POLYBASIC MANUAL

polycorp

POLYBASIC MANUAL

POLYCORP

Holdings Ltd

Development Finance Centre, Cnr Featherston & Grey Streets, Wellington.
Telephone 737-081. Telex 31351. PO Box 3090.

CONTENTS

1. THE POLY SYSTEM

POLYBASIC incorporates all standard BASIC commands as well as extensions to control the special features of the Poly System such as multiple screens, colour and graphics. The POLYBASIC manual is a reference manual which describes the Poly System and each command.

The Poly System may be set up either as independent Poly units or as a system with all the Poly units connected to a shared disk.

1.1. STANDALONE MODE

A Poly unit is operated either in Standalone mode or in System mode.

In STANDALONE MODE, the Poly is not connected to other units or any disk Unit. Only a restricted POLYBASIC is available. This contains most of the commands available in the full POLYBASIC but all disk handling and a few other commands are omitted. In this mode, Tape cassettes are used for storing programs.

1.2. SYSTEM MODE

In SYSTEM MODE, each of the Poly units is connected via a communications line to a central disk unit on which all programs are stored. Tape cassettes may be used as well. All commands and statements are available in System Mode.

1.3. THE POLY KEYBOARD

The Poly keyboard has been specially designed to handle the special requirements of educational computing. The lefthand part of the keyboard contains a standard "QWERTY" keyboard with special keys added on the right hand side.

The special character keys may either be used in programs using their ASCII decimal value or they may be assigned a special function using the ON KEY command which causes an interrupt to the program whenever the key is pressed. A subroutine is then executed similar to an ON ERROR subroutine.

Special keys include:-

1. Cursor control keys. These are used to move the cursor around the screen. These keys may be used to edit programs (see Editing POLYBASIC) or may be programmed.

2. The character insert and delete keys are also used to edit source files or may programmed.

3. The line insert and delete keys may be programmed to carry out these functions.

4. The <CALC> key when pressed interrupts the program currently running, and allows calculations to be made on the bottom line of the screen.

5. The <HELP> key interrupts the program currently running and displays the required information from a disk file.

6. The <PAUSE> key is used to temporarily stop the operation of a POLYBASIC program or a listing when using either the LIST or +LIST command. Operation is restarted by pressing any key. If the <PAUSE> or the <SPACE> bar is pressed, the program lines are executed or listed one by one.

7. The <EXIT> key stops the program executing. Execution may be restarted from where the program left off by typing the command CONT <ENTER>. In most teaching modules the <EXIT> key is trapped and causes the MENU program to be executed.

8. The <NEXT>, <BACK> and <REPEAT> keys are used extensively in the teaching modules and may be programmed as required.

9. The numeric keypad usually returns the digits 0 to 9 but may also be programmed for special functions using ON KEY as well as being able to be set to return any ASCII value.

1.4. THE POLY SCREENS

The POLY system has the unique feature of multiple screens. These may be compared to a series of transparencies one behind the other. These screens consist of two text screens, two graphics screens and a half intensity background screen. The individual screens may be switched ON or OFF as required, and have a fixed display priority of 1, 2, 3, 4, Background. Screens 1 and 3 are the text screens and 2 and 4 are the graphics screens.

The text screens contain 24 rows of 40 characters, the rows being numbered 0 to 23 from the top of the screen, and the columns being numbered 0 to 39 from the left hand side.

The Teletext character generator is used to display on the text screens and the Teletext control character conventions are therefore followed.

Control characters are printed onto the screen to achieve colour, flashing, double height, and chunky graphics. On the text screens, the chunky graphics have six blocks per character position and may be set by using special control characters followed by the printing of one of the characters which has a

chunky graphics equivalent. Individual chunks may be set using SET and RESET.

The text screens may be split at any row so that the upper and lower portions may be used independently, each with its own scrolling. Alternatively, the scrolling may be turned off. Specific portions of the screen may be printed using PRINT@.

The graphic screens each contain 204 rows of 240 pixels, the rows being numbered 0 to 203, and the columns 0 to 239.

Each graphics screen is capable of displaying four colours at a time. These are either:

	Red	Blue	Green	White
or	Red	Magenta	Yellow	White
or	Magenta	Blue	Cyan	White
or	Yellow	Cyan	Green	White

These are set using the COLOUR and MIX commands.

Finer graphics may be achieved by selecting screen 5. This uses the graphics screens 2 and 4 and gives a pixel definition of 204 rows of 480 pixels. Any of the 4 colour sets given above are available. The 480 graphics screen has a priority similar to screen 2.

As well as using SET and RESET for displaying particular points on the screen, LINE, DRAW, DRAW@, FILL and FILL@ provide simple means of displaying more complicated graphics.

1.5. SWITCHING ON

The operation manual describes the plugging in and switching on procedures. When the system is connected up, each Poly unit is switched on using the switch on the back of the unit. The magenta LOGON screen immediately appears. In Standalone mode, the user types in his initials and presses <ENTER>. The BASIC programming mode is then entered.

In the System Mode, the user types in his initials and a password. These determine the security level for file access. A menu then appears, the contents of which depend on the disk currently in use.

Should STANDALONE BASIC appear, type LOGOFF, press <ENTER> and wait 15 seconds before attempting to LOGON again.

On the menu, the magenta flashing square is moved to the required box and <ENTER> is pressed.

