRE alter the value of the X-coordinate and Y-coordinate of the pen, to correspond to the same point on the revised co-ordinate system.

Example 1: some boxes

```
10 OPEN#0,0,"200":OPEN#129,  
11  
20 PLOT RANGE (4,2.4),  
CENTRE (2,1.2), DEGREES  
30 PLOT PLACE (0,0), AXES (0,0)  
(1,0.5)  
40 FOR I = 1 TO 3: PLOT PLA  
(I-2.4,0.4), TURN (0)  
50 FOR J = 1 TO 4: PLOT MBY  
(.8), TBY (-90): NEXT J  
60 NEXT I  
70 END
```

This will draw the picture



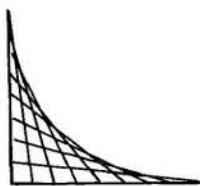
## CHAPTER 10 – GRAPHICS

### 4 THE PLOT COMMANDS

Example 2: an “envelope curve”

```
100 PLOT RNG (250,150)
110 PLOT CENTRE (10,10), WIPE
120 FOR I = 0 TO 20
130 PLOT PLA (I * 10,0), MVE
    (0,140-7*I)
140 NEXT I
150 END
```

GOTO 100 will plot this envelope



Example 3: a “pie chart”

This complete program uses many of the PLOT facilities. It makes use of a screen other than the console for the high-resolution display, to leave the full depth of a screen available to the console stream.

```
10 CLOSE#1:OPEN#1,4,2,”s200”
20 CLOSE#2:OPEN#2,11,
    ”#1w180”
30 plotrange(8,4.4),cen(4,2),deg
40 plotpla(0,-1),turn(0),arc(2*PI,
    360)
```

```
50 ?#1,”Title –”
60 LINPUT#1,a$
70 plotplace(-2,2,1),a$
80 ?#1,”Total amount:”
90 INPUT#1,t
100 j=2
110 PUT#1,31
120 ?#1,”Enter subheadings and
    amounts”
130 PUT#1,22,1,3,2,30
140 ?#1,”Sub-heading:”
150 LINPUT#1,a$
160 PUT#1,22,1,3,2,30
170 ?#1,”Amount of sub-heading
    ”;a$;” – ”
180 INPUT#1,x
190 IF r+x<=t THEN GOTO 220
200 ?#1,”Too large (only”;t-r;”
    left)”
210 GOTO170
220 a=360*t/b=360*x/t
230 plotpla(0,0),turn(a),dby(1,1)
240 plotturn(a+b),dby(1,1)
250 plotturn(a+b/2),col(0),mby
    (1,2),col(1)
260 IF pen(0)>0 THEN GOTO 290
280 l=LEN(a$)*.18:plotplace(pen
    (0)-l,pen(1))
290 plota$
300 j=j+1:plot pla (0,0)
310 IF j<3 THEN GOTO 330
320 plotcol(0),turn(a+b/2),mby
    (.9),col(1),fill:j=1
330 a=a+b:r=r+x
340 IF r<t THEN GOTO 130
350 ?#1,”Type NEWLINE to end”;
360 LINPUT#1,a$
370 END
```

## CHAPTER 10 – GRAPHICS

### 4 THE PLOT COMMANDS

- LINE 160 Places cursor and blanks out two lines.
- LINE 250 Positions pen just outside the circle.
- LINE 280 Provides room for the text on the left (the value 0.18 is found by experiment).
- LINE 300 Allows GOTO 300 if a sub-heading is too and gives an error by going off the screen.
- LINE 360 Ensures the completed display is visible.

### 5 THE PEN FUNCTION

The function PEN returns as a number the value of a parameter pertaining to a graphics stream. It is used like any function, and has two forms, so that the stream number may be omitted:

PEN (# stream number, parameter)  
PEN (parameter)

If the second form is used, the default graphics stream number is used, as for PLOT. Unpredictable results may occur if the stream number used is not a graphics stream. The parameters which may be returned are:

PEN (0)	X-coordinate of pen
PEN (1)	Y-coordinate of pen
PEN (2)	pen angle (*)
PEN (3)	pen colour
PEN (4)	background colour
PEN (5)	mode
PEN (6)	colour of point at current pen position

- (\*) The pen angle is given in degrees if a PLOT DEGREES statement has been executed, unless cancelled by PLOT RADIANS.

## CHAPTER 10 – GRAPHICS

### 6 GRAPHICS STREAM DEFAULTS

When a graphics stream is opened, the pen is set as follows:

Range	1,1
Centre	0,0
Pen position	0,0
Pen angle	0
Angles in	Radians
Pen colour	1
Background	0
Mode	0

## **CHAPTER 11**

### **HELP IN AN EMERGENCY**

When using the NewBrain it is sometimes possible to reach a situation where the computer does not seem to respond, or where no sensible display is visible on the screen or line display. Usually this problem can be cleared fairly easily. However, the misuse of certain BASIC commands can produce a situation where the only recourse is to switch off.

- 1. INTERRUPTION**
- 2. CHANGING THE DISPLAY**
  - 2.1 Character Set**
  - 2.2 Console Device**
- 3. FREEING THE KEYBOARD**
- 4. RELEASING MEMORY**
- 5. WHEN TO SWITCH OFF**
  - 5.1 OPEN #0**
  - 5.2 ON BREAK and ON ERROR**
  - 5.3 POKE**
  - 5.4 CALL**

## CHAPTER 11 – HELP IN AN EMERGENCY

### 1 INTERRUPTION – STOP

Normally any BASIC command or program may be interrupted by pressing the STOP key. The following program will loop indefinitely:

```
10 GO TO 10
```

but pressing STOP will restore control to the keyboard.

When the command being executed is a cassette READ or VERIFY, the asterisk key "\*" is required instead of the STOP key to interrupt the command.

### 2 CHANGING THE DISPLAY

#### 2.1 Character Set

The various character sets are shown in Appendix 5. The 8 x 8 characters shown there are not easy to read. If the display shows these characters when the keys are pressed, type CTRL/W followed by B to obtain the normal display mode. This will also be effective if the screen is a T.V. set, as these will not show a black-on-white display as clearly as white-on-black.

#### 2.2 Console Device

If there is no visible display at all, and no flashing cursor, then the console stream may be set to the wrong device for your model of NewBrain. Note that the console should be one of the following device types:

- |   |   |                       |
|---|---|-----------------------|
| 0 | – | screen only           |
| 3 | – | built-in display only |
| 4 | – | combined display      |

If there is no screen device connected to your NewBrain, but it has a built-in display, the console stream should be type 3. If there is no built-in display, a monitor or TV must be connected and the console stream should be type 0. The user should type the following carefully, although the effect may not be seen until it is completed:

## CHAPTER 11 – HELP IN AN EMERGENCY

### 2 CHANGING THE DISPLAY

CTRL/HOME to blank the current line; if any  
NEWLINE to clear a STOP condition in case the STOP key was pressed earlier.  
OPEN # 0, 0 to select a screen, or  
OPEN # 0, 3 to select the line display.

### 3 FREEING THE KEYBOARD

CTRL/0 may be typed to cancel a SHIFT lock.  
SH/ESCAPE may be typed to clear the condition called "attribute on". This condition is entered by means of SH/ $\uparrow$  and allows the keyboard to be used to produce characters which are not normally obtainable. These "attribute on" characters are not intelligible as BASIC commands, hence the need to use SH/ESCAPE.

## CHAPTER 11 – HELP IN AN EMERGENCY

### 4 RELEASING MEMORY

When all available memory is used up, the NewBrain may respond to even the simplest command with ERROR 10, i.e. insufficient memory. The areas in which memory is used up are:

- Program lines
- Memory for streams
- Variables and arrays

To obtain more memory for use, it may be necessary to reduce the amount required in one of these areas.

A single program line may be removed from memory by simply typing the line number, followed by NEWLINE. This is the most effective way of freeing memory, as no extra memory is required in order to do so.

An unneeded stream may be closed. The amount of memory freed by this will depend on the device driver type.

Storage for variables and arrays may be released with the CLEAR command. However, the selective form

**CLEAR** list of items

requires more memory for its execution than may be available.

If the insufficient memory error is frequently encountered, it is probably necessary to extend the NewBrain configuration by adding a RAM extension box.

### 5 WHEN TO SWITCH OFF

The following statements, if misused, may jam the NewBrain to the extent that it must be switched off to regain the use of the console. As switching off the computer loses the program in memory, these statements should be used with the greatest care.

#### 5.1 OPEN #0

The console stream, number 0, should always be assigned to a device which is capable of input through the keyboard. The form

**OPEN #0, X**

which assigns the console stream to device type X will only be sensible if X is 0, 3 or 4.

In particular,

**OPEN #0**

which assigns the console to the default back-up store device should be avoided.

#### 5.2 ON BREAK and ON ERROR

The STOP key is provided to allow the user to escape from otherwise impossible conditions. If this key is interrupted with an ON BREAK statement then a program may tie up the computer completely.

## CHAPTER 11 – HELP IN AN EMERGENCY

### 5 WHEN TO SWITCH OFF

E.g. the program

```
10 ON BREAK GO TO 20
20 GOSUB 20
```

will fill up all available memory without any possibility of interruption by the user.

### 5.3 POKE

The POKE statement is used to insert a value into a memory location. As many locations are used to provide essential pointers for the operating system on the hardware, as well as control information used by BASIC, the operation of the computer can be completely upset by misuse of POKE. The best way, for instance, to enter an assembly coded routine into memory is shown in this program.

```
1 REM echoes characters until "Q"
10 RESERVE 100 : DELETE 10
20 M = TOP + 1
30 FOR I = M TO M+39
40 READ X : POKE I, X
50 NEXT I
60 DATA 30, 5, 231, 52, 30, 3,
231, 52
70 DATA 30, 5, 123, 231, 50, 216,
30, 3, 123, 231, 51, 216, 30, 5,
62, 1
80 DATA 231, 48, 231, 54, 216,
30, 5, 231, 49, 254, 81, 200,
30, 3, 24, 240
```

### 5 WHEN TO SWITCH OFF

```
90 CALL M
100 END
```

### 5.4 CALL

The CALL statement allows an assembly-coded routine to be entered from BASIC. Such a routine must preserve certain registers needed by BASIC, and must exit by means of one of the RET codes. Details are given in Appendix 6. If these conventions are not observed, or essential memory locations are changed, then the routine will not satisfactorily return control to BASIC.

# **APPENDICES**

- 1. ERROR NUMBERS**
- 2. BASIC TECHNICAL SPECIFICATION**
- 3. SCREEN EDITOR TECHNICAL SPECIFICATION**
- 4. BASIC RESERVED WORDS**
- 5. LINE AND SCREEN DISPLAY CHARACTER SETS**
- 6. BASIC STATEMENT KEYWORDS**
- 7. DEVICE DRIVER SUMMARY**
- 8. CALL STATEMENT AND O/S ROUTINES**
- 9. HARDWARE SPECIFICATION**

# **APPENDIX 1**

**1**



# APPENDIX 1 ERROR NUMBERS

(S) Indicates a syntax error.  
(R) Syntax error also a run-time error.

- |          |  |
|----------|--|
| 0,1      | Not used.  |
| 2        | Arithmetic error.<br>E.g. division by 0, arc sin of value > 1.   |
| 3        | No further statement to execute, but no END or STOP statement found.   |
| 4 (S)    | Illegal line number.<br>E.g. 1000000 A = 1.  |
| 5        | Not used.  |
| 6        | Illegal value — value in the range 0 .. 65535 required.<br>E.g. in an array subscript or a switch value in ON.                                       |
| 7        | Illegal array subscript value. A subscript value must be in the range 0 .. dimension of array (or 1 .. dimension, when OPTION BASE 1 has been used). |
| 8,9      | Not used.  |
| 10       | Insufficient memory space.   |
| 11,12,13 | Not used.  |
| 14 (S)   | Something other than IN #, OUT # or # follows OPEN.<br>E.g. OPEN ?2.   |
| 15,16    | Not used.  |
| 17 (S)   | Error in numeric function argument.<br>E.g. LOG (X - "A") LOG (2).   |

# APPENDIX 1 ERROR NUMBERS

- |          |   |
|----------|---|
| 18       | Not used.   |
| 19       | Wrong number of subscripts in array element.<br>E.g. DIM A(5) : B = A (2,3)                         |
| 20 (S)   | Expression of wrong type.<br>E.g. A\$ = 2 N = "PQR".  |
| 21 (S)   | Error in expression: unrecognisable thing.<br>E.g. X(A ? 5).  |
| 22 (S)   | Error in expression: type mismatch.<br>E.g. A\$ * 4.  |
| 23,24,25 | Not used.   |
| 26 (S)   | Name of variable or keyword doesn't begin with a letter.<br>E.g. * PRINT 2<br>FOR, = 1 TO 10.       |
| 27       | Not used.   |
| 28       | Switch value in ON is 0 or greater than the number of line numbers in the list.                     |
| 29       | Attempt to GOTO (or goto) a line which doesn't exist.   |
| 30       | Error in input to an INPUT statement. If the input is from the console the prompt will be repeated. |
| 31 (S)   | In a PRINT statement a comma or semicolon was expected but something else was found.                |
| 32       | Not used.   |
| 33       | RETURN without corresponding GOSUB.   |
| 34 (S)   | GOTO or GOSUB not found where expected in ON.   |

# APPENDIX 1 ERROR NUMBERS

- 35 (S) Hyphen or end of line expected in LIST or DELETE.  
E.g. LIST 1000 + 20  
DELETE 22,23
- 36 Bad INPUT.  
E.g. quotation mark not allowed in unquoted  
INPUT string.
- 37 Can't TAB to column Ø.
- 38 Can't POKE a value > 255.
- 39 Insufficient DATA for READ.
- 40 (S) Illegal item in CALL parameter list.
- 41 NEXT without FOR.
- 42 (S) Empty DATA line.
- 43 Not used.
- 44 (S) Illegal control variable in FOR statement.  
E.g. FOR A\$ = 1 TO 10.
- 45 (S) A syntax error exists in a FOR or NEXT statement  
within the for-next loop which begins at this statement.  
E.g. 10 FOR I = 1 TO 10  
20 FOR 2 = 1 TO 2  
30 NEXT J  
40 NEXT I  
will give ERR 45 AT 10
- 46 (S) TO not found where expected in a FOR statement.  
E.g. FOR I = 3 T 7.
- 47 (S) STEP not found where expected in a FOR statement.  
E.g. FOR I = X TO X + 17! 2.