2. PROGRAMMING IN BASIC

POLYBASIC commands and programs may be entered and run whenever the yellow prompt Ready or the cursor appears.

2.1. IMMEDIATE MODE

BASIC commands and statements when entered from the keyboard without a line number, are executed immediately the <ENTER> key is pressed.
For Example:-

```
PRINT 24 * 6
```

prints the answer 144 on the screen.
This statement has been executed in Immediate Mode.

2.2. PROGRAM MODE

When a statement is preceded by a line number it becomes part of a program as soon as the <ENTER> key is pressed, and is not executed until a RUN is requested.
For Example:-

```
10 PRINT 24 * 6
```

does not print out the answer until RUN is typed in, followed by <ENTER>.

A program is executed in Program Mode.

2.3. CONVENTIONS

a. In programs, each line may contain several statements, as long as they are separated by a colon (:).

b. The maximum length of any line, including the line number, is 255 characters when it is in the compiled form. Line numbers must be between 1 and 32767. A POLYBASIC program starts execution at the lowest line number.

c. When writing programs, the numbers should be allocated in steps of 10 or more, to allow later insertion of new lines. Spacing within lines makes them easier to read.

2.4. ENTERING and EDITING POLYBASIC Programs

To INSERT a line into an existing program, type in the line with a line number between those in the program, and press <ENTER>. POLYBASIC inserts it into the correct position.

To DELETE a line, type in the line number of the line to be deleted, and press <ENTER>.

To ALTER a line, LIST the line on the screen, make the required alteration using the <- and -> cursor keys and the character insert and delete keys, move the cursor to the end of line being altered and press <ENTER>.

The program lines already entered can be looked at by using the LIST command.

2.5. COMPILED POLYBASIC

A POLYBASIC program has two forms - source and compiled. These are stored on disk with the file extensions .BAS and .BAC respectively. The compiled form is usually used for execution as it is more efficient and being in an encoded form which cannot be listed, gives some security to the program.

2.6. STORAGE AND RETRIEVAL OF PROGRAMS

To SAVE a program on disk type

SAVE"filename" <ENTER>

TO LOAD A BASIC program from disk, whether in source or compiled form, type

LOAD"filename" <ENTER>

To COMPILE a source program on disk, type

COMPILE"filename" <ENTER>

To RUN a program, whether it is stored in source or compiled form, type

RUN"filename" <ENTER>

To save a program on cassette, ready the cassette for recording it, and type

CSAVE "filename" <ENTER>

Only source files may be stored on cassette.To LOAD a program from cassette, type

CLOAD "filename" <ENTER>

If the filename is omitted, then the next file on the tape is loaded.

2.7. VARIABLES

A variable is a "box" into which different values may be placed. Each "box" (or variable) is assigned a name, and the contents later retrieved or changed by using this name.
For Example:-

If a variable is named BOX then the value 66 is placed in BOX by the statement

LET BOX = 66

or merely by

BOX = 66

Variable names must begin with a letter (A-Z) and may be followed by other letters or numbers. Variable names may be any length. Variables containing lower case letters are different from those only in upper case. The following are valid variable names:-

A, A8, NAME, P1234, address, Sub0

Care must be taken not to choose variable names which are used within commands. These are RESERVED WORDS. By choosing meaningful variable names, programs are more eaily understood.

There are 3 types of variables in POLYBASIC: INTEGER, FLOATING POINT and STRING. The first two types are used to store numeric values with differing degrees of precision; a String stores strings of characters - letters, blanks, numbers and symbols.

2.7.1. Floating Point Variables

Floating Point Variable names are as set out above. These variables are used for holding either numbers containing decimal points, or numbers too large or too small to hold in Integer variables. Very large or very small numbers are converted to Scientific notation. The number of digits displayed can be set by the DIGITS command. Unless reset, 6 digits before and after a decimal point are printed in decimal notation.
For Example:-

-0.00000123 is displayed as -1.23E+06

Examples of Floating Point variable names are:-

AB, X1, X, VALUE2, SummedTotal, n1549

Examples of valid Floating Point values are:-

-3.2, 79.1, 1E-06, 32768

Each Floating Point variable uses 8 bytes of memory.

2.7.2. Integer Variables

Integer variable names are made up of a Floating Point variable name followed by a % sign. Integer variables may only contain whole numbers in the range -32768 to 32767. If an integer variable is assigned a value in the range 32768 to 65536, it is converted to its twos complement by subtracting 65536 from it. This allows address values in this range to be handled by integer variables.

Where possible, use Integer variables rather than Floating Point variables as they use less memory space and allow the program to run faster.

Examples of Integer variable names are:-

A%, X2%, ab%, GROSS%, Tax1982%

Examples of valid Integer values are:-

27, -201, 32767

Each Integer variable uses 2 bytes of memory.

2.7.3. String Variables

String variable names are made up of Floating Point variable names followed by a $ sign. Up to 65535 characters, numbers and symbols may be placed in any one String variable, provided sufficient memory is available. The function FRE(-1) indicates the size of the largest string which may be allocated.

Examples of valid string variable names are:-

Ab$, A$, X1$, address$

Examples of valid String values are:-

```
A$="This is a string literal"
B$=""
C$= CHR$(3)+"Yellow"+CHR$(1)+"Red"
D$="12345 is part of a string"
E$=" cYellow aRed"
```

Each String variable uses 4 bytes of memory plus the string.

2.7.4. Tables or Arrays

Tables of values may be held.
For Example:-

A table of 5 integers are held in an array. The items in this array are referenced as A%(0), A%(1), A%(2), A%(3) and A%(4). The index of the first item in any array is always zero.

The size of the array must always be defined in a DIM statement.
For Example:-

```
DIM ARRAY%(10)
```

defines an array containing 11 integer values, ARRAY%(0) to ARRAY%(10).

Multi-dimensional arrays may also be set up and are referenced within the brackets by row and column.
For Example:-

```
DIM BRANCH$(2,4)
```

sets up a two dimensional string array containing 3 rows (rows 0, 1 and 2) and 5 columns (columns 0, 1, 2, 3, and 4).

BRANCH$(1,2) is the string held in row 1, column 2:-

	Columns				
	0	1	2	3	4
ROW 0	x	x	x	x	x
ROW 1	x	x	0	x	x
ROW 2	x	x	x	x	x

Arrays may have any number of dimensions.
For Example:-

```
DIM MULTI(4,4,4,4)
```

dimensions a 4 four dimension floating point array with 5 items in each dimension.

Note that this would take 5 * 5 * 5 * 5 * 8 = 5000 bytes of memory. Care must therefore be taken to ensure it will fit into available memory.

Variable names within each type of variable must be unique, but the same variable names may be used for each type of variable. Lower case and upper case are also different.
For Example:-

```
FINAL, FINAL%, FINAL$, final$, Final and FINAL$(6)
```

are all different variables.

2.8. LITERALS AND CONSTANTS

2.8.1. String Literals

Any string of characters enclosed in double quotation marks is a string literal. A null string is written as "".
For Example:-

```
PRINT"ABCDEF1234"
```

displays ABCDEF1234 on the screen.

Poly uses the Teletext conventions for specifying control characters. Within strings, these may be represented by a ↑ followed by the character of the control character where:-

```
↑A or ↑a = 1
↑B or ↑b = 2
.
.
.
↑Z or ↑z = 26
```

To print Hello in Red, the string may be written either as

```
PRINT CHR$(1);"Hello"
```

or as `PRINT "↑AHello"`

or as `? "↑aHello"`

2.8.2. Numeric Constants

Numeric constants are stored in either Integer or Floating Point form. Integers are stored in 2 bytes, Floating Point numbers in 8 bytes. Some examples of valid integer constants are:-

```
32767
    2
   -7
```

Some examples of valid floating point constants are:-

```
  32768    Too big for integer
    1.2    Decimal point
-3.4E+8    Scientific notation
```

2.9. TYPES OF OPERATORS

2.9.1. Arithmetic Operators

These are:-

SYMBOL	MEANING	EXAMPLE	MEANING	
+	Add	6 + 2	Add 6 and 2	8
-	Subtract	6 - 2	Subtract 2 from 6	4
*	Multiply	6 * 2	Multiply 6 by 2	12
/	Divide	7 / 2	Divide 7 by 2	3.5
MOD	Remainder (Modules)	7 MOD 3	The integer remainder when 7 is divided by 3	1
DIV	Integer Division	7 DIV 3	The integer result of 7/3	2
↑	Exponentiation (The <EXP> key is used to enter the ↑ symbol)	6 ↑ 2.1	6 to the power of 2.1	43.064

When an arithmetic expression containing several of the above symbols is evaluated, it is processed in the reverse order to that shown in the list above, that is exponentiation first, followed by multiplication and division, (including MOD and DIV) and addition and subtraction last. Where there is equal priority, an expression is processed from left to right.
For Example:-

```
6 + 4 * 2 - 9 / 2↑2
    =  6 + 4 * 2 - 9 / 4      (exponentiation evaluated)
    =  6 + 8 - 2.25           (* and / evaluated)
    =      11.75              (+ and - evaluated)
```

Brackets may be added to alter the order as expressions within brackets are evaluated first.
For Example:-

```
(6 + 4) * 2 - (8 / 2)↑2
    =  10 * 2 - 4↑2
    =      20 - 16
    =         4
```

If the result is in the range -32768 to 32767 then the result will be integer. It will be converted to floating point if one of the numbers or the result contains a decimal point or is outside the integer range.
For Example:-

```
5 / 2 = 2.5
```

gives a floating point result.

2.9.2. Relational Operators

These allow the testing of the relationship between a variable and other variables or constants.

The relational operators recognized in POLYBASIC are:-

SYMBOL	MEANING	EXAMPLE	MEANING
=	Equal	X = 4	X is equal to 4
<>	Not equal	X <> 4	X is not equal to 4
<	Less than	X < 4	X is less than 4
>	Greater than	X > 4	X is greater than 4
<=	Less than or equal	X <= 4	X is less than or equal to 4
>=	Greater than or equal	X >= 4	X is greater than or equal to 4

For Example:-

```
3 + 2 > 4
```

is TRUE.

See also the section on String Operators.

A TRUE result has the value -1 while a FALSE result has the value 0.
Thus:-

```
PRINT 8>2
```

prints the result as -1 as it is TRUE.

It is sometimes useful to set up variables TRUE = -1 and FALSE = 0 which can be used later in the program for setting and testing variables and expressions.