## APPENDIX 1 ERROR NUMBERS

- |                |     |  |
|----------------|-----|--|
| 48             | (S) | No NEXT statement found to match the current FOR statement.  |
| 49             | (S) | Illegal for-next loop nesting.   |
| 50,51          |     | Not used.  |
| 52             | (S) | Comma not found where expected.<br>E.g. INPUT # 5A, B.   |
|                | (R) | In reply to INPUT, or in DATA.   |
| 53             | (S) | End-of-line not found where expected.<br>E.g. LIST 5-100, 1000-2000.   |
|                | (R) | In reply to INPUT etc.   |
| 54             | (S) | CLOSE not followed by #.   |
| 55             | (S) | Equals sign not found where expected, or keyword misspelt.<br>E.g. PRIJT A,B HJGK<br>FOR I2TO7.                              |
| 56             | (S) | Open parenthesis not found where expected.<br>E.g. DIM A22,7)  |
| 57             | (S) | Closing parenthesis not found where expected.<br>E.g. TAB (2,  |
| 58, . . . , 62 |     | Not used.  |
| 63             | (S) | Neither THEN nor a keyword found after the conditional expression in an IF statement.<br>E.g. IF A BG3<br>IF C D?4THEN A = 1 |
| 64             |     | Not used.  |
| 65             | (S) | No closing quotation marks in a string constant.<br>(R) In response to INPUT.  |

# APPENDIX 1 ERROR NUMBERS

66,67	Not used.
68 (S)	OPTION not followed by BASE.
69	OPTION BASE used after an array has been created.
70 (S)	OPTION BASE not followed by 0 or 1.
71	Dimensioning of an array which already exists.
72	Dimension too large.
73	Dimension 0 when OPTION BASE 1 specified.
74 (S)	Error in formatter, other than error 75.
75 (S)	Number in formatter not in range 0 .. 255, or not present.
76,.. 79	Not used.
80 (S)	DEF not followed by FN.
81 (S)	Illegal user defined function name in DEF statement. E.g. DEF FN\$=A\$.
82	Not used.
83 (S)	No DEF statement for user defined function.
84	Redefinition of a user defined function to have a different number of arguments or reference to an array with the wrong number of dimensions.
85 (S)	Expression too complex to evaluate, or user defined function references too deeply nested to evaluate.
86	Not used.

# APPENDIX 1 ERROR NUMBERS

- |           |   |
|-----------|---|
| 87        | Can't CONTINUE.   |
| 88 (S)    | Illegal list of arguments to CLEAR statement.                     |
| 89        | Not used.   |
| 90        | Device stream or port not in range 0..255.                        |
| 91        | Failure to VERIFY.  |
| 92        | Can't CLOSE stream 0.   |
| 93 (S)    | ON ERROR or ON BREAK not followed by GOTO.<br>E.g. ON ERROR STOP. |
| 94 (S)    | Line number 0 not allowed.  |
| 95        | VAL error – string is not a number.<br>E.g. VAL ("1.1").          |
| 96        | LINPUT numeric variable.  |
| 97 (S)    | Attempt to ON ERROR or ON BREAK to a non-existent<br>line-number. |
| 98        | PUT error.  |
| 99        | Out of DATA.  |
| 100       | Insufficient memory to open stream.                               |
| 101,..104 | Not used.   |
| 105       | Stream not open.  |
| 106       | No such device.   |
| 107       | Device-port pair already open.                                    |

# APPENDIX .1 ERROR NUMBERS

- |                |  |
|----------------|--|
| 108            | Stream already open.   |
| 109            | System error.  |
| 110            | Syntax error in parameter string.  |
| 111            | Attempt to open device which requires mains power when no mains power connected.               |
| 112            | Insufficient memory for FILL request.  |
| 113            | Linked stream not a screen device.   |
| 114            | Requested height too large for memory available to the linked stream.                          |
| 115            | Linked stream has been closed.   |
| 116            | Position off the screen illegal in this context.   |
| 117            | Unrecognised PLOT command or PEN parameter.  |
| 118            | Cannot use input from graphics device (use PEN instead).                                       |
| 119            | Attempt to output to graphics device before input function completed.                          |
| 120            | Syntax error in baud rate parameter string.  |
| 121            | Port number other than zero for serial device.   |
| 122, . . . 129 | Not used.  |
| 130            | Tape read error : hardware failure (Tape dropout).   |
| 131            | Tape read error : attempt to read block into a buffer which is too small, or hardware failure. |

## APPENDIX 1 ERROR NUMBERS

- |           |   |
|-----------|---|
| 132       | Tape read error : hardware failure (Checksum error).              |
| 133       | Attempt to read past the end of a tape file, or hardware failure. |
| 134       | Attempt to read a tape file out of sequence, or hardware failure. |
| 135       | Attempt to output to a tape file opened for input or vice versa.  |
| 136       | Syntax error in parameter string.                                 |
| 137,..199 | Not used.   |
| 200       | Time out error on software serial input port.                     |
| 201,..255 | Not used.   |

# **APPENDIX 2**

**2**





# APPENDIX 2 BASIC TECHNICAL SPECIFICATION

---

## 1. DATA TYPES

---

**1.1 NUMBERS** These are handled by the mathematics package.

**1.1.1 Storage** Six bytes are allocated by BASIC for the storage of a number value. The mathematics package maintains and operates on numbers to a precision of 10 or more significant figures in a range of  $0 \dots \pm 10^{150}$ .

**1.1.2 Output** By default output is rounded to 8 significant figures and is in free format (integer, floating point or scientific "E" notation according to value). Output to specific field sizes and formats can be forced by format specifiers for the PRINT statement and the STR\$ function. Output range is  $0 \dots \pm 10^{99}$ .

**1.1.3 Input** Any output format is accepted as input, additionally any number of digits is allowed in the mantissa and spaces may be disregarded.

**1.1.4 Constants** Any number valid for input may be used as a numerical constant.

**1.1.5 Variables** Any simple name consisting of a letter or a letter followed by a letter or a digit may be used to denote numeric values. A variable may hold any valid number.

**1.2 STRINGS** In NewBrain BASIC a string is any sequence of bytes (i.e. numbers in the range 0 .. 255). Bytes often stand for characters, in particular those in the range 32 .. 127 stand for the ASCII printing characters.

**1.2.1 Storage** Strings can be of any length between 0 and 32767 bytes. An additional overhead of four bytes is required for each string stored. Storage allocation is dynamic (i.e. the length of a string can change during program or command execution).

**1.2.2 Output** Any string can be output. Input and output devices interpret bytes in different ways. For instance the keyboard screen editor device, which is usually the console for BASIC, interprets 0 .. 31 as control codes, 32 .. 127 as ASCII character codes and 128 .. 255 as mosaic graphics character codes.

**1.2.3 Input** Any valid string constant may be supplied in response to a BASIC INPUT or READ statement. If the constant does not contain quotation marks ("), commas or TAB (code 09) characters the enclosing quotation marks may be omitted. Comma and TAB characters are used by INPUT to separate consecutive strings. In response to LINPUT any string not containing a NEWLINE (code 13) character may be supplied.

## APPENDIX 2 BASIC TECHNICAL SPECIFICATION

**1.2.4 Constants** Any string not containing a NEWLINE character may be enclosed in quotation marks ("") and used as a string constant in a BASIC statement. Quotation marks within a string constant are denoted by doubled quotation marks ("").

**1.2.5 Variables** Any simple name followed by a dollar (\$) character may be used as a string variable name. A string variable may hold any string.

**1.3 LOGICAL** There is no special logical datatype, but numbers may be used to store logical values. In those cases where a logical value is required for a binary choice, -1 is taken as TRUE and all other values are taken as FALSE. Logical operations are performed bitwise on 16-bit words. In this sense -1 is TRUE, 0 is FALSE and other values from the range -32768 .. 32767 take intermediate truth value; some operations require arguments from this subrange.

**1.4 ARRAYS** Arrays may be of numbers or strings, of 1 or 2 dimensions to a maximum of 5575 elements. Array storage must be reserved by DIMensioning, but each element of a string array may vary in length during program or command execution. In addition to the storage required for the values of the elements there is a storage overhead of 6 bytes for an array and a further overhead of 2 bytes per element for a string array.

There is no provision for input/output of whole arrays.

Individual array elements may be treated the same way as numeric and string scalars for input/output.

There are no array constants.

Any number (or string) variable name may be used for a numeric (or string) array. A variable name thus used may also be used for a scalar of the same type, but may not be used for another array of the other dimension.

# APPENDIX 2 BASIC TECHNICAL SPECIFICATION

## 2. EXPRESSIONS

### 2.1 ATOMIC EXPRESSIONS The allowed atomic expressions are:

*constant*           *variable name*           *array element*           *function call*

### 2.2 MOLECULAR EXPRESSIONS Atomic expressions may be built up to form molecular expressions.

If X and Y are valid atomic expressions then

*unary operation X*           *X binary operation Y*           (X)

are valid molecular expressions subject to the type restrictions detailed below.

*Constant* This may be of any string or number constant.

*Variable Name* This may be any string or number variable name.

*Array Element* This is *array name (expression)* or *array name (expression, expression)*. The expressions involved must evaluate to numbers within the dimensions of the array.

Numbers are rounded to the nearest whole number for this purpose.

*Function Call* The general form is

*function name (arguments)*

The number and type of arguments depends on the function, given this an argument can be any expression of the correct type. Arguments are separated by commas.

#### 2.2.1 Functions with No Arguments

PI	The mathematical constant
RND	A pseudo-random number from the uniform distribution of the unit interval (0,1)
POS	The current position of the printhead on the console output device
TRUE	Evaluates always to -1
FALSE	Evaluates always to 0
ERRNO	The error number of the most recent error or break-in unless cleared
ERRLIN	The line number of line being executed at the time of the most recent error or break-in
FREE	The number of bytes of free store available to but not used by BASIC
TOP	The lowest store address not available to BASIC
FILE\$	The parameter string returned by the most recently OPENed input device

## APPENDIX 2 BASIC TECHNICAL SPECIFICATION

### 2.2.2 Functions with One Numeric Argument

INT	Integer part (INT(X)) is the greatest integer not greater than (X))
ABS	Absolute value (modulus)
SGN	Sign (-1 for negative argument, 1 for positive, 0 for zero)
SQR	Square root
SIN COS TAN	Trigonometric functions
ASN ACS ATN	Inverse trigonometric functions
LOG	Natural logarithm
EXP	Exponential function
PEEK	Contents of memory location whose address is the argument
PEN	Value of graphics parameter
CHR\$	The string consisting of the single character whose internal code is the argument
STR\$	The string consisting of the numeric output format of the argument The format may be forced. The expression STR\$(X [formatter]) produces a string in the format determined by the formatter. Allowed formatters are: n Integer format with n digits n.m Fixed point format with n digits before and m after the point n.mE Scientific format with n digits before the point, m after and 2-digit exponent for the power of 10 nF Free format in a field width of n In all formats leading zeroes in the mantissa are replaced by spaces, there is a leading space or minus sign and a trailing space.

### 2.2.3 Numeric Valued Functions with One String

Argument	
LEN	Length of the string -
VAL	The numeric value of the string, if the string happens to be in a valid number input format
NUM	= -1 (TRUE) if the string happens to be in a valid number input format = 0 (FALSE) otherwise
ASC	The NewBrain internal code for the first character of the string

# APPENDIX 2 BASIC TECHNICAL SPECIFICATION

(for characters in the ASCII set this coincides with the ASCII code)

## 2.2.4 Substring Functions

LEFT\$	LEFT\$(X\$,N) extracts the leftmost N character substring
RIGHT\$	RIGHT\$(X\$,N) extracts the rightmost N character substring
MID\$	MID\$(X\$,P) extracts the right hand substring starting at the P'th character of X\$
	MID\$(X\$,P,N) extracts the substring of length N starting at character P
INSTR	INSTR(X\$,Y\$) finds the numerical position of the string Y\$ in X\$
	INSTR(X\$,Y\$,P) finds the position in MID\$(X\$,P)

**2.2.5 User Defined Functions** These are numeric or string values with one or no numeric or string argument. A user defined function name is a simple name preceded by FN. User defined functions must be declared equal to an expression in a DEF statement.

**2.2.6 Unary Operations** These all take numeric arguments and yield numeric results.

+ - unary plus and minus

## APPENDIX 2 BASIC TECHNICAL SPECIFICATION

NOT      The bitwise Boolean operation

### 2.2.7 Binary Operations with Numeric Arguments which Yield Numeric Results

+ - \* /      Plus minus times divide  
↑      Raise to a power

< <= =      Relational operators  
> >= <>      (less than, less than or equal, equal, greater than,  
greater than or equal, not equal)

AND OR      The bitwise Boolean operations

### 2.2.8 Binary Operations with String Arguments which Yield Numeric Results

Results  
< <= =      Relational operators returning Boolean results as in  
> >= <>      the numeric case. Characters are ordered according to  
the NewBrain internal codes (which ordering agrees  
with ASCII and hence with alphabetic order) and  
strings are ordered by first difference.

### 2.2.9 Binary Operations with String Arguments which Yield String Results

+      Concatenation  
(also &)

## 2.3 EXPRESSION EVALUATION

Within an expression atomic expressions and expressions in parentheses are evaluated first. Operation are evaluated in the order

+ -      (unary operations)  
↑  
\* /  
+ - &      (binary operations)  
< <= = > >= <>  
NOT  
AND  
OR

OR having the least binding power

Mathematical operations and functions are evaluated by the mathematics package. The mathematics package has been specially designed to minimise cumulative rounding errors and maintain stability in recursive and iterative calculation, and to obtain maximal computation speed consistent with ten significant figure accuracy and long term stability.

# **APPENDIX 2 BASIC TECHNICAL SPECIFICATION**

---

## **3. INPUT/OUTPUT**

---

This is performed by the NewBrain operating system.

### **3.1 I/O DEVICES**

I/O devices are signified by whole numbers in the range 0 to 255. In practice in a program devices are usually named as numeric variables whose values signify the device. All input/output peripherals are configured by their device drivers as byte serial devices.

### **3.2 DATA STREAMS**

Data streams are signified by whole numbers in the range 0 to 255, though in practice numeric variable names are used. In BASIC, after opening a device on a datastream all communication with that device is via the datastream. When appropriate devices and device drivers are connected appropriate control datastreams can be used to implement all current I/O modes including random access.

# APPENDIX 2 BASIC TECHNICAL SPECIFICATION

---

## 4. BASIC STATEMENTS, BLOCKS AND PROGRAMS

---

A BASIC statement generally consists of a keyword followed by certain arguments. A BASIC line consists of statements (simple statements or FOR-block statements) separated by colons. A line may be used as a command, or introduced by a line number as part of a BASIC program. Line numbers are whole numbers in the range 1 . . 65535.

### 4.1 SIMPLE STATEMENTS

#### 4.1.1 Declaration

**OPTION BASE 0** Sets the *array base* globally to 0.

**OPTION BASE 1** Sets the base globally to 1.

The default base is 0.

**DIM array name (dimension)**

The dimension must be an appropriate numeric expression or a pair of expressions separated by a comma. Multiple dimensionings separated by commas are allowed. Scalars are not explicitly declared, arrays need not be. In the latter case a default dimension of (10) or (10,10) is assumed.

**DEF function name (argument) = expression** declares a user defined function.

#### 4.1.2 Assignment

**LET assignee = expression**

An assignee may be a variable name or an array element. The assignee must be of the type of the evaluated expression. The keyword LET may be omitted.

#### 4.1.3 Control

**IF expression THEN line number**

**IF expression THEN line**

In the latter case the keyword THEN may usually be omitted.

**GOSUB line number**

**RETURN (also RET)**

**ON expression GOTO line number list**

**ON expression GOSUB line number list**

**ON ERROR GOTO line number** Set error trap.

**ON BREAK GOTO line number** Set interrupt trap.

**RESUME** Resume after trap.

## APPENDIX 2 BASIC TECHNICAL SPECIFICATION

RESUME *line number*

GOTO *line number*

CONTINUE (also CONT)

STOP

RUN

NEW

CALL *expression, arguments*

This is to call a machine code or other language routine

### 4.1.4 Input and Output

OPEN *direction #stream, device, port, parameter string* Direction is IN or OUT, it is optional and defaults to IN. Stream is an appropriate numeric expression. Device and port are numeric expressions which may be omitted. Parameter string is a string expression which defaults to null. The information is passed to the operating system which assigns the stream to the device and opens it.

INPUT *assignee list* INPUT *#stream, assignee list*

INPUT *(prompt) assignee list*

The *prompt* string expression us used to substitute for the default "?" prompt when the stream is the default (console) stream.

PRINT *print list* PRINT *#stream, print list ?*

? is a synonym for PRINT.

*Print list* is a sequence of *print items* and *print separators*. *Print items* must be separated by at least one *print separator*, and can be

*expression*

*numeric valued expression [formatter]*

TAB (*numeric valued expression*)

The format may be forced. The expression

[*formatter*]

produces output in the format determined by the formatter. Allowed formatters are:

n Integer format with n digits

n.m Fixed point format with n digits before and m after the point

n.mE Scientific format with n digits before the point, m after and 2-digit exponent for the power of 10.

## APPENDIX 2 BASIC TECHNICAL SPECIFICATION

*for-statement blocks next-statement*  
*for-statement is*

FOR *variable* = *initial value* TO *limit* STEP *increment* *initial value*, *limit* and *increment* are numeric valued expressions.

STEP *increment* may be omitted.

*next-statement is*  
NEXT *variable*

The control variable must be numeric and the same in the for and in the next statement.

### 4.3 PROGRAMS

A program is both a sequence of lines and a sequence of blocks. An END statement should be present.

## **APPENDIX 2 BASIC TECHNICAL SPECIFICATION**

---

### **5. MODE OF OPERATION**

---

A certain input/output device, called the console, is open when BASIC is started up. Input from this device is treated by BASIC either as lines which are commands to be obeyed immediately, or as lines with line numbers which are added to the current program to be obeyed later when the program is run. Reserved words in lines are "entokened" in the program in order to minimise store usage. When a command or block is obeyed it is first "compiled" into NewBrain BASIC "object code" (this compilation includes the setting up of appropriate datastructures) and then the "object code" is "executed". When a program is being run the object code is kept and so once obeyed a command can be obeyed again without having to be compiled again. This mode of operation saves time and optimises the performance of programs.

Despite BASIC being a dynamic compiler, all interactive features are present. While an ON BREAK trap is not set the user can normally break into a program by using the STOP key, inspect and alter values and program lines and then CONTINUE execution with all other states preserved.

AS

## **APPENDIX 2 BASIC TECHNICAL SPECIFICATION**

---

### **6. NOTES**

---

**(a) Spaces**

Spaces are insignificant in almost all places in NewBrain BASIC.

**(b) Character Set**

NewBrain BASIC allows the user a set of 256 distinct characters. It is assumed that the character set includes the ASCII characters (though it need not include all of them). BASIC distinguishes between upper and lower case alphabetic characters only within string constants and REMark strings. The NewBrain keyboard character set includes all the ASCII characters, viewdata graphics characters and others as well.

**(c) Errors**

NewBrain BASIC produces over 90 numerically coded error messages to aid debugging.

**(d) Nesting**

The depth of nesting of for-blocks, GOSUBs, parentheses etc. is limited only by the total amount of memory available.

**(e) Extension**

The NewBrain operating system contains an extension mechanism comparable to but more advanced than simple 'trapping'. This makes NewBrain 'ROM software' real software, not just 'firmware'. Programs in RAM or ROM can replace or extend programs already present in ROM. In particular additional features can be added to NewBrain BASIC without having to replace the original ROMs. The NewBrain paged memory system enables such extension to be virtually indefinite.

# **APPENDIX 3**

**3**





# **APPENDIX 3 SCREEN EDITOR TECHNICAL SPECIFICATION**

## **Screen Editor – Technical Specification**

### **General Description**

The NewBrain keyboard-Screen-vf display editor, XIO, is an interactive input/output device for communication between a user or operator of the NewBrain and a NewBrain user-program such as BASIC. The editor interfaces with the NewBrain Input/Output System, IOS.

Communication between the editor and the program is via the five standard IOS commands, OPENIN, OPENOUT, INPUT, OUTPUT and CLOSE. Since the editor is an Input/Output device OPENIN is equivalent to OPENOUT.

### **The Displays**

The TV/video display editor holds a page of between 1 and 255 lines of 40 or 80 characters per line. The screen display will show 24 or 30 lines and this window is scrolled up and down the page. IOS allows for multiple copies of a device to be open; thus, memory allowing, up to 255 pages can be simultaneously maintained.

The vf display editor holds a single line between 16 and 254 characters in length. The 16 character vf display window is scrolled backwards and forwards over the line. The editor waits for the user to press the NEWLINE key before displaying a new line. This wait can be suppressed by outputting the appropriate control codes. During the wait the window can be scrolled from side to side by the cursor control keys.

The vf display can also be used as a window on a screen page, displaying the part of the current line around the cursor.

### **Character Sets**

The vf display character set consists of the 64 ASCII upper case characters, excepting character 95 for which the £ sign is substituted, and 64 graphics characters. All characters can be blinked. The vf editor uses only 65 of these characters and reserves blinking as an indication of cursor position.

Characters sent to the editor are recoded before display, so that lower case letters are displayed as their upper case equivalents, and so the coding agrees with that used for the screen display.

The screen display character set depends on the character generator ROM fitted. 512 characters are available, though at most 255 characters can be displayed at a time. Character set selection is achieved by the "Set TV Mode" (control W) code. In certain modes 127 characters are available for display, in normal or reverse field. According to mode the background can also be set

## **APPENDIX 3 SCREEN EDITOR TECHNICAL SPECIFICATION**

### **Output**

Calls to output cause a character to be put onto the screen or vf display. Thirty-two characters are used as control codes, accessing special features of the editor. The effect of control codes other than NEWLINE is the same whether entered via OUTPUT or entered via the keyboard interaction in response to a call to INPUT, when the cursor is at the beginning of a line.

### **Screen lines, Cursor display, Autodelete feature**

Lines on the screen may be of any length up to the size of page. Lines longer than 40 or 80 characters wrap round, a continuation character being displayed at the start of each screen line other than the first to indicate that this is a continuation line. The cursor is displayed as a blinking underline or a blinking block. The latter mode of display is used for two purposes. When the cursor is beyond the right end of the line — i.e. the next character entered will be the first character of a subsequent continuation, it is displayed at the right-most position in the line as a block. This contrasts with being displayed in the right-most position as an underline when the next character entered is to be entered at that position. The other use of the block cursor is at the left-most position of a line just after a NEWLINE has been entered. This indicates that if the next character entered is not a control code the present contents of the line will be cleared to spaces. This "autodelete" feature enables a page to be overprinted without having to clear outdated information. The autodelete mode is cancelled if the first character entered is a control code, so that overprinting of forms etc, where information previously printed is to be retained, can be achieved.

### **Use as BASIC console**

BASIC opens an editor device on Stream 0 when started up. The parameters used for this depend on the hardware. For a NewBrain D, MDB L1I0 with a length of 80 is used, for A or AD TL10 with a length of 40 and depth of 24, for M, MD TV10 with a length of 80 and depth 24. Port number 0 is used.

# APPENDIX 3 SCREEN EDITOR TECHNICAL SPECIFICATION

## CONTROL CODES

HEX	DECIMAL	CONTROL	KEY	
0	0	@		Null
1	1	A	Sh/insert	Insert line
2	2	B	Sh/↓	Delete line
3	3	C	Cntl/newline	Send page
4	4	D		End of file
5	5	E		Send Line
6	6	F		Show cursor
7	7	G		Cursor off
8	8	H	←	Cursor left
9	9	I	Cntl/escape	Tab 8 spaces
A	10	J	↓	Cursor down
B	11	K	↑	Cursor up
C	12	L	Home	Cursor home
D	13	M	Newline	Newline
E	14	N	Sh/↑	Attribute on
F	15	O	Sh/escape	Attribute off
10	16	P		Graphics escape
11	17	Q	Insert	Enter insert mode
12	18	R	Grph/↑	Make new line
13	19	S	Grph/↓	Make continuation line
14	20	T		Send cursor character
15	21	U		Send x, y
16	22	V		Set cursor x, y
17	23	W		Set TV control
18	24	X	Sh/←	Delete left
19	25	Y	Sh/→	Delete character
1A	26	Z	→	Cursor right
1B	27	[	Escape	Escape next character
1C	28	/	Cntl/←	Cursor home left
1D	29	]	Cntl/→	Cursor home right
1E	30	↑	Cntl/home	Clear line
1F	31		Shift/home	Clear page

# APPENDIX 3 SCREEN EDITOR TECHNICAL SPECIFICATION

## Interpretation of Control Codes

- 0 No action.
- 1 A line of spaces is inserted at the cursor, the cursor line and subsequent lines are shifted down. The last line is lost.
- 2 The whole cursor line including the lines of which it is a continuation (if any) and any continuation of it is deleted. Subsequent lines are shifted up, sufficient lines of spaces being inserted at the end.
- 3 Subsequent calls to INPUT will return all the characters of the page, including NEW LINEs. This mode is cancelled when all characters are returned, or by a call to OUTPUT. After returning all characters the cursor will be in the top left "home position" on the screen.
- 4 No action (this code has a special meaning in BASIC).
- 5 Subsequent calls to INPUT will return all the characters of the current line, including the NEWLINE. Cancelled as code 3.
- 6 The cursor will be displayed at all times until code 7 is entered.
- 7 The cursor will not be displayed, except during keyboard interaction, until code 6 is entered.
- 8 Move cursor left one space. This is not possible if at the top left-most position on a line.
- 9 Move cursor right at least one space and sufficiently many spaces to bring it to a screen column a multiple of 8 spaces from the start, or failing that, to the start of the next line or to the space after the continuation character on the next line.
- 10 Move cursor vertically down one space. This is not possible if on the bottom line of the screen.
- 11 Move cursor vertically up one space. This is not possible if on the top line of the screen.
- 12 Home cursor to top left of screen.

## **APPENDIX 3 SCREEN EDITOR TECHNICAL SPECIFICATION**

- 13 Move cursor to the start of the next line. In INPUT end keyboard interaction. On the bottom line scroll screen up losing top line.
- 14 Invert the top bit of all subsequent non-control codes before displaying. Subsequent characters 32 .. 127 will become 160 .. 255, 129 .. 255 will become display codes 1 .. 127. (128 will become 0 which is not displayable and is always treated as a NULL control code). This mode is cancelled only when code 15 is entered.
- 15 Cancel mode introduced by code 14.
- 16 No action — this code is reserved for use by a high resolution display editor.
- 17 Subsequent characters will be inserted at the cursor position, characters to the right on the line being scrolled right and, when appropriate, down (causing subsequent lines to be scrolled down and the bottom line lost). If the line already fills the "line image" (LIO) or the screen page (TVIO or TLIO) this is not possible. This mode is cancelled when any control code is entered.
- 18 The current screen line, if it is a continuation line, is made the start of a new line. Continuations of it are scrolled back and up if required.
- 19 The current screen line is made a continuation of the line above it. Subsequent continuations of it are scrolled down and right if necessary.
- 20 A subsequent call to INPUT will return the character at the cursor position. This mode is cancelled in the same way as code 3.
- 21 Two subsequent calls to INPUT will return the "x-y cursor address", firstly the x address which is the horizontal displacement of the cursor from the left of the screen starting at 1, then the y address which is the vertical displacement from the top of the screen, starting at 1 (irrelevant in the case of LIO). Cancelled as code 3.
- 22 The two bytes next entered are interpreted as an x-y cursor address (see code 21), and the cursor is set to the given position. This mode is cancelled when the two bytes have been received by the editor, or if a call to INPUT intervenes.

## APPENDIX 3 SCREEN EDITOR TECHNICAL SPECIFICATION

- 23 Set TV control, certain bits of the next byte entered are loaded into the video hardware "TV control register". The bits are currently used as follows:

bit	
3	1 = 8 lines/character, upper Character ROM set 0 = 10 lines/character, lower Character ROM set
1	1 = full character set 0 = half character set, top bit used to reverse character field
0	1 = black background 0 = white

This mode is cancelled in the same way as code 22.

- 24 Delete the character to the left of the cursor moving the cursor left one space. This has no effect if at the start of a line (not a continuation line) or if the character is outside the LIIO window. The rest of the line to the right of the cursor is scrolled to the left, and it and subsequent lines up if necessary.
- 25 Delete the character at the cursor. Otherwise as code 24.
- 26 Move cursor right one space, this is not possible if at the end of a line.
- 27 Put the next character directly into the display — i.e. do not treat it as a control code. This mode is cancelled after the next character has been entered, or if a call to INPUT intervenes.
- 28 Send cursor to the left-most position on this line.
- 29 Send cursor to the right-most position on this line.
- 30 Replace the current line by a line of spaces, scrolling up subsequent lines if appropriate. Send cursor to beginning of line.
- 31 Clear the whole screen to spaces and send cursor to 'home' top left.

# **APPENDIX 4**

**4**



# APPENDIX 4 BASIC RESERVED WORDS

## NEUBRAIN RESERVED WORDS

These are of 4 types:

1. Words which introduce a BASIC command.
2. Words which continue the syntax of a BASIC command. These must appear in the right context, as defined by type 1 words.
3. Functions, which return a value.
4. Symbols which represent operations or relations.

Many of the reserved words have a shorter alternative form.