2.9.3. Logical Operators

These allow combination of relationships.

SYMBOL	EXAMPLE	MEANING
AND	X > 4 AND X < 10	X is greater than 4 AND X is less than 10
OR	X = 4 OR X = 10	X is equal to either 4 OR 10
NOT	X = 4 AND NOT Y = 3	X is equal to 4 and Y is not equal to 3

Logical operations are performed <u>after</u> the arithmetical and relational operations.

2.9.4. String Operators

+ may be used to join (or concatenate) two strings together.
For Example:-

```
B$ = "XXXX"

A$ = B$ + "ABC"
```

After this operation, A$ has the value "XXXXABC".

The relational operators may be used to compare strings.
For Example:-

```
IF A$ < "M" THEN 800
```

X$ < A$ means that X$ would precede A$ if the contents were sorted into alphabetical order, "=" means the strings are identical, and ">" means "comes after" when sorted alphabetically.

2.10. TYPES OF INSTRUCTIONS

The Poly unit may be given instructions in various forms. The reserved words used in these instructions may not be written with spaces embedded within them. These may be classified into:

2.10.1. Statements

STATEMENTS are instructions to the computer which may generally be used either within a program in Program Mode, or in Immediate Mode.
For Example:-

```
PRINT@(10,5)"This is my new program"
```

2.10.2. Functions

FUNCTIONS are a special type of instruction which carry out special purpose actions. Functions all return a value.
For Example:-

```
X1 = SQR(25) + 2
```

The function SQR carries out the operation SQUARE ROOT and the answer of 5 + 2 is placed in the variable X1.

Functions may be used in either Immediate Mode or Program Mode. Single line functions may be defined using DEF FN within a program and used only within that program.

2.10.3. Expressions

EXPRESSIONS are any valid sequence of constants, variables, functions and operators that gives a value. They may generally be used wherever numbers or strings are expected.
For Example:-

```
3 * 4 + 5 / 6 + TAN(20)
```

is an expression.

```
A$ + "XXXX" + B$
```

is a string expression.

```
PRINT@(A% + C%/2, SQR(B%)) A$ + B$ + LEFT$(C$,5)
```

shows expressions used within the PRINT@ statement.

2.11. FILE NAMING CONVENTIONS

File names may be up to 8 characters long. The first character must be alphabetic and the remainder must be alphanumeric. File names may be followed by a "." plus a 3 letter extension. If an extension is not given it will default to:-

BASIC source files	.BAS
BASIC compiled files	.BAC
Data files	.DAT
Print files	.PRT
Operating system files	.SYS
Operating system commands	.CMD

If the file is associated with a specific drive then the drive number may be added to the filename either at the beginning or the end.
For Example:-

```
0.PROG1.BAS  or  PROG1.BAS.0
```

Both associate file PROG1.BAS with drive 0.

Some statements assume that a particular extension is to be used. If some other extension is required it must be specified.

2.12. MULTIPLE SCREENS

The POLY computer has 4 screens which are displayed in the following order

1 - TEXT 1
2 - GRAPHICS 1 (240 pixels across)
3 - TEXT 2
4 - GRAPHICS 2 (240 pixels across)
The BACKGROUND is displayed at half intensity behind these.

At any time only one screen is current for writing on. This is initially TEXT 1 but can be changed at any time by using SELECT. This selects the current screen but does not display it. DISPLAY must be used to switch it on. This allows screens to be written on when they are not displayed. The BACKGROUND may be altered at any time using the BACKG statement.

5 - FINE GRAPHICS (480 pixels across)

The fine graphics is only possible by combining GRAPHICS 1 and GRAPHICS 2 screens. This is selected by SELECT 5.

If PRINT commands are used while the current screen is a graphics screen, the text is written and displayed on screen 1, whether or not it was currently being displayed.

2.13. CHOOSING COLOUR

2.13.1. Text

Colour on the text screens is selected by printing a colour control character before the data;

```
     PRINT CHR$(1);"POLY SYSTEM"
or   PRINT "#aPOLY SYSTEM"
```

Section 5.2 shows a complete list of the control characters.

2.13.2. Graphics

Colour Choice
Each graphics screen is capable of displaying only 4 colours at a time. Initially these are;

RED, BLUE, GREEN and WHITE

Other colours may be obtained in either of two ways:-

(a) MIXING screens

The first option in MIX, allows the screen 2, screen 3 and screen 4 colours to be mixed. To understand this, the way the secondary colours are obtained must be understood. Each dot on the screen is made up of 3 coloured beams, Red, Blue and Green. The colours obtained are;

```
RED                    = RED
      BLUE             = BLUE
             GREEN     = GREEN
RED + BLUE             = MAGENTA
RED         + GREEN    = YELLOW
      BLUE  + GREEN    = CYAN
RED + BLUE  + GREEN    = WHITE
```

When MIX is ON, the beams for each dot on the screen are mixed with those at the same point on the other screens, and the composite displayed. That is, to get YELLOW, Screen 2 must display a particular dot in RED and Screen 4 the same dot in GREEN (or vice versa). COLOUR allows the secondary colours to be selected and when one is selected, Screen 4 is written on as well as Screen 2.

(b) Additive MIX

Each of the graphics screen may have a particular colour beam added to all dots switched on. This means that instead of RED, BLUE, GREEN and WHITE being displayed, when

RED is added then RED, MAGENTA, YELLOW, and WHITE are displayed,

BLUE is added then MAGENTA, BLUE, CYAN and WHITE are displayed,

GREEN is added then YELLOW, CYAN, GREEN and WHITE are displayed.

Colour Size

On each of the graphics screens, each set of 6 pixels on a row must all be displayed in the same colour. When an attempt is made to set a pixel in a set of 6 to a different colour, then all pixels in that group are turned OFF and only the pixel in the new colour displayed.

3. POLYBASIC

POLYBASIC is a BASIC interpreter designed specially to run on the Poly system. It accepts most standard BASIC commands but some additions and changes have been necessary in order to handle the special features of the Poly System. POLYBASIC includes special features for handling the text and graphics screens.

3.0.1. ABS(X)

Type:- Function

Availability:- All

Syntax:- ABS(numeric-expression)

The absolute value of X is returned, i.e. if X is positive then the value returned is X. If X is negative, the value returned is its positive value.
For Example:-

If A% has the value 16 then

```
B% = ABS(A%)
```

places the value 16 in B%.

If A% has the value -16, then

```
B% = ABS(A%)
```

also places the value 16 in B%.

The statement

```
IF ABS(X) < 4 THEN GOTO 100
```

will cause a branch to line 100 only if X is between -4 and +4.

3.0.2. AND

Type:- Logical Operator

Availability:- All

Syntax:- (expression)AND(expression)

See section 2.9.3 on Logical Operators.

3.0.3. ASC(X$)

Type:- Function

Availability:- All

Syntax:- ASC(string-expression)

Returns the ASCII code (in decimal form) of the first character of the specified string. (See Appendix 5.3 for the ASCII values of characters). If the string is a null string, then 0 is returned.
For Example:-

```
A% = ASC("A")
```

assigns A% the value 65.

When A$ = "POLY" then

```
A% = ASC(A$)
```

assigns A% the value 80 (The ASCII decimal value of P).

This function is useful for converting between upper and lower case as lower case is simply the upper case ASCII value plus 32. Decoding is also simpler.
For Example:-

If the values "A", "B", "C" and "D" are expected it is easier (and faster) to use

```
ON ASC(A$)-64 GOSUB 150, 200, 300, 450
```

than to test for the individual values separately.

3.0.4. ATN(X)

Type:- Function

Availability:- All

Syntax:- ATN(numeric-expression)

The angle whose tangent is X is returned. The returned angle value is in radians.
To change an angle in radians to degrees, multiply by 180/PI.
For Example:-

```
PRINT ATN(1)
```

displays the value .785398 (in radians).

```
PRINT ATN(1) * 180/PI
```

displays the value 45 (in degrees).

3.0.5. BACKG

Type:- Statement

Availability:- All

Syntax:- BACKG numeric-expression

BACKG sets the half intensity background ON or OFF. The numeric expression specifies the colour where

0 = off
1 = red
2 = green
3 = yellow
4 = blue
5 = magenta
6 = cyan
7 = white

For Example;

```
BACKG 4
```

sets the background to blue.

3.0.6. +CAT

Type:- Disk Operating System Command

Availability:- Not available in Standalone Mode or Program Mode.

Syntax:- +CAT [disk-unit-number]

+CAT lists all the files not catalogue protected on the specified disk unit. If the disk unit is not specified, the working unit is listed. The working drive is #0.
For Example:-

```
+CAT 1
```

lists all the files on disk unit 1.

3.0.7. CHAIN

Type:- Statement

Availability:- Not available in Standalone Mode.

Syntax:- CHAIN string-expression[,variable-list]...

CHAIN terminates the current program and loads and runs the program designated in the string. The chained program may be either a BASIC or machine language program. The current screens are not switched off but the current screen is changed to screen 1. All variables except those specified in the variable-list are lost. FETCH is used to put the variables or values into the chained program.

If no filename extension is specified, .BAC is assumed. Uncompiled (.BAS) files may also be chained, but the .BAS extension must be specifically specified.
For Example:-

```
90 A$ ="FILEA"
100 CHAIN A$
```

This chains to program FILEA.BAC. All variables in the current program are lost.

```
1200 CHAIN "NEXTP.BAS", A$, B%, AX*B, D$(4)
```

This chains to program NEXTP.BAS saving the variables and values specified.

The program NEXTP.BAS must have the first statement in the form

```
10 FETCH A$, A%, A, A1$
```

where, although the variable names need not match, the types must.

3.0.8. CHR$(X)

Type:- Function

Availability:- All

Syntax:- CHR$(numeric-expression)

CHR$ performs the opposite of the ASC function, it creates a single character string containing the character represented by the ASCII value (in decimal form) specified. The numeric expression must have a value between 0 and 255 inclusive. Non integer values are truncated. Any value outside this range causes an error. This function may be used when setting up Teletext control characters for the screen.
For Example:-

```
PRINT CHR$(4); "POLY"
```

displays the word POLY on the screen in Blue characters. Appendix 5.2 lists all the Teletext control characters.

```
100 A$ = INCH$
110 IF A$ <> CHR$(13) THEN 100
```

Line 100 uses the INCH$ function to test the keyboard for a key depression. If any key other than the <ENTER> key (CHR$(13)) has been pressed it is ignored.

In order to simplify the insertion of teletext control chracters within strings, as an alternative to writing CHR$(expression), they may be written ||x as part of the string. x is 1 character in length, where

```
||a or ||A is 1
||b or ||B is 2
etc
```

The full list is given in Appendix 5.2.
For Example:-

```
100 PRINT "||aPOLY SYSTEM"
```

writes POLY SYSTEM in red.

3.0.9. CLEAR

Type:- Command

Availability:- All

Syntax:- CLEAR

CLEAR sets the current screen to 1, turns 1 ON and all others OFF, sets the current colour to white and sets the MIX mode to OFF. The screen SPLIT is 24 and scrolling is turned ON. The background is turned OFF.

3.0.10. CLOAD

Type:- Statement

Availability:- All

Syntax:- CLOAD ["filename"]

CLOAD reads the specified file into memory from a cassette tape or other storage device attached to the USER port. The file may be either a BASIC or machine language program. If the filename is not specified then the next file on the device is read. The filename may be up to 8 characters long.
For Example:-

```
CLOAD "MYFILE"
```

3.0.11. CLOCK

Type:- Function

Availability:- All

Syntax:- CLOCK

CLOCK returns a value in the range -32768 to 32767 indicating 10 millisecond intervals.
For Example:-

```
100 A% = CLOCK
          .
          .
          .
200 B% = CLOCK
210 IF A% < B% THEN T = B% - A%
   ELSE T = B% + 65535 - A%
220 ?"TIME TAKEN = ";T*100;"SECONDS"
```

3.0.12. CLOSE

Type:- Statement

Availability:- Not available in Standalone Mode or Immediate Mode.

Syntax:- CLOSE [KILL] channel

CLOSE terminates I/O between BASIC and a file and makes the channel available for another file. The channel must have an integer value between 1 and 12, and must match the channel number specified in the OPEN statement. An error occurs if the specified file is not OPEN.
For Example:-

```
100 CLOSE 3
```

CLOSEs the file which was OPEN on channel 3.

If the file was opened as a NEW file, then the file is added to the directory when it is OPENed.

Files should be OPEN for as short a time as possible, as they maintain a buffer area in memory which is returned to the work area on CLOSING the file.

KILL closes the file and removes it from disk.
For Example:-

```
100 CLOSE KILL 2
```

File 2 is deleted.

3.0.13. CLS

Type:- Statement

Availability:- All

Syntax:- CLS

CLS clears the current screen, and puts the cursor in position (0,0).
For Example:-

```
90 SELECT 1
100 CLS
110 SELECT 2
120 CLS
```

This clears both screens 1 and 2.

3.0.14. COLOR

Type:- Command

Availability:- All

Syntax:- COLOR colour

See COLOUR.

3.0.15. COLOUR

Type:- Command

Availability:- All

Syntax:- COLOUR colour

COLOUR selects the current colour for SET, FILL, LINE and DRAW on the graphics screens. The colour codes are

```
0 = BLACK
1 = RED
2 = GREEN
3 = YELLOW
4 = BLUE
5 = MAGENTA
6 = CYAN
7 = WHITE
```

For Example:-

```
100 COLOUR 2
```

selects the current colour as green.

If no colour has been selected when a DRAW, LINE, SET or FILL operation is made, then WHITE is used.

The secondary colours CYAN, MAGENTA and YELLOW may be obtained only by using MIX in either of two ways:

1. By combining the colours on the two graphics screens. In this case, the drawing is put onto both screens with one colour on each graphics screen. (Note that use of screen 4 reduces memory by 8192 bytes) The colours are then mixed to give the secondary colour. In order to see the secondary colour, both screens 2 and 4 must be DISPLAYed, and MIX must be ON.
For Example:-

```
70 DISPLAY 2
80 DISPLAY 4
90 MIX ON
100 COLOUR 3
110 SELECT 2
120 SET 100,220
```

This sets point (100,220) ON, on screen 2 in red and on screen 4 in green. The MIX command mixes the 2 colours to give YELLOW. For yellow the RED is set on the current screen and GREEN on the other graphics screen

When secondary colours are selected, the colours are displayed as follows

	Current Graphics Screen	Other Graphics Screen
Yellow	Red	Green
Magenta	Blue	Red
Cyan	Green	Blue

2. To each of the graphics screens, a colour may be added to give a further set of colours. This is achieved by using MIX in the second form.
The colour sets available are:

No colour added	red	green	blue	white
Red added	red	yellow	magenta	white
Green added	yellow	green	cyan	white
Blue added	magenta	cyan	blue	white

When using MIX in this form the primary colour is specified by COLOUR and the overlay colour to be added, is specified in MIX.
For Example:-

```
70 DISPLAY 2
80 SELECT 2
90 COLOUR 1
100 MIX 2,2
110 SET (100,210)
```

This sets the point (100,210) on G1 in yellow.

Remember that the primary colours are:-

RED GREEN BLUE

YELLOW = RED + GREEN
MAGENTA = RED + BLUE
CYAN = BLUE + GREEN
WHITE = RED + BLUE + GREEN

3.0.16. COMPILE

Type:- Command

Availability:- Not available in Standalone Mode or Program Mode.