Reserved word, alternative	type	context
ABS	3	
AND	4	
ACS	3	
ARC	2	
ASC	3	
ASN	3	
ATN	3	
AXES, AXE	2	PLOT
BACKGROUND, BCK	2	PLOT
BASE	2	OPTION
BREAK	2	ON
CALL	1	
CENTRE, CEN	2	PLOT
CHR\$	3	
CLEAR	1	
CLOSE	1	
COLOUR, COL	2	PLOT
CONTINUE, CONT	1	after STOP
COS	3	
DATA	1	with READ, RESTORE
DEF	1	
DEGREES, DEG	2	PLOT

## APPENDIX 4 BASIC RESERVED WORDS

Reserved word, alternative	type	context
DELETE	1	
DIM	1	
DOT	2	PLOT
DRAW, DRW	2	PLOT
DRAWBY, DBY	2	PLOT
END	1	
ERRLIN	3	after REPORT
ERRNO	3	after REPORT
ERROR	2	ON
EXP	3	
FALSE	3	
FILE\$	3	after OPEN
FILL, FIL	2	PLOT
FN	2	DEF
FOR	1	
FREE	3	
GET	1	
GO SUB, GOSUB	1, 2	also ON
GO TO, GOTO	1, 2	also ON
IF	1	
INPUT	1	
INSTR	3	
INT	3	
IN#	2	OPEN
LEFT\$	3	
LEN	3	
LET	1	
LINPUT	1	
LIST	1	
LOAD	1	
LOG	3	
MERGE	1	
MID\$	3	
MODE, MDE	2	PLOT
MOVE, MVE	2	PLOT
MOVEBY, MBY	2	PLOT

## APPENDIX 4 BASIC RESERVED WORDS

Reserved word, alternative	type	context
NEW	1	
NEXT	1, 2	FOR
NOT	4	
NUM	3	
ON	1	
OPEN	1	
OPTION	1	
OR	4	
OUT#	2	OPEN
PEEK	3	
PEN	3	
PI	3	
PLACE, PLA	2	PLOT
PLOT	1	
POKE	1	
POS	3	
PRINT	1	
PUT	1	
RADIANS, RAD	2	PLOT
RANDOMIZE	1	
RANGE, RNG	2	PLOT
READ	1	
REM	1	
REPORT	1	after ON
RESERVE	1	
RESTORE	1	with DATA
RESUME	1	after ON
RETURN, RET	1	after GOSUB
RIGHT\$	3	
RND	3	
RUN	1	
SAVE	1	
SGN	3	
SIN	3	
SQR	3	
STEP	2	FOR

## APPENDIX 4 BASIC RESERVED WORDS

Reserved word, alternative	type	context
STOP	1	
STR\$	3	
TAB	2	PRINT
TAN	3	
THEN	2	IF
TO	2	FOR
TOP	3	
TRUE	3	
TURN, TRN	2	PLOT
TURNBY, TBY	2	PLOT
VAL	3	
VERIFY	1	
WIPE, WIP	2	PLOT
#	2	OPEN, CLOSE
&	4	
(	2	
)	2	
*	4	
+	4	
-	4	
/	4	
<	4	
<=	4	
<>	4	
=	4	
>	4	
>=	4	
?	1	
↑	4	
[	2	formatter
]	2	formatter

# **APPENDIX 5**

**5**





# APPENDIX 5 LINE AND SCREEN DISPLAY CHARACTER SETS

## 5.1 LINE DISPLAY CHARACTER SET

!	"	#	\$	%	&	'	(	)	*	+	,	-
.	/	0	1	2	3	4	5	6	7	8	9	:
;	<	=	>	?	@	A	B	C	D	E	F	G
H	I	J	K	L	M	N	O	P	Q	R	S	T
U	V	W	X	Y	Z	[	\	]	↑	UNPRINTABLE	£	

# APPENDIX 5 LINE AND SCREEN DISPLAY CHARACTER SETS

## 5.2 SCREEN DISPLAY CHARACTER SETS

There are four distinct character sets which may be selected by using one of the key sequences given below. In addition, the background may be white with black characters ("black on white"), or the background may be black with white characters ("white on black").

CTRL/W A	Black on white, 8 x 10 characters, character set 1.
CTRL/W B	White on black, 8 x 10 characters, character set 2.
CTRL/W C	Black on white, 8 x 10 characters, character set 2.
CTRL/W D	White on black, 8 x 10 characters, character set 1.
CTRL/W H	White on black, 8 x 8 characters, character set 3.
CTRL/W I	Black on white, 8 x 8 characters, character set 3.
CTRL/W J	White on black, 8 x 8 characters, character set 4.
CTRL/W K	Black on white, 8 x 8 characters, character set 4.

Having selected a character set using the appropriate CONTROL/W sequence, most of the characters can be generated directly from the keyboard by means of the GRAPHICS key and the "Attribute On" mode, which may be set by typing SHIFT/↑ and cleared by typing SHIFT/ESCAPE. For the keys A-Z, (, -, ) and + the following simple rules apply:

Attribute Off:	Unshifted:	As key tops (lower case)
	Shifted:	As key tops (upper case)
	With GRAPHICS:	Characters 129–158

Attribute On:	Unshifted:	Characters 225–250
	Shifted:	Characters 193–223
	With GRAPHICS:	Characters 1–30

All 255 characters may be generated by a BASIC program, as the following example will quickly demonstrate:

```
FOR I = 1 TO 255: PUT 27, I: NEXT I
```

The value 27 is the ESCAPE code, and is included here so that the screen editor will display the next number as a character instead of obeying it as a control code (see Appendix 3 for a list of control codes).

# APPENDIX 5 LINE AND SCREEN DISPLAY CHARACTER SETS

## Keyboard Mode

The character set generated by the keyboard may be changed by typing CONTROL with a number key. This is purely a keyboard function and has nothing to do with the editor control codes. (When using a keyboard device driver these functions are not accessed by typing but instead by PUTting the appropriate byte to the driver). The character sets generated by the different keyboard modes are summarised below:

CONTROL	Unshifted keys	Shifted keys	With GRAPHICS
0	normal	upper case	characters 129–158
1	upper case (shift lock)	upper case	characters 129–158
2	normal	characters 225–254	characters 129–158
3	upper case	characters 225–254	characters 129–158
4	normal	upper case	characters 161–190
5	upper case	upper case	characters 161–190
6	normal	characters 225–254	characters 161–190
7	upper case	characters 225–254	characters 161–190
8	normal	characters 225–254	characters 193–222
9	upper case	characters 225–254	characters 193–222

## APPENDIX 5 LINE AND SCREEN DISPLAY CHARACTER SETS

0	1	2	3	4	5	6	7	128	129	130	131	132	133	134	135
8	9	10	11	12	13	14	15	136	137	138	139	140	141	142	143
16	17	18	19	20	21	22	23	144	145	146	147	148	149	150	151
24	25	26	27	28	29	30	31	152	153	154	155	156	157	158	159
!	/	#	\$	%	&	/	!	!	#	\$	%	&	/	/	/
32	33	34	35	36	37	38	39	160	161	162	163	164	165	166	167
(	)	*	+	,	-	.	/	)	*	+	,	-	.	/	/
40	41	42	43	44	45	46	47	168	169	170	171	172	173	174	175
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
48	49	50	51	52	53	54	55	176	177	178	179	180	181	182	183
8	9	:	;	<	=	>	?	8	9	:	;	<	=	>	?
56	57	58	59	60	61	62	63	184	185	186	187	188	189	190	191
@	A	B	C	D	E	F	G	@	A	B	C	D	E	F	G
64	65	66	67	68	69	70	71	192	193	194	195	196	197	198	199
H	I	J	K	L	M	N	O	H	I	J	K	L	M	N	O
72	73	74	75	76	77	78	79	200	201	202	203	204	205	206	207
P	Q	R	S	T	U	V	W	P	Q	R	S	T	U	V	W
80	81	82	83	84	85	86	87	208	209	210	211	212	213	214	215
X	Y	Z	[	\	I	↑	-	X	Y	Z	[	\	I	↑	-
88	89	90	91	92	93	94	95	216	217	218	219	220	221	222	223
~	a	b	c	d	e	f	g	~	a	b	c	d	e	f	g
96	97	98	99	100	101	102	103	224	225	226	227	228	229	230	231
h	i	j	k	l	m	n	o	h	i	j	k	l	m	n	o
104	105	106	107	108	109	110	111	232	233	234	235	236	237	238	239
p	q	r	s	t	u	v	w	p	q	r	s	t	u	v	w
112	113	114	115	116	117	118	119	240	241	242	243	244	245	246	247
x	y	z	{		}	~	█	x	z	{		}	~	█	
120	121	122	123	124	125	126	127	248	249	250	251	252	253	254	255

### CHARACTER SET 1

## APPENDIX 5 LINE AND SCREEN DISPLAY CHARACTER SETS

0	1	2	3	4	5	6	7	128	129	130	131	132	133	134	135
8	9	10	11	12	13	14	15	136	137	138	139	140	141	142	143
16	17	18	19	20	21	22	23	144	145	146	147	148	149	150	151
24	25	26	27	28	29	30	31	152	153	154	155	156	157	158	159
!	/	#	\$	%	&	/	/	Γ	α	β	ψ	δ	ε	ψ	γ
32	33	34	35	36	37	38	39	160	161	162	163	164	165	166	167
(	)	*	+	-	.	/	/	η	ι	ξ	κ	λ	ρ	ν	ο
40	41	42	43	44	45	46	47	168	169	170	171	172	173	74	175
0	1	2	3	4	5	6	7	π	Ξ	ρ	σ	τ	θ	ω	ς
48	49	50	51	52	53	54	55	176	177	178	179	180	181	182	183
8	9	:	;	<	=	>	?	χ	υ	ζ	Θ	Φ	Ψ	Λ	Ω
56	57	58	59	60	61	62	63	184	185	186	187	188	189	190	191
@	A	B	C	D	E	F	G	Σ	Δ	Π	←	↓	→	↖	↑
64	65	66	67	68	69	70	71	192	193	194	195	196	197	198	199
H	I	J	K	L	M	N	O	ȝ	ι	ȝ	ѧ	ѧ	ѧ	ѧ	ѧ
72	73	74	75	76	77	78	79	200	201	202	203	204	205	206	207
P	Q	R	S	T	U	V	W	ȝ	ȝ	ȝ	ȝ	ȝ	ȝ	ȝ	ȝ
80	81	82	83	84	85	86	87	208	209	210	211	212	213	214	215
X	Y	Z	[	\	]	↑	-	ð	ð	ð	ð	ð	ð	ð	ð
88	89	90	91	92	93	94	95	216	217	218	219	220	221	222	223
~	a	b	c	d	e	f	g	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ
96	97	98	99	100	101	102	103	224	225	226	227	228	229	230	231
h	i	j	k	l	m	n	o	ø	ø	ø	ø	ø	ø	ø	ø
104	105	106	107	108	109	110	111	232	233	234	235	236	237	238	239
p	q	r	s	t	u	v	w	ø	ø	ø	ø	ø	ø	ø	ø
112	113	114	115	116	117	118	119	240	241	242	243	244	245	246	247
X	Y	Z	(	)		)	~	█	§	€	œ	^		°	█
120	121	122	123	124	125	126	127	248	249	250	251	252	253	254	255

### CHARACTER SET 2

## APPENDIX 5 LINE AND SCREEN DISPLAY CHARACTER SETS

0	1	2	3	4	5	6	7	128	129	130	131	132	133	134	135
8	9	10	11	12	13	14	15	136	137	138	139	140	141	142	143
16	17	18	19	20	21	22	23	144	145	146	147	148	149	150	151
24	25	26	27	28	29	30	31	152	153	154	155	156	157	158	159
!	"/"	#	\$	%	&	/		!	/*	#	\$	%	&	/	
32	33	34	35	36	37	38	39	160	161	162	163	164	165	166	167
<	>	*	+	,	-	.	/	<	>	*	+	-	.	/	
40	41	42	43	44	45	46	47	168	169	170	171	172	173	174	175
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
48	49	50	51	52	53	54	55	176	177	178	179	180	181	182	183
8	9	:	;	<	=	>	?	8	9	:	;	<	=	>	?
56	57	58	59	60	61	62	63	184	185	186	187	188	189	190	191
C	A	B	C	D	E	F	G	C	A	B	C	D	E	F	G
64	65	66	67	68	69	70	71	192	193	194	195	196	197	198	199
H	I	J	K	L	M	N	O	H	I	J	K	L	M	N	O
72	73	74	75	76	77	78	79	200	201	202	203	204	205	206	207
P	Q	R	S	T	U	V	W	P	Q	R	S	T	U	V	W
80	81	82	83	84	85	86	87	208	209	210	211	212	213	214	215
X	Y	Z	[	\	I	↑	-	X	Y	Z	[	\	I	↑	-
88	89	90	91	92	93	94	95	216	217	218	219	220	221	222	223
L	+	+	+	+	+	+	+								
96	97	98	99	100	101	102	103	224	225	226	227	228	229	230	231
104	105	106	107	108	109	110	111	232	233	234	235	236	237	238	239
112	113	114	115	116	117	118	119	240	241	242	243	244	245	246	247
120	121	122	123	124	125	126	127	248	249	250	251	252	253	254	255

### CHARACTER SET 3

## **APPENDIX 5 LINE AND SCREEN DISPLAY CHARACTER SETS**

A 10x26 grid of 260 small square icons, each containing a different character or symbol. The characters are arranged in rows as follows:  
Row 0: 0, 1, 2, 3, 4, 5, 6, 7  
Row 1: 8, 9, 10, 11, 12, 13, 14, 15  
Row 2: 16, 17, 18, 19, 20, 21, 22, 23  
Row 3: 24, 25, 26, 27, 28, 29, 30, 31  
Row 4: !, //, #, \$, %, &, /  
Row 5: 32, 33, 34, 35, 36, 37, 38, 39  
Row 6: <, >, \*, +, -, ., /  
Row 7: 40, 41, 42, 43, 44, 45, 46, 47  
Row 8: 0, 1, 2, 3, 4, 5, 6, 7  
Row 9: 48, 49, 50, 51, 52, 53, 54, 55  
Row 10: 8, 9, :, ;, <, =, >, ?  
Row 11: 56, 57, 58, 59, 60, 61, 62, 63  
Row 12: C, A, B, C, D, E, F, G  
Row 13: 64, 65, 66, 67, 68, 69, 70, 71  
Row 14: H, I, J, K, L, M, N, O  
Row 15: 72, 73, 74, 75, 76, 77, 78, 79  
Row 16: P, Q, R, S, T, U, V, W  
Row 17: 80, 81, 82, 83, 84, 85, 86, 87  
Row 18: X, Y, Z, [ , \ , ] , ↑ , -  
Row 19: 88, 89, 90, 91, 92, 93, 94, 95  
Row 20: L, L, +, r, T, J, L,  $\diamond$   
Row 21: 96, 97, 98, 99, 100, 101, 102, 103  
Row 22: 104, 105, 106, 107, 108, 109, 110, 111  
Row 23: 112, 113, 114, 115, 116, 117, 118, 119  
Row 24: 120, 121, 122, 123, 124, 125, 126, 127  
Row 25: 128, 129, 130, 131, 132, 133, 134, 135  
Row 26: 136, 137, 138, 139, 140, 141, 142, 143  
Row 27: 144, 145, 146, 147, 148, 149, 150, 151  
Row 28: 152, 153, 154, 155, 156, 157, 158, 159  
Row 29: 160, 161, 162, 163, 164, 165, 166, 167  
Row 30: 168, 169, 170, 171, 172, 173, 174, 175  
Row 31: 176, 177, 178, 179, 180, 181, 182, 183  
Row 32: 184, 185, 186, 187, 188, 189, 190, 191  
Row 33: 192, 193, 194, 195, 196, 197, 198, 199  
Row 34: 200, 201, 202, 203, 204, 205, 206, 207  
Row 35: 208, 209, 210, 211, 212, 213, 214, 215  
Row 36: 216, 217, 218, 219, 220, 221, 222, 223  
Row 37: 224, 225, 226, 227, 228, 229, 230, 231  
Row 38: 232, 233, 234, 235, 236, 237, 238, 239  
Row 39: 240, 241, 242, 243, 244, 245, 246, 247  
Row 40: 248, 249, 250, 251, 252, 253, 254, 255

## CHARACTER SET 4



# APPENDIX 6

6

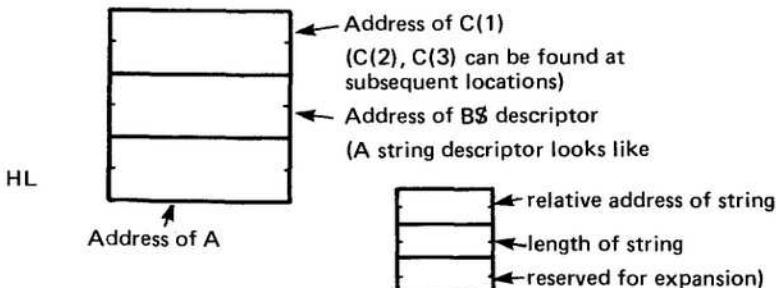




# APPENDIX 6 BASIC STATEMENT KEYWORDS

## BASIC STATEMENT KEYWORDS

- # Denotes stream identifier in input/output statements. See OPEN, INPUT, PRINT, LIST, CLOSE etc. A stream identifier is a numeric expression whose value is in the range 0..255.
- ? Synonym for PRINT (q.v.).  
Also output as the default INPUT prompt — see INPUT.
- BASE See OPTION.
- BREAK See ON.
- CALL Enter machine code sub-routine and supply pointers to BASIC variables.  
  
E.g. CALL 1000, A, B\$, C(1)  
will go to machine memory address 1000 with the Z80 HL register pointing at a six byte block of addresses.



The address is computed as in POKE (q.v.).

# APPENDIX 6 BASIC STATEMENT KEYWORDS

CLEAR	Releases memory used for variables. CLEAR by itself releases memory used by all numeric and string variables and arrays. Future references to variables will return value $\emptyset$ for numbers and null for strings; arrays can be redimensioned. CLEAR list of variables and arrays E.g. CLEAR X, A1, B\$, L\$( ), A() clears only those variables and arrays in the list.
CLOSE	Closes input/output stream. E.g. CLOSE #1 CLOSE #PR. CLOSE # $\emptyset$ , i.e. close the console, is not allowed. See OPEN # $\emptyset$ .
CONT	Synonym for CONTINUE (q.v.).
CONTINUE	Continue execution after a STOP or END statement is encountered, after an error or after the STOP key is pressed. If a radical change has been made to the program in the meantime then this may not be possible.
DATA	See READ.
DEF	Define user defined function. A user defined function may be numeric or string valued and have no arguments or a single string or numeric argument. A user defined function name must be of the form FN letter. FN letter letter, or FN letter digit, with a \$ for a string valued function. The definition must consist of a single expression. E.g. DEF FNA(X) = SQR(X↑ 2 + 1) DEF FNFS\$(A\$) = MID\$(A\$, 1, 1) DEF FNA1(A\$) = ASC(A\$)-65 DEF FNE = PEEK (8 * 256 + 7).
DELETE	Deletes program lines. DELETE by itself deletes no lines. DELETE - 100 deletes lines up to and including 100. DELETE 250 - deletes line 250 and those above it. DELETE 150 deletes line 150.

## APPENDIX 6 BASIC STATEMENT KEYWORDS

DELETE 30—150	deletes lines 30 to 150 inclusive.
DELETE —	deletes all lines.
DIM	Dimension array. An array may be string or numeric and have one or two dimensions. Several arrays may be dimensioned in one DIM statement. E.g.     DIM A\$6(15), X\$(5,20)    DIM A\$(N + 1). An array may not be redimensioned unless cleared by a CLEAR statement. If an array is referred to before a DIM statement for it has been encountered it is assumed to have dimension 10 or 10,10 and it may not be redimensioned. The base of arrays is set by OPTION BASE (q.v.). The greatest dimension allowed is limited by the amount of memory available, but in any case an array may not have more than 5461 elements.
END	End execution of program and await command.
ERROR	See ON.
FOR	Initiate for-next loop. A for-next loop has the form FOR-statement block of statements NEXT-statement. E.g.     10 FOR I = 1 TO 53 STEP. 5 20 PRINT I, SQR (I) 30 NEXT I or       FOR I = 1 TO 255: CLOSE #I: NEXT I. The block of statements is executed repeatedly while the "control variable", I in the above examples, is incremented each time by the STEP value (.5 in the first example) until the TO value (53 in the first example) is reached. If the STEP value is omitted, as in the second example, it is assumed to be 1. The first value and the TO and STEP values may be any numeric expressions, E.g.     FOR I = SIN(X) to SIN(Y) STEP (Z↑2 + 1)/2.

## APPENDIX 6 BASIC STATEMENT KEYWORDS

FOR continued . . .

The block of statements may contain "nested" for-next loops.

E.g.    10 FOR I = 1 TO 10  
      20 FOR J = 1 TO 20: PRINT A(J,J): NEXT J  
      30 A (I + 1,1) = X  
      40 FOR K = 2 TO 20  
      50 A (I + 1,K) = A(I + 1,K - 1) \* K  
      60 NEXT K  
      70 NEXT I

But the control variable in the NEXT-statement must always match that in the FOR statement, so for-next loops may not overlap.

E.g.    10 FOR I = 1 TO 10  
      20 A(I) = (X + 1)/I  
      30 FOR J = 1 TO 10  
      40 A(J) = A(I)/J  
      50 NEXT I: NEXT J

is not allowed.

The STEP value may be negative,

E.g.    FOR I = 50 TO 1 STEP -1: PRINT I: NEXT I.  
The block of statements will not be executed at all if the control variable is already past the TO value. When a for-next loop finishes the value of the control variable is the first value not used — e.g. after FOR I = 1 TO 10: NEXT I, I will equal 11.

FOR and NEXT statements may not be used immediately after THEN in IF statements (q.v.).

GET      Get single characters or bytes from an input stream.

E.g.    GET #1, A\$:    GET #3, A,B  
The stream must be open.

# APPENDIX 6 BASIC STATEMENT KEYWORDS

GOSUB	Execute a subroutine. A subroutine is a sequence statements ending with a RETURN statement. E.g.    20 GOSUB 1000 30 . . 1000 PRINT #1,A,B,C, 1010 RETURN GOSUB 1000 cause the subroutine at line 1000 to be executed, after the RETURN execution continues at line 30. See also ON.
GOTO	Continue execution at a new line number. E.g.    GOTO 60 transfers to line 60. See also ON.
IF	Execute a sequence of statements depending on a condition. E.g.    IF A\$ = "HI" PRINT "HELLO": G = 2. The sequence of statements must be on a single line. The conditional expression (A\$ = "HI" in the example) may be any numeric valued expression. E.g.    10 A = TRUE 20 FOR I = 1 TO 100 30 A = A AND X\$(I) = "Y" 40 NEXT I 50 IF A GOTO 100 60 .. .. .. 100 .  IF X = 1 THEN 100 is the same as IF X = 1 GOTO 100 IF X = 1 THEN PRINT 22 * X is the same as IF X = 1 PRINT 22 * X When the statement immediately following the condition is a LET statement with the LET keyword omitted, a THEN keyword must be used;

## APPENDIX 6 BASIC STATEMENT KEYWORDS

IF continued . . .

E.g. IF A=1 B = 2 is not allowed; but

IF A=1 THEN B = 2 and

IF A=1 LET B = 2 are allowed

The statement immediately following the condition may not be a FOR or NEXT statement.

IN # Denotes input stream identifier in OPEN statement (q.v.).

INPUT Input variables from an input stream. The stream must be open.

E.g. INPUT # KB, A\$

INPUT # 3,A,B,A\$

INPUT A1,YL(23),Z

If the stream identifier is omitted, as in the third example, stream  $\emptyset$ , the console, is assumed.

In response to INPUT a sequence of characters corresponding to a string or numeric constant is expected from the input stream. When there is more than one variable to be input by a single input statement, the corresponding constants must be separated by comma or tab characters (ASCII 44 or 09). The end of the list of constants from the input stream must be denoted by a new-line character (ASCII 13). A string constant cannot, of course, contain a new-line character. A string constant without its enclosing quotes (ASCII 34) may be supplied by the input stream but in that case it may not contain comma or tab characters, or "quote images" (repeated quotes in a string constant used to denote the presence of a quote character rather than the end of the constant). When the input stream is the console and designated thus by the omission of a stream identifier in the INPUT statement a prompt (question-mark space) is output to the console. This can be suppressed by the form

INPUT (string expression) list of variables

in which case the string expression is issued as a prompt.

E.g. INPUT ("") A issues no prompt at all.

Furthermore if an error arises in the input, input is requested again — i.e. execution is not stopped.

## APPENDIX 6 BASIC STATEMENT KEYWORDS

**LET** Assigns a value to a variable. The keyword LET may be omitted.

E.g.     LET A = 1  
          B(2,7) = Ø  
          X2 = SIN(P/14) - 1/X2  
          A\$ = "HELLO"  
          ME\$ = CHR\$(X) + MID\$(S\$,2,Y)

A numeric variable or array element may be assigned any value which is acceptable to the maths pack, this gives a larger range than allowable for a constant.

E.g.     although 99 is the largest exponent allowed in a constant, A = 2 \* 99999999E99 is allowed.

A string variable or array element may be assigned any sequence of characters or bytes up to a maximum length of 32767 (whereas a string constant cannot contain a newline character);

E.g.     A\$ = CHR\$(10) + CHR\$(13) + ME\$ + CHR\$(244) is allowed.

**LINPUT** Input a line from an input stream.

E.g.     LINPUT #SI, A\$  
          LINPUT X\$(N + 1)

Characters are collected from the stream until newline (ASCII 13) is received. The sequence of characters is assigned to the string (A\$ and X\$(N + 1) in the examples). If the stream designator is omitted then stream Ø, the console, is presumed. In this case a prompt ("?") is issued on the console stream. This prompt may be substituted for by using the form

      LINPUT (string expression) string as in INPUT.

**LIST** Output the program in ASCII text to an output stream.

E.g.     LIST #PR, 30-100  
          LIST #1  
          LIST

## APPENDIX 6 BASIC STATEMENT KEYWORDS

If the stream identifier is omitted the console stream is assumed.

LIST 10        lists line 10  
LIST 10-100    lists lines 10 to 100 inclusive  
LIST -100     lists up to and including line 100  
LIST 100-      lists from line 100 upwards  
LIST - and LIST by itself    list the entire program

LOAD	<p>Input the whole program from an input stream. The program must begin with a blank line and finish with a line containing only the character EOF (ASCII 04).</p> <p>The lines may be in any order and in interal or ASCII text format.</p> <p>E.g.    LOAD #TP</p> <p>The stream must be open.</p> <p>After loading execution stops.</p> <p>All variables are cleared by LOAD.</p> <p>The form    LOAD filename    where filename is any string expression is equivalent to</p> <p>OPEN # 128, def, 0, filename LOAD # 128 CLOSE # 128</p> <p>Here def is the default back-up store device, an operating system parameter, usually tape drive 1 or disc drive 0.</p> <p>LOAD by itself is equivalent to LOAD filename where the filename is null.</p>
MERGE	<p>The same as LOAD (q.v.), except that variables are not cleared, previous program lines are overwritten only when an input line has the same line number, and execution does not stop.</p> <p>E.g.    MERGE #1</p> <p>          MERGE "seg 2"</p> <p>Note that LOAD is equivalent to    NEW MERGE END.</p>
NEW	Delete the entire program and clear all the variables.
NEXT	See FOR.

## APPENDIX 6 BASIC STATEMENT KEYWORDS

ON	Transfer execution on a condition. E.g. ON X GOTO 100, 200, 300 If X = 1 then execution continues at line 100, if 2 at line 200, if 3 at line 300. If none of these values an error arises. The general form is ON numeric-expression GOTO line-number-list. The expression is rounded to the nearest integer. ON X GOSUB line-number-list is the same as ON . . GOTO except that the sequence of lines to which transfer is made is treated as a subroutine. When a RETURN statement is encountered transfer is made back to the statement following the ON statement. ON ERROR GOTO line-number causes control to be transferred to the given line if an error arises (instead of execution stopping as normal). This condition can be cancelled by a subsequent ON ERROR GOTO Ø. After an ON ERROR transfer execution can be resumed at the statement in which the error occurred or elsewhere by the RESUME statement (q.v.). The system function ERRNO will give the error number, the system function ERRLIN will give the number of the line in which the error occurred. The REPORT statement will cause an error message to be issued and execution to stop as in normal error handling. ON BREAK GOTO line number is the same as ON ERROR except that depressions of the STOP key ("break-ins") rather than errors are trapped. ERRNO will give the system interrupt number (Ø for the STOP key). RESUME, REPORT, ERRLIN act as in ON . . ERROR. ON BREAK GOTO Ø cancels the trapping of the STOP key. ON ERROR and ON BREAK traps should be left via RESUME or REPORT statements — executing a series of such traps without exiting in this way will cause system performance to be degraded.
OPEN	Open an input/output stream. A stream identifier is associated with a physical device and necessary parameters are passed via

## APPENDIX 6 BASIC STATEMENT KEYWORDS

the operating system to open it.

E.g. OPEN #1, 1, "file 1"

OPEN #PR, 8

OPEN #UP, 6, 9

The full form is

OPEN # stream, device, port, parameter, string. Stream is a numeric expression by the value of which the stream is referenced in subsequent input/output statements until the stream is closed. Certain devices are input or output: these may require IN#stream or OUT#stream — see device specifications (Appendix 7),

E.g. OPEN OUT #1, "file".

Device is a numeric expression by the value of which the device is known to the operating system. For instance 0 is the video keyboard editor, 1 is the tape cassette drive 1.

The device may be omitted, in which case the default back-up store device assumed by the operating system is used — usually tape drive 1 or disc drive 0.

Port is a parameter for the device and its meaning will depend on the device. In general distinct port numbers may be used to OPEN multiple copies of a device,

E.g. OPEN #1, 0, 1 will open a copy of the video keyboard editor on stream 1 (the first copy may be the console on stream 0). A Port may only be specified in an OPEN statement when a device is specified. Stream, port and device must evaluate to 0 .. 255, the value is rounded to the nearest integer.

Parameter string is a string expression the value of which is passed via the operating system to the device. It is often a filename. It may be omitted, in which case null is assumed.

A stream already open may not be opened, except the console (stream 0). OPEN #0 first closes the console (CLOSE #0 is not allowed as this would end communication with the machine) and then open the new one.

OPEN #0 by itself is not recommended as the default back-up store device is seldom a sensible console.

A device-port pair already open may not be opened — a distinct port number must be used.

# APPENDIX 6 BASIC STATEMENT KEYWORDS

OPTION	Only used in and	OPTION BASE 0 OPTION BASE 1
		Set the base of arrays (which is usually 0). OPTION BASE, if used, must precede any DIM statements or use of arrays. If the array base is 0 an array A of dimension 5 will have an element A(0), an array B of dimension 5, 5 will have elements B(0, 0), B(0, 1), ... B(0, 5) and B(1, 0), B(2, 0), ... B(5, 0). If the base is 1 then the first element of A will be A(1) and the first element of B will be B(1, 1).
OUT#		Denotes output stream identifier in OPEN statement (q.v.).
POKE	Put value into machine memory. E.g.	POKE A + B, 34 POKE BA, FNX(3) + 1
		The first argument is evaluated and rounded to an integer in the range 0 .. 65535 to give an "address" in the machine memory. The second argument is evaluated and rounded to an integer in the range 0 .. 255 and this value is put into the address. This can cause the machine to cease to function.
PRINT	Output values of expression to an output stream. E.g.	PRINT 1 + 2; PRINT #1, "HI THERE STREAM ONE", "374/88="; 374/88 If the stream is omitted, the console, stream 0 is assumed. The stream must be open. String expressions are evaluated and output as the precise sequence of characters (or bytes) which they comprise, except that if output is to stream 0 and the device on stream zero supports formatting (see device specification) a newline will be output if there is insufficient space to fit the string on the current line. Numeric expressions are output by first applying the equivalent of the STR\$ function and then outputting them as strings are output. Numeric expressions may be followed by a formatter as in STR\$. Commas or semicolons are used to separate items in a print-list. Semicolons cause no output, but commas, when output is to stream 0 and the device concerned supports

# APPENDIX 6 BASIC STATEMENT KEYWORDS

formatting, cause sufficiently many spaces to be output to position the next print-item at the start of a print-zone. On output to other devices a comma causes single tab character (ASCII 09) to be output.