Syntax:- COMPILE "filename"

COMPILE saves a program on disk in compiled form. The filename should be specified in quotes. If no filename extension is specified, BAC is assumed. The compiled program must be loaded later either with a RUN, LOAD or CHAIN. The compiled program takes less storage than a source file and loads faster. A COMPILEd program cannot be listed.
For Example:-

COMPILE "NEWPRM"

This compiles the BASIC program currently in memory onto the working disk as NEWPRM.BAC.

3.0.17. CONT

Type:- Command

Availability:- Not available in Program Mode.

Syntax:- CONT

CONT is used to restart a program when it has been stopped on a STOP statement by pressing the <EXIT> key, or an error. The restart is made following the EXIT if <EXIT> is pressed, at the statement following STOP, or at the start of the statement causing the error.

CONT will not restart the program if any program lines are added while the program is stopped.

3.0.18. CONVERT

Type:- Statement

Availability:- All

Syntax:- CONVERT string-variable1 TO string-variable2

CONVERT changes a line description stored in a string into a boundary description (see FILL). FILL converts a line description to a boundary description each time a FILL is performed, but for efficiency a line description may be converted prior to the FILL by the use of CONVERT.
For Example:-

```
100 CONVERT A$ TO A$
110 CONVERT X$ TO Z$
```

If line descriptions have been set up in A$ and X$ then line 100 converts A$ to a boundary description and line 110 converts X$ to a boundary description storing it as Z$.

3.0.19. COS(X)

Type:- Statement

Availability:- All

Syntax:- COS(numeric-expression)

Returns the COSine of X where X is in radians. To convert degrees to radians, multiply by PI/180.
For Example:-

```
X = COS(60 * PI/180)
```

Assigns X the value 0.5. i.e. COS (60 degrees) = 0.5

3.0.20. CSAVE

Type:- Statement

Available:- All

Syntax:- CSAVE ["filename"]

CSAVE can be used to save BASIC source programs onto cassettes tapes and other devices attached to the user port. If the filename is omitted, then the file has a null name. The files may be loaded using CLOAD. The filename has a maximum of 8 characters.
For Example:-

```
CSAVE "MYFILE"
```

3.0.21. CVER

Type:- Statement

Availability:- All

Syntax:- CVER "filename"

CVER compares the BASIC source file currently in memory with either the specified file, or the next file or the cassette tape attached to the user port.
For Example:-

```
CVER "MYFILE"
```

3.0.22. CVT

Type:- Function

Availability:- All

Syntax:- CVT%$(numeric-expression)
CVT$%(string-expression)
CVTF$(numeric-expression)
CVT$F(string-expression)

The CVT functions store numeric data in strings, and vice versa, using integer or floating point formats. Integer values are stored in 2 bytes while floating point values take 8 bytes. These functions should not be confused with STR$ and VAL which convert ASCII decimal values to and from integer or floating point formats (packed values).
For Example:-

CVT%$ moves an integer value into a 2 character string. (% to $)
CVT$% moves a 2 character string value back into a integer ($ to %).
CVTF$ moves a floating point number into an 8 character string (F to $).
CVT$F moves an 8 character string value back into a floating point number ($ to F).

If a string is longer than required only the first 2 or 8 characters are taken, depending on whether CVT specifies integer or floating point. These functions allow fast storage of numeric values in strings and are especially useful for the storage of numeric data in records on a file as they pack them into the minimum of disk space.
For Example:-

```
100 A$ = CVT%$(25665)
```

This stores the 2 byte integer value 25665 in the 2 character string A$. If printed out the string A$ would appear as:-

```
dA
```

as the first character would have the ASCII decimal value 100 (25665/256) and the second character the ASCII decimal value 65.

3.0.23. DATA

Type:- Statement

Availability:- Not available in Immediate Mode.

Syntax:- DATA value [,value]...

DATA lets data be stored inside a program and be accessed by a READ statement when it is required. Each item in the list is separated by a ",". Each time a READ is encountered, the next item in the list is read. Strings do not need to be enclosed in quotes unless they include embedded commas or colons. Multiple DATA lines may be used and need not be placed together in the program. When all the data has been read from one DATA statement, the next READ accesses the next DATA statement. A DATA statement must be the only statement on the line. Strings and numerics may be mixed. RESTORE may set the line number at which the read is to start.
For Example:-

```
 50 DIM Array$(3)
100 DATAJAN, FEB, MAR, APR, MAY
200 DATA "  100", " 1000", "10,000"
300 DATA50,40,30,20
400 RESTORE 200
500 FOR I = 1 TO 3
600 READ Array$(I)
700 NEXT
```

This sets up Array$ with the 3 string values of 100, 1000 and 10,000.

3.0.24. DATE$

Type:- Function

Availability: All

Syntax:- DATE$

Returns the current date in the form: DD-MON-YY
For Example:-

If the current date is the 21 July 1981, then

```
100 D$ = DATE$
```

assigns D$ as a 9 character string with the value 21-JUL-81. The date is set when the first POLY unit logs onto the system.

3.0.25. DATE$(1)

Type:- Function

Availability:- All

Syntax:- DATE$(1)

This returns, as a 6 character string, the current date in the form YYMMDD. Dates used in this form can be immediately compared numerically.
For Example:-

```
100 A = VAL(DATE$(1))
```

gets the date and converts it to its numeric value.

```
200 A$ = DATE$(1)
```

gets the date as a 6 character string.