E.g. PRINT , , , 19 usually causes 19 to be printed in the fourth print-zone. If a print statement does not end in a semi-colon or comma a newline will be output after all the items. The TAB function may only appear in a PRINT statement. If the output is to stream 0 and the device connected supports formatting then TAB (54), for example, moves the print head forwards, to the 54'th column. Otherwise a single tab character (ASCII 09) is output regardless of the TAB value. The general form is TAB (numeric expression) — the expression must evaluate, when rounded, to an integer in the range 0 .. 65535. This is reduced modulo the line length and the sufficiently many spaces are output to reach the desired column — this may mean moving to a new line.

**PUT** Output single characters or bytes to an output stream.  
E.g. PUT #P, (A + B + C)/2, "HELLO"  
A numeric expression is evaluated and must be rounded to a value in the range 0 .. 255.  
Only the first character of a string is output. If the string is null, zero is output.  
The stream must be open.

**RANDOMIZE** Re-initialize the random number generator to a new unknown value.

**READ** Read the next data-item.  
Data items are found in DATA statements. The items in a DATA statement are numeric and string constants (or unquoted string constants of the sort valid for INPUT inputs — see INPUT) separated by commas. After RUN the data item pointer points to the first DATA item in the first DATA statement in the program (a DATA statement, if part of a multistatement line must be the last statement on that line). As DATA items are read the pointer is advanced until all are

# APPENDIX 6 BASIC STATEMENT KEYWORDS

used. A DATA statement may not exceed 255 characters in length — the surplus will be ignored.

RESTORE sends the data pointer back to the start. RESTORE 100 sends the data pointer to the first data item in the first DATA statement in or after line 100.

E.g.      10 READ A, B, C

20 DATA 1, 2, 3

30 END

Sets      A = 1, B = 2, C = 3.

10 DIM A\$(20)

20 DATA HELLO

30 FOR I = 1 to 20: READ A\$(I)

40 RESTORE: NEXT I

Sets each element of the string array A\$ to "HELLO". In the absence of the RESTORE statement an error would arise as the data would be used up after the first read.

REM	A remark or comment. Any sequence of characters may follow REM. A REM statement must be the last in any line. E.g.      10 END : REM a very dull program.
REPORT	Print the latest error message and end execution. If there is no error it is an error to call report. See ON ERROR, ON BREAK.
RESERVE	Set aside an area of machine memory to be inaccessible to BASIC. The start address of this area will be returned by the TOP system function after a RESERVE statement has been executed. E.g.      RESERVE 1000 reserves 1000 bytes at the top of memory. The number of bytes to be reserved is evaluated as a numeric expression rounded in range 0.. 65535. This is treated as a 2's complement number, so RESERVE 65534 will be treated as a request to return 2 bytes previously reserved — this can sometimes cause loss of memory contents and a system failure.

## APPENDIX 6 BASIC STATEMENT KEYWORDS

RESTORE	See READ.
RESUME	Resume execution after an error or break-in, provided ON ERROR or ON BREAK condition is set. RESUME by itself resumes at the statement in which the error occurred or at the start of the statement before which the break-in occurred. RESUME line-number resumes execution at the start of the given line. See ON ERROR, ON BREAK.
RET	Synonym for RETURN (q.v.).
RETURN	Transfer execution to the statement following that at which the most recent GOSUB was encountered. See GOSUB, ON . . GOSUB.
RUN	Clear all variables and start execution at the lowest numbered line.
SAVE	Save program in internal format on an output stream. The stream must be open. E.g.      SAVE #3. The alternative form SAVE filename      is equivalent to OPEN # 128, def, Ø, filename SAVE # 128 CLOSE # 128 Where def is as in LOAD (q.v.).
STEP	See FOR.
STOP	End execution and print a message saying where execution ended.
THEN	See IF.
TO	See FOR.

## **APPENDIX 6 BASIC STATEMENT KEYWORDS**

<b>VERIFY</b>	The same as LOAD (q.v.) except that variables are not cleared, no program lines are loaded and execution does not necessarily stop. The program on the input stream (which must be internal format) is compared with that in memory. If there are no differences the message 'VERIFIED' is output to the console, if there are differences an error occurs.
---------------	--



# APPENDIX 7

7





# APPENDIX 7 DEVICE DRIVER SUMMARY

## NEW BRAIN DEVICE DRIVER SUMMARY

### TVIO

Number: 0

Function: Screen Editor, I/O

Ports: Ignored by the driver — may be used to make multiple copies of the device.

Parameters: *Width Height*

*Width* is "S" for short (40 character) lines,  
"L" for long (80 character) lines,  
and defaults to "S".

*Height* is an integer, between 2 and 255, the number of lines on the page, the default is 24.

Examples: OPEN #0,0,"L40"  
re-opens the console as a 80 character by 40 line screen.  
OPEN #0,0:OPEN #1,0,1  
re-opens the console as a 40 character by 24 line screen and makes another 40 character by 24 line screen on stream 1.  
OPEN #7,0,"s100"  
makes a 100 line by 40 character screen on stream 7.

### CASS1

Number: 1

Function: Tape Cassette 1, I or O.

Ports: Must be 0

Parameters: *buffer size print option filename*

*buffer size* is "\* integer" (or "\* integer:" if another parameter follows), and determines the tape buffer size as multiple of 256 bytes. The default is 4 (i.e. a 1K byte buffer).

## APPENDIX 7 DEVICE DRIVER SUMMARY

*print option* is ")" or null and defaults to null. A ")", if present, suppresses the printing of filenames on the console while searching for a file during tape OPEN IN.

*filename* is any string not beginning with "\*" or ")\"", up to 256 characters in length. It is used as a file identifier on the tape. Default is null (for input first file found).

- Examples:
- OPEN OUT #1, 1, "my file"  
records a filename "myfile" on stream 1.
  - OPEN IN #3, 1, "\*10:) my file"  
searches for a file called "my file" but does not print the names of the files found while searching. A buffer length of two and a half kilobytes is available.
  - OPEN OUT #1, "20"  
opens a file for output on stream 1 with a null filename and a buffer size of 5K.
  - records a filename "myfile" on stream 1.
  - OPEN #1, 1  
searches for a file on stream 1 and opens it for input.

### CASS2

- Number: 2
- Function: Tape Cassette 2, 1 or 0.
- Ports: Must be Ø.
- Parameters: As device 1, CASS1.
- Example: OPEN #1, 2  
Search for and open a file on stream 1 from tape cassette drive 2.

### LIOO

- Number: 3
- Function: VF display editor, I/O.

## APPENDIX 7 DEVICE DRIVER SUMMARY

**Ports:** As device  $\emptyset$ , TVIO.

**Parameters:**  $length$   
 $length$  is an integer between 16 and 254, the number of characters in the line. The default is 80.

**Examples:** OPEN # $\emptyset$ , 3  
opens an 80 character VF display as the console.  
OPEN #1, 3, 1  
opens (possibly a copy of) the VF display editor on stream 1.  
OPEN #11, 3, "150"  
opens a VF display editor for a 150 character line on stream 11.

### TLIO

**Number:** 4

**Function:** Screen editor with VF display, I/O.

**Ports:** As device  $\emptyset$ , TVIO.

**Parameters:** As device  $\emptyset$ , TVIO.

**Example:** OPEN # $\emptyset$ , 4  
Open combined screen and VF display as the console.

### KBWIO

**Number:** 5

**Function:** Keyboard input.  
(Note output to this device sets the "keyboard mode" — i.e. the way in which keys are interpreted as characters, for input from the corresponding stream).

**Ports:** May be used to create multiple copies.

## APPENDIX 7 DEVICE DRIVER SUMMARY

Parameters: None.

Example: OPEN #1, 5  
Open keyboard input on stream 1.

### KBIIIO

Number: 6

Function: Keyboard input, as device 5, KBWIO, but with immediate return — i.e. will return immediately on call to INPUT (i.e. GET) regardless of whether a key has been pressed — will return character  $\emptyset$  (NULL) if no key entered.

Ports: May be used to create multiple copies.

Parameters: None.

Example: OPEN #12, 6  
opens keyboard input, with immediate return, on stream 12.

### UPIO

Number: 7

Function: User port, I/O.

Ports: Set the Z80 hardware port address.

Parameters: None.

Examples: OPEN #1, 7, 3  
opens Z80 port 3 on stream 1.  
OPEN #25, 7, 25  
opens Z80 port 25 on stream 25.

## APPENDIX 7 DEVICE DRIVER SUMMARY

### LPIO

Number:	8
Function:	Software serial line printer output. The line printer output interprets TAB and NEWLINE codes in the conventional way.
Ports:	Irrelevant.
Parameters:	"T" <i>baud rate</i> . The "T" is optional. <i>Baud rate</i> is an integer between 75 and 19200 , and defaults 9600 . This sets the transmit baud rate.
Examples:	OPEN #8, 8 opens 9600 baud printer on stream 8. OPEN #1, 8, "110" opens 110 baud printer on stream 1.

### JGIO

Number:	9
Function:	Software serial port, I/O.
Ports:	Irrelevant.
Parameters:	"T" <i>baudrate</i> "R" <i>baud rate</i> The "T" and "R" are optional, except that the "R" is required if a receive baud rate is specified. The <i>baud rates</i> are the transmit and receive baud rates as for LPIO (device 8). They default to 9600 .
Examples:	OPEN #9,9, "110 R110" opens the software serial interface to transmit and receive at 110 baud on stream 9.

## APPENDIX 7 DEVICE DRIVER SUMMARY

**OPEN #1, 9, "4800"**

opens the serial interface on stream 1 with a transmit baud rate of 4800, and receive baud rate of 9600.

### DUMMY

**Number:** 10

**Function:** None.

**Ports:** Irrelevant.

**Parameters:** None.

**Example:** OPEN #2, 10

opens the dummy device on stream 2.

# APPENDIX 7 DEVICE DRIVER SUMMARY

## GRAPH

Number: 11

Function: High resolution screen display, shared with a screen editor.

Ports: May be used to make multiple copies.

Parameters: *linked stream width option height*

*Linked stream* is "# integer" or null and defaults to 0, i.e. the console device, and determines the stream whose display area is to be shared. The selected stream must be a screen device, whose height is sufficient to accommodate the requested height. Graphics lines require up to ten times as much memory as the character lines they replace.

*width option* is "W" or "N" (default is "W"), and determines whether the full width of the screen or a narrower part of it is to be used. Selecting "N" reduces the memory requirement by 20%.

*height* is "integer" (or ", integer" if the width option is omitted), and determines the number of graphics lines on the page. The default is 150. The height must be a multiple of 10.

Examples: OPEN#0, "s100" : OPEN#1,11, "80"

re-opens the console as a 40-character by 100 line screen, then opens a graphics stream using the full width and the lower third of that screen. The console would then be reduced to 17 character lines.

OPEN#4,0, "L254" : OPEN#1,11, "#4w229"

## APPENDIX 7 DEVICE DRIVER SUMMARY

opens a 80 character by 254 line screen as stream 4, then opens a graphics stream using the full width and all but two lines of that screen.

The width given when the linked stream was opened determines the resolution of each of the graphics line: 320 (wide) or 256 (narrow) pixels when the linked stream uses 40-character lines, 640 or 512 pixels when it uses 80-character lines.

# **APPENDIX 8**

**8**





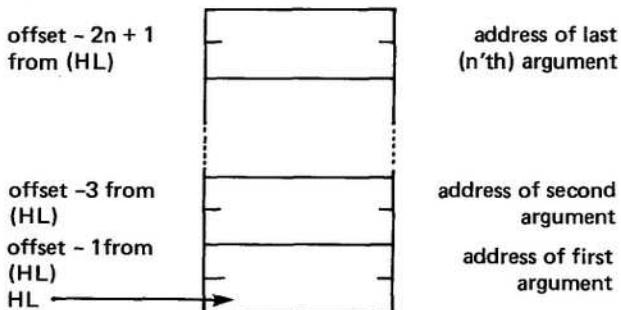
# APPENDIX 8 CALL STATEMENTS AND O/S ROUTINES

## CALL statement and O/S routines

Syntax is      **CALL expression [,argument] \***

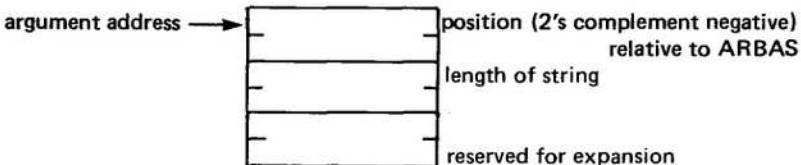
*expression* must be numeric valued and is the Z80 address which is to be called.

*argument* may be a string or number variable or array element or a numeric constant. On entry to the CALLED routine HL will point to a block of addresses of the arguments.



The C register contains a count of the arguments. The addresses are stored in the usual Z80 manner — with the low order byte at the low address.

The arguments are found at their respective addresses in their standard NewBrain formats. In the case of a numeric argument this is a six byte floating point number (such numbers may be manipulated by the maths pack). When the argument is an array successive six byte blocks will contain successive elements of the array. In case of a string argument a six-byte block containing a "string descriptor" will be found. This is

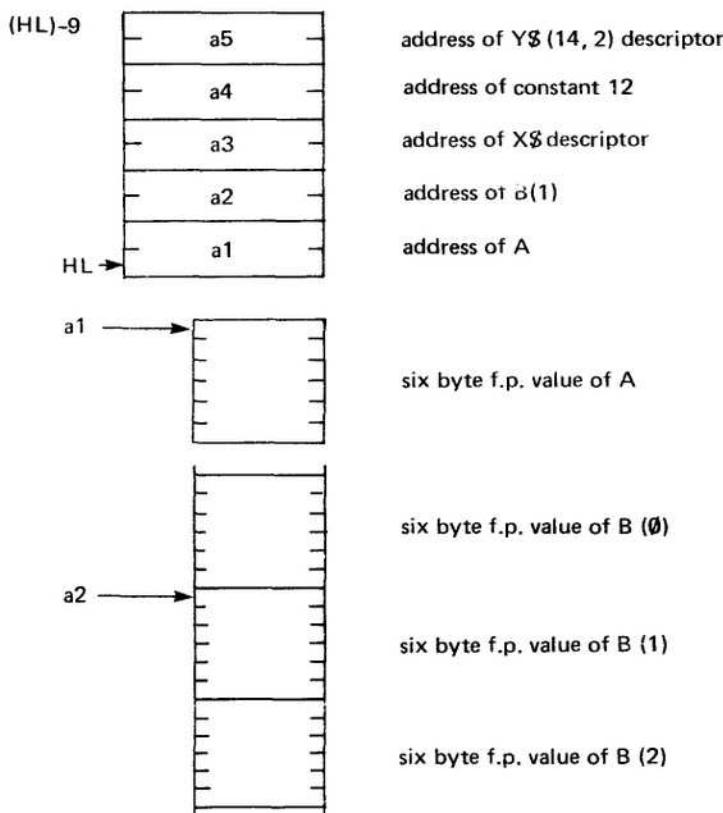


## APPENDIX 8 CALL STATEMENTS AND O/S ROUTINES

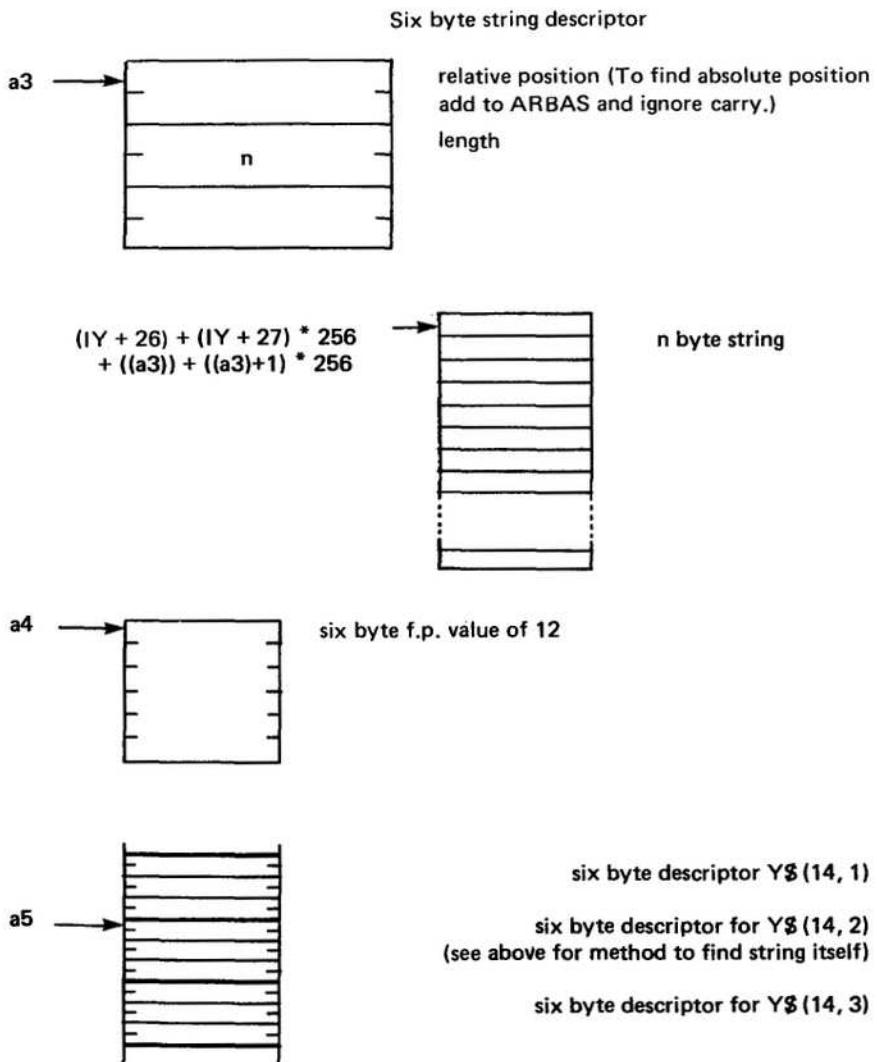
The string itself will be found at the address of the position; ARBAS (the base below which strings are stored) will be found at IY + 26, IY + 27. As with numbers, for an array successive six byte descriptors refer to successive elements of the array. Note that two dimensional array elements are stored in the order (0, 0), (0, 1), . . . (1, 0), (1, 1), (1, 2), . . .

Example: CALL 32000, A, B(1), X\$, 12, Y\$(14, 2)

will cause the Z80 program counter to be set to 32000 and the HL register will point into a block of 5 addresses (the C register will have value 5):



## APPENDIX 8 CALL STATEMENTS AND O/S ROUTINES



## APPENDIX 8 CALL STATEMENTS AND O/S ROUTINES

- Notes
- (1) A user machine code program can return to BASIC by means of the Z80 RET instruction (op-code C9H). On return to BASIC an error will be flagged if and only if the carry flag is set and then an error message will be printed with error number equal to the contents of the A register. A program returning to BASIC must preserve the IX and IY registers.
  - (2) A user program should not use the alternate register set, or the restarts.
  - (3) There is no easy way to assign to a string and change its length. The user must ensure that a string's length is correct before CALLing a user machine code program.
  - (4) Operating system routines may be called by the Z80 instruction RST 20H. The byte immediately following the restart op-code is interpreted by the operating system as a calling code and this determines which operating system routine is called.  
E.g.  
`1E00 LD E, 0 ; set stream 0  
E7 RST 20H ; call operating system  
31 DEFB 31H ; code for INPUT  
57 LD D, A ; Transfer input byte to D`  
calls the operating system routine INPUT to get a byte from stream 0 to the D register.

Useful operating system routines are as follows:

INPUT	input a byte from stream to accumulator
Entry:	Register E = stream number
Exit:	CY clear, A = byte or CY set, A = error number BCDEGHL preserved.
Action:	As BASIC GET #stream, byte
Calling code:	31H

## APPENDIX 8 CALL STATEMENTS AND O/S ROUTINES

OUTPUT	output accumulator to stream
Entry:	E = stream number A = byte
Exit:	CY clear – no error or CY set – A = error number BCDEHL preserved.
Action:	As BASIC put #stream, byte
Calling code:	30H
BRKTST	test to see if STOP key has been pressed
Entry:	—
Exit:	CY set, A = 0 if stop key pressed, CY clear otherwise BCDE preserved.
Action:	Checks if a STOP key interrupt has occurred since last check
Calling code:	36H
LDF	load floating point accumulator
Entry:	HL = address of floating point number to be loaded
Exit:	BCHL preserved
Action:	Copies number into floating point accumulator, FACC (which is inaccessible to the user)
Calling code:	2BH
STF	store floating point accumulator
Entry:	HL = address at which to store
Exit:	BCHL preserved
Action:	Copies number from FACC to the address given
Calling code:	2DH
Zero argument Maths – load FACC with value	
Entry:	No conditions
Exit:	CY clear
Action:	load FACC with floating point constant $\pi$ , 1, 0 or -1

## APPENDIX 8 CALL STATEMENTS AND O/S ROUTINES

Calling codes:	PI, 01H; FPONE, 03H; FPZER, 05H; FPMON, 04H.
One argument Maths – perform maths operation on FACC	
Entry:	No conditions.
Exit:	CY clear if operation ok, CY set if maths error,
Action:	apply floating point function to FACC. E.g. FACC := log (FACC). Functions available are absolute value, arc cosine arc sine, arc tangent, cosine, exponential function, logarithm, negative, sign (-1 for negative, 0 for zero, +1 for positive), sine, square root, tangent and integer part.
Calling codes:	ABS, 09H; ACOS, 14H; ASIN, 13H; ATAN, 0AH; COS, 0BH; EXP, 0CH; LOG, 0EH; NEG, 07H; SIGN, 0FH; SINE, 10H; SQRT, 11H; TAN, 12H; INT, 0DH.
Two arguments Maths – perform maths operation of FACC	
Entry:	DE = address of second argument .
Exit:	CY clear if operation ok, CY set if maths error.
Action:	Perform dyadic floating point operation on FACC and (DE), leaving answer in FACC. E.g. FACC:= FACC - (DE). Operations are plus, minus, times, divide, raise to power
Calling codes:	ADD, 16H; SUB, 17H; MULT, 18H; DIV, 19H; RAISE, 1AH.
INP	Convert ASCII to floating point
Entry:	DE points to an ASCII string

## APPENDIX 8 CALL STATEMENTS AND O/S ROUTINES

Exit:	DE points to first character not read. Floating point number read into FACC. CY set if and only if error
Action:	Read floating point ASCII to FACC, comparable to BASIC X = VAL(Y\$)
Calling code:	2AH.
OUT	Convert floating point to ASCII
Entry:	BC = format code
Exit:	HL will point to a string containing the ASCII equivalent of FACC. C = count of length
Action:	Convert FACC to ASCII string according to format specified in BC ( $256(64f + i) + t$ , where f = 0 for fixed point, 1 for free format and 2 for exponential format; i = digits before point, t = total digits)
Calling code:	2CH.
COMP	floating point compare.
Entry:	HL point to first floating point argument, DE to second .
Exit:	CY and Z set as for (DE) - (HL).
Action:	Compares two floating points numbers and sets flags accordingly.
Calling code:	26H.
FIX	Fix floating point number to integer.
Entry:	HL points to argument ,
Exit:	CY set if and only if error. DE = answer .
Action:	Fixes (HL) to positive binary in range 0 .. 65535 (taking integer part) .
Calling code:	27H.

## **APPENDIX 8 CALL STATEMENTS AND O/S**

<b>ROUND</b>	Round to integer
Entry:	HL points to argument .
Exit:	CY set if and only if error .
Action:	DE = answer
Calling code:	Rounds (HL) to positive binary in range Ø .. 65535.
<b>FLT</b>	Float binary number
Entry:	DE = argument
Exit:	(Answer in FACC)
Action:	Float positive 16 bit binary into FACC .
Calling code:	28H

# **APPENDIX 9**

**9**





# **APPENDIX 9 HARDWARE SPECIFICATION**

## **Model A and AD**

Z80A microprocessor running at 4MHz  
COP 420M micro controller with 1K system ROM  
32K byte RAM  
28K ROM  
Dual 1200 baud cassette ports with drive motor control  
75 ohm UHF channel 36 output  
CCITT 1v, 75 ohm composite video output  
RS232/V24 Bi-directional port  
RS232/V24 Printer port  
(Both RS232/V24 ports are software dirven and non-autonomous)

Character Generator provides 512 characters including the 96 upper and lower case ASCII/ISO printing characters, 64 viewdata mosaic graphics symbols, Western European accented characters, full Greek upper and lower case characters, line drawing graphics, games graphics and other symbols generated in 8 x 10 and 8 x 8 matrices.

Video and UHF outputs provide a display of up to 25 or 30 lines of 40 or 80 characters per line. A high resolution display of up to 250 dots vertically by 256, 320, 512 or 640 dots horizontally may be mixed with a separately scrollable character mode display.

## **Model AD**

An on-board blue-green vacuum fluorescent 16 character, 14 segment display behind a brown tinted filter.



# INDEX

ABS . . . . .	69,112,135,184
ACS . . . . .	66,112,135,184
Accuracy . . . . .	25,26,66,109
Alphabetic order . . . . .	114
Alpha numeric . . . . .	9
AND . . . . .	29,30,31,46,114,135
Argument (see also parameter) . . . . .	99,103,104,111–114,116, 179–186
Arithmetic expression . . . . .	28
Arithmetic operator . . . . .	28
Array . . . . .	28,54–55,99,100,103,110, 111,119,152,153,161
Assignment . . . . .	34,116,157
Attribute on . . . . .	8,93,129,131,142
ARC . . . . .	87,135
ASC . . . . .	80,81,112,135
ASCII . . . . .	31,80,81,109,113,114,122, 125,157,184,185
ASN . . . . .	66,112,135,184
ATN . . . . .	66,112,135,184
Auto-delete . . . . .	9,12,128
Auto-repeat . . . . .	10
AXES . . . . .	87,135
BACKGROUND . . . . .	86,135
Background . . . . .	86,90,125,132,142
BASE . . . . .	57,99,103,116,135,151
BASIC compiler . . . . .	16
BASIC definitions . . . . .	23–32
BASIC statement . . . . .	24,116
Baud rate . . . . .	105,173,189
BCK (see BACKGROUND)	
Binary . . . . .	29
Boolean Operation . . . . .	114
BREAK . . . . .	52,94–95,135,151
Break-in . . . . .	111,159,164

# INDEX

BRKTST . . . . .	183
Buffersize . . . . .	169
Byte . . . . .	63,109,111,118,132,154, 161,162,182
Byte serial . . . . .	115
CALL . . . . .	95,101,117,135,151, <b>179</b> — <b>186</b>
CASSI . . . . .	169
CASS2 . . . . .	170
Cassette recorder (see tape)	
CENTRE . . . . .	87,90,135
Channel . . . . .	17
Character . . . . .	15,16,31,35,80-81,92,109, 122,125,142-143,161,189
Character set . . . . .	15,16,92,122,125,132, <b>141</b> — <b>147</b>
Check-sum . . . . .	106
CHR\$ . . . . .	15, <b>81</b> ,112,135
CLEAR . . . . .	39, <b>56</b> ,74,94,104,119,132, 135,152,158,164
CLOSE . . . . .	20, <b>60</b> ,102,104,118,125, 135,138,152
Code 0 . . . . .	126
Colon . . . . .	24,116
COLOUR . . . . .	<b>86</b> ,135
Comma (.) . . . . .	35,100,102,109,111,116, 118,156,161-162
Command . . . . .	12,24, <b>60</b> ,109,110,116,119 121,135,153
Command mode . . . . .	24
Concatenation . . . . .	<b>31</b> , <b>76</b> ,114
Condition . . . . .	46,155
Connecting up . . . . .	2-5
Console . . . . .	4,61,92,100,109,111,117, 118, <b>121</b> ,128,152,156,158, 160,161,169,171,175

# INDEX

Constant . . . . .	<b>25–26,57,109,110,111,119</b>
	156,157
CONTinue . . . . .	<b>40–41,104,117,121,135,</b> 152
Continuation line, etc. . . . .	<b>9,13,15,128,130,131,132</b>
CONTROL key . . . . .	127,143
CONTROL/O . . . . .	<b>8,92,127</b>
CONTROL/1 . . . . .	<b>8,127</b>
Control/Escape . . . . .	129
Control/Home . . . . .	10,13,129
Control/Newline . . . . .	129
Control/W . . . . .	142
Control/→ . . . . .	10,129
Control/← . . . . .	10,129
Control . . . . .	<b>19,43–52,116–117</b>
Control characters . . . . .	9
Control code . . . . .	<b>62,109,125,126,127,128,</b> <b>129–132,142–143</b>
Control variable . . . . .	<b>44,101,120,153,154</b>
Conversion . . . . .	<b>79–82,185–186</b>
COS . . . . .	<b>66,112,135,184</b>
Current line . . . . .	<b>10,12,13,14,15,125,130,</b> 131,132,161
Cursor . . . . .	<b>9,10,11,12,13,14,15,19,</b> <b>125,127,128,130,131</b>
Cursor address . . . . .	131
Cursor control . . . . .	<b>10,11,125,129</b>
Cursor position . . . . .	61,126,127,131
DATA statement . . . . .	<b>57–58,101,102,104,119,</b> 135,137,152,162–163
Datastream . . . . .	17,115
Data structure . . . . .	<b>53–58,121</b>
DBY (see DRAWBY)	
Declaration . . . . .	116
DEF . . . . .	<b>73,103,113,116,135,152</b>
Default backup store device . . . . .	<b>94,119,158,160</b>