3.0.26. DEF FN

Type:- Statement

Availability:- Not available in Immediate mode

Syntax:- DEF FN variable-name (dummy-variable) = expression

DEF FN allows users to define their own single line functions containing one argument and returning a floating point value.

The function name is formed by preceding the variable name in the definition by FN.
For Example:-

```
DEF FNA(B) = SQR(B)/2 + 10
```

This defines Function FNA which takes the square root of the argument, divides it by 2 and adds 10.

This could then be used in an expression such as:

```
A = FNA(16) + FNA(36)
```

which would give A the value 12 + 13 = 25.

In the example, B is a dummy variable inserted simply to identify the dummy variable in the expression. The returned value will be floating point. String functions cannot be included within the expression. DEF FN must be the last statement in the line.

3.0.27. DIGITS

Type:- Statement

Availability:- All

Syntax:- DIGITS total-number [,number-of-decimal-places]

DIGITS specifies how many digits to print. The number is in the range 1-17. The second argument specifies the number of decimal places, and must be less than or equal to the total number. If a number will not fit the format it is printed in scientific notation. If DIGITS is not specified a number with more than 6 digits before or after the decimal point will be printed in scientific notation.
For Example:-

```
10 DIGITS 5,4
20 E=2.7182818285
30 PRINT E
```

This will return 2.7183

```
40 DIGITS 2
50 PRINT 999
```

This will return 9.99E+2

3.0.28. DIM

Type:- Statement

Availability:- Not available in Immediate Mode.

Syntax:- DIM variable(size) [,variable(size)]...

Sets the number of items allowed in each of the dimensions in an array. All arrays must be dimensioned before they can be used. There is no limit to the number of dimensions.
For Example:-

```
100 DIM A$(10), B%(4,2)
200 DIM C$(2,2,2,2,2)
```

Line 100 sets up a one dimensional string array with subscript elements 0 - 10; and a two dimensional integer array with elements 0,0 to 4,2.

Line 200 sets up a 4 dimensional string array with elements 0 to 2 in each dimension.

Arrays cannot be redimensioned within a program.

3.0.29. DISPLAY

Type:- Statement

Availability:- All

Syntax:- DISPLAY screen number [OFF]

DISPLAY is used to turn screens ON and OFF. The screen numbers are

1 - Text 1 screen
2 - Graphics 1 screen (240 by 204)
3 - Text 2 screen
4 - Graphics 2 screen (240 by 204)
5 - Fine graphics screen (480 by 204)

For Example:-

```
DISPLAY 1
```

Displays the top text screen.

```
DISPLAY 3
```

Displays the lower text screen ON.

Following a RUN, CHAIN or CLEAR, all screens except screen 1 are turned OFF but not cleared.

If screen 4 is displayed while screen 5 is still being displayed, vertical white lines may appear on the screen.

Screen 5 has a priority the same as screen 2.

3.0.30. DIV

Type:- Arithmetical Operator

Availability:- All

Syntax:- numeric-expression DIV numeric-expression

DIV performs division and gives a truncated integer result.
For Example:-

```
100 A = 10DIV4
```

sets A to the value 2.

3.0.31. DOS

Type:- Command

Availability:- Immediate mode

Syntax:- DOS

This places the Poly unit in the Disk Operating System (DOS) mode. A description of operation in this mode, along with a description of the full set of DOS utilities are contained in the POLY SYSTEM UTILITY Manual.

To return to POLYBASIC, type

```
BASIC <ENTER>
```

3.0.32. DPEEK

Type:- Function

Availability:- All

Syntax:- DPEEK (address)

DPEEK returns the integer (2 character) value held at the specified address. The address must be between 0 and 65535. While in BASIC, the operating system and screen areas cannot be accessed using this command.
For Example:-

```
A%=DPEEK(1165)
```

puts into A% the value held in addresses 1165-1166.

Note that addresses in the range 32768 to 65535 may be placed in integer variables and used, but if printed out will print as the 2's complement, ie with 65535 subtracted from them.

3.0.33. DPOKE

Type:- Statement

Availability:- All

Syntax:- DPOKE address,value

DPOKE puts the 2 byte integer value at the specified address. The value must be in the range 0 to 65535.
For Example:-

DPOKE 1165, 2456

This puts into memory location 1165-1166 the value 2456.

Care must be taken in using this statement to ensure that the values changed are at locations where the meaning of the contents are known.

3.0.34. DRAW

Type:- Statement

Availability:- All

Syntax:- DRAW string-variable
or DRAW # channel

DRAW places onto a screen, the line whose coordinates have previously been defined using LDES$ and put in a string variable or onto a file, or a screen dump which has been stored in a variable or file using STORE.

A line is drawn in the current colour, on the current screen, at the place it was defined.

For a screen dump, the current screen must be a graphics screen. The screen dump is placed in the same place on the screen from which it was stored in the original colours.
For Example:-

```
10 CLS
100 Car$ = LDES$(115,0,115,20,100,20,100,
    70,130,70,130,0,115,0)
110 SELECT 2:CLS:DISPLAY 2
120 DRAW Car1$
```

This draws the car as specified on screen 2.

```
130 STORE (100,0)(130,70)Car$
135 CLS
140 SELECT 4 : DISPLAY