# INDEX

Degrees . . . . .	<b>86,89,135</b>
DELETE . . . . .	<b>101,119,136,152–153</b>
Depth . . . . .	<b>85,126,128</b>
Device . . . . .	<b>60,104,105,111,115,117,</b> <b>125,126,159,160</b>
Device driver . . . . .	<b>16,17,115,169–175</b>
Device type	
(also device number, driver number) . . . . .	<b>17,60,92,169–175</b>
DIMension statement . . . . .	<b>28,55,56,102,110,116,</b> <b>136,153</b>
Dimension . . . . .	<b>28,54,55,99,103,110,111,</b> <b>116,152,153,161</b>
Direction . . . . .	<b>117</b>
Display . . . . .	<b>9–16,35,38,39,61</b>
Divide (/)	
DOT . . . . .	<b>28,114,184</b>
DRAW . . . . .	<b>86,136</b>
DRAWBY . . . . .	<b>86,136</b>
DRW	
(see DRAW)	
DUMMY . . . . .	<b>174</b>
Dummy variable . . . . .	<b>73</b>
Editing . . . . .	<b>10–14,15</b>
Element . . . . .	<b>28,54,57,100,110,111,116,</b> <b>157</b>
END . . . . .	<b>24,39–40,120,136,152,153</b>
EOF . . . . .	<b>158</b>
Equal to (=)	
ERRLIN . . . . .	<b>29,46,114</b>
ERRNO . . . . .	<b>52,111,136,159</b>
ERROR . . . . .	<b>52,111,136,159</b>
Error code . . . . .	<b>12,13,24,32,52,94–95,</b> <b>111,122,136,153,156,159,</b> <b>163,165,182</b>
Error message . . . . .	<b>32,99–106,122</b>
	<b>32,57,58,60,99–106,122,</b> <b>159,163,182</b>

# INDEX

ESCAPE . . . . .	126,129
Execution . . . . .	39,40,41,46,121,152,158, 164,165
EXP . . . . .	67,112,136,184
Expansion box . . . . .	17,18
Exponent . . . . .	25,112,157
Expression . . . . .	24,28,29–31,100,102, 103,111
Extension . . . . .	17,94,122
FALSE . . . . .	29,30,82,110,111,112,136
FILE\$ . . . . .	111,136
File (see also TAPE) . . . . .	18,19,106,170
Filename (see also Title) . . . . .	119,158,160,164,169–170
FILL . . . . .	87,105,136
Floating point . . . . .	25,109,179,181,183–186
FN . . . . .	27,73,103,113,136
FOR–NEXT . . . . .	11,12,15,44–46,101,102, 119–120,136,137,138,153
Format . . . . .	26,35,109,112,117–118, 161,162
Format specification . . . . .	21,112,117–118
Formatter . . . . .	21,35,103,112,117–118, 138,161
FREE . . . . .	111,136
Free format . . . . .	112,117–118
Function . . . . .	65–74,75–78,79–82,85, 89,99,111–114,116,135
GET . . . . .	62–63,118,136,154,172
GOSUB . . . . .	48–49,100,116,122,136, 137,155,159
GOTO . . . . .	39–40,47,100,104,117, 136,155,159
GRAPH . . . . .	175

# INDEX

GRAPHICS key . . . . .	10,127, <b>142—143</b>
GRAPHICS/↑ . . . . .	10,13,14,129
GRAPHICS/↓ . . . . .	10,14,129
Graphics . . . . .	<b>83—90</b> ,118,175
Graphics characters . . . . .	See mosaic characters
Graphics pen . . . . .	<b>85</b> ,90
Greater than (>) . . . . .	29,46,114
Greater than or equal to (>=) . . . . .	29,46,114
Height . . . . .	105,169,175
High resolution display . . . . .	<b>85</b> ,131,175, <b>189</b>
Help in emergency . . . . .	<b>91—95</b>
HOME . . . . .	<b>10</b> ,11,13,129,130,131,132
IF . . . . .	<b>27</b> , <b>46—47</b> ,116,136,155
IF — THEN . . . . .	<b>46—47</b> ,116,136,138,155
Increment . . . . .	<b>24</b> , <b>44</b> ,120
IN# . . . . .	117,136,156
INPUT . . . . .	<b>37—38</b> ,100,101,102,109, 117,125,127,128,130,131, 132,156,169,170,171,172, 182
Input/Output . . . . .	<b>16</b> , <b>59—64</b> ,115,151,169, 170,171,172,173
Input—Output system . . . . .	<b>17</b> ,125
INSERT . . . . .	<b>10</b> , <b>11</b> ,12,13,14,129
INSTR . . . . .	<b>78</b> ,113,136
INT . . . . .	<b>69</b> ,112,136,184
Integer . . . . .	<b>25</b> ,112,185—186
Integer format . . . . .	109,112, <b>117</b>
Internal code (see also character set) . . . . .	80, <b>81</b> ,112
Internal reader . . . . .	119
Interruption . . . . .	<b>18</b> ,92,116,183
Intrinsic function . . . . .	<b>65—74</b>
JG10 . . . . .	<b>173</b>

# INDEX

KB110 .....	172
KBW10 .....	171
Keyboard .....	7–8,16,37,62,125,127, 128,130,131,143,160,171, 172
Keyword (see also reserved word) .....	24,61,100,102,109,116, 151
LEFT\$ .....	77,113,136
LEN .....	76,112,136
Length .....	76,109,112,126,128,157, 162,171
Less than (<) .....	29,46,114
Less than or equal to (≤) .....	29,46,114
LET .....	27,34,47,116,136,155,157
L110 .....	126,128,131,132,170
Line display .....	14–15,39,61
Line editor .....	14,170
Line number .....	12,21,24,99,100,104,111, 116,121,155,158,164
Linked stream .....	85,175
LINPUT .....	61,104,109,118,136,157
LIST .....	21,38–39,63,101,102,118, 136,157–158
LOAD .....	18,24,63,119,136,158
LOG .....	67,112,136,184
Logical expression .....	29,110
Logical operator .....	20,46,110
Lower case .....	8,27,122,125,127,142,143
LP10 .....	173
Machine code .....	95,117,119,151,182
Mains power .....	3,105
Mantissa .....	25,109,112,118
MBY (see MOVEBY)	

# INDEX

<b>MDE</b>	
(see MODE)	
<b>Memory</b>	94
<b>Memory space (storage)</b>	55,56,85,94,99,104,105, 109,110,122,125,152,153, 163,175
<b>MERGE</b>	119,136,158
<b>MID\$</b>	77,113,136
<b>Minus (-)</b>	28,113,114,184
<b>Monitor</b>	4,9
<b>MODE</b>	82,136
<b>MOVE</b>	86,136
<b>MOVEBY</b>	86,136
<b>Mosaic characters, etc.</b>	
(see also character set)	84,109,122,125,189
<b>Multiple copies</b>	122,129,160,169-175
<b>Multi-statement line</b>	
(see also colon)	57,162
<b>MVE</b>	
(see MOVE)	
<b>NEG</b>	184
<b>NEW</b>	117,137,158
<b>NEWLINE</b>	10,11,12,13,109,118,125, 127,128,129,130,156,157, 161,162,173
<b>NEXT</b>	44-46,101,102,119,137, 154,158
<b>NOT</b>	29,31,46,114,137
<b>Not equal to (&lt;&gt;)</b>	29,46,114
<b>Null string</b>	38,40
<b>NUM</b>	82,109,112,137
<b>Number</b>	25,26,28,82,109
<b>Numeric constant</b>	25-26,57,109

# INDEX

Object code . . . . .	121
ON . . . . .	27,50–51,52,53,94–95, 99,100,135,136,137, <b>159</b>
ON BREAK . . . . .	52,94–95,104,116,119, 121,159,164
ON ERROR . . . . .	<b>52</b> , 94–95,104,116,119, 159,164
ON-GOSUB . . . . .	<b>50</b> – <b>51</b> ,116,159
ON-GOTO . . . . .	<b>50</b> – <b>51</b> ,116,159
OPEN . . . . .	14,15,17,20,21,60,93,94, 99,111,126,136,137,138, <b>159</b> – <b>160</b>
OPEN IN . . . . .	125
OPEN OUT . . . . .	125
Operating system . . . . .	<b>16</b> ,115,117,122,160
OPTION . . . . .	<b>57</b> ,99,103,116,135,137, 161
OR . . . . .	27,29, <b>30</b> ,31,46,114,137
OUT # . . . . .	<b>117</b> , <b>161</b>
Output . . . . .	17,21,26, <b>54</b> – <b>59</b> ,109,117, 125,128,130,169,170,173
Page . . . . .	125
Paged memory . . . . .	122
Parameter (see also argument) . . . . .	<b>24</b> ,38,45, <b>74</b> ,78,86,89,101, 159
Parameter string . . . . .	<b>105</b> ,106,111, <b>117</b> ,160,169– 176
Parenthesis . . . . .	<b>31</b> ,102,111,122
PEEK . . . . .	<b>112</b> ,137
PEN . . . . .	<b>89</b> ,105,112,137
Pen colour . . . . .	85,86
Peripheral . . . . .	16,17
PI . . . . .	26,27, <b>66</b> ,111,137,184
Pixel . . . . .	176
PLACE . . . . .	<b>86</b> .137

# INDEX

PLOT . . . . .	61,85–89,105,118,135–138, 161
Plus (+) . . . . .	28,113,114,184
POKE . . . . .	95,101,119,137,161
POS . . . . .	111,137
Port . . . . .	17,62,104,106,117,126, 160,189
Power . . . . .	3,28,31,68–69
Precedence of operators . . . . .	31
PRINT . . . . .	12,15,21,26,32,34,35–36, 100,109,117,137,151,161– 162,173
Printer . . . . .	5,21,62,189
Printhead . . . . .	111,118,162
Print option . . . . .	169–170
Print zone . . . . .	35,36,162
Processor . . . . .	17,18,189
Program . . . . .	5,10,12,18,19,21,24,41, 109,110,115,118,119,120, 121,158,164
Program execution (see Execution)	
Program testing . . . . .	40,41,122
Prompt . . . . .	37,100,117,118,127,156, 157
Prompt expression . . . . .	38,49,61,156,157
PUT . . . . .	62–63,104,118,137,143
Quotation mark . . . . .	102,109,110,156
RADIANS . . . . .	86,89,137
Radian . . . . .	66,86
Raise to a power ( $\uparrow$ ) . . . . .	28,31,68–69,114,184
RANDOMIZE . . . . .	71,119,137,162
Random access . . . . .	115
Random number . . . . .	71–73,111,119,162
RANGE . . . . .	87–90,137

# INDEX

Range . . . . .	118
READ . . . . .	57–58,101,119,135,137, 162
Relational operator . . . . .	29,46,114
REM . . . . .	41,119,122,137,163
REPEAT Key . . . . .	9,13
REPORT . . . . .	51–52,119,136,137,159, 163
RESERVE . . . . .	95,119,137,163
Reserved word . . . . .	26,27,121,135–138
RESTORE . . . . .	57–58,119,135,137,163, 164
RESUME . . . . .	52,116,137,159,164
RETurn . . . . .	48–49,100,116,137,155, 159,164
Reverse field . . . . .	125,126,132
RIGHT\$ . . . . .	78,113,137
RND . . . . .	71,111,137
RNG (See RANGE)	
RUN . . . . .	12,39–40,117,137,162,164
SAVE . . . . .	19,63,118,137,164
Scientific format, notation . . . . .	25,26,109,112,117–8
Screen . . . . .	9–14,35,38,105
Screen display . . . . .	9–14,35,38
Screen editor . . . . .	10,109,125–132,169,171, 175
Scroll . . . . .	12,13,125,130,131,132,189
Semi-colon (;) . . . . .	35,100,118,161–162
SENSI . . . . .	5
SENS2 . . . . .	5
Serial device, port . . . . .	105,106,115,173
SGN . . . . .	70–71,112,137,184
SHIFT key . . . . .	8–10,127
SHIFT/ESCAPE . . . . .	8,93,129,142
SHIFT/HOME . . . . .	10,11,81,129

# INDEX

- SHIFT/INSERT ..... 10,12,129  
SHIFT/ $\rightarrow$  ..... 10,11,129  
SHIFT/ $\leftarrow$  ..... 10,11,129  
SHIFT/ $\downarrow$  ..... 10,13,129  
SHIFT/ $\uparrow$  ..... 8,84,142  
Shift Lock ..... 8,93,127,143  
SIN ..... 66,112,137,184  
SQR ..... 28,68,112,137,184  
STEP ..... 44,45,101,120,137,  
154,164  
STOP ..... 18,19,24,40–41,  
52,92,117,121,135,  
138,152,158,164,183  
STR\$ ..... 82,109,112,138,161  
Stream ..... 17,19,20,21,60,85,92,  
94,104,105,115,117,  
151,152,154,156,158,  
160,161,162,165,183  
Stream number ..... 61,86,89  
String ..... 27,31,35,87,101,109,  
112,114,118,184  
String constant ..... 57,102,109,110,122,  
162  
String descriptor ..... 179,181  
String expression ..... 31,117  
String handling ..... 75–78  
String variable ..... 35,110  
Subroutine ..... 41,48,49,151,155  
Subscript  
(see also Array Element) ..... 54,99,100  
Substring ..... 113  
Switching on ..... 6  
Switch setting ..... 5  
TAB ..... 36–37,45,101,102,  
109,117–118,138,156,  
162,173

# INDEX

TAN .....	66,112,138,184
Tape .....	5,18-20,160,169, 189
TAPE1 .....	18
TAPE2 .....	19-20
Tape error .....	105,106
Tape recorder (see Tape)	
TBY	
(see TURNBY)	
Television .....	4,9
Test plotting .....	87
THEN (see IF)	
Title .....	18,19
Times (*) .....	28,114,184
TL10 .....	126,128,131,171
TO .....	27,44-46,101,120, 138,154,164
Token .....	63,118,119,121
Tone control .....	5
TOP .....	95,111,138,163
TRN	
see TURN	
Trap .....	52,116,121,122,159
Trigonometric function .....	66,112
TRUE .....	29,30,82,110,111,112
Truth tables .....	29,30
TURN .....	86,138
TURNBY .....	86,138
TV control .....	17,84,132
TV10 .....	126,128,131,169
TV mode .....	125
UP10 .....	172
Upper Case .....	8,27,122,125,127, 142,143

# INDEX

User defined functions . . . . .	<b>73—74,103,113,116,</b> 152
User port . . . . .	<b>17,172</b>
VAL . . . . .	<b>82,104,112,138,185</b>
Variable . . . . .	<b>27,28,35,39,41,44,55,</b> 73,100,104,109,110, 111,116,119,151,152, 157
VERIFY . . . . .	<b>19,63,104,119,138,</b> 165
Viewdata graphics . . . . .	See mosaic characters
Volume control . . . . .	<b>5</b>
Width . . . . .	<b>169,175</b>
Window . . . . .	<b>14,15,125,126,132</b>
WIPE . . . . .	<b>86,138</b>
X10 . . . . .	<b>125</b>
Z80 . . . . .	<b>17,172,179,180,</b> 182,189





**Grundy Business Systems Ltd.**

**Sales and Administration**  
Somerset Road, Teddington,  
Middlesex TW11 8TD  
Tel: 01-977 1171

**Marketing and R&D**  
Science Park, Milton Road,  
Cambridge CB4 4BH  
Tel: 0223 350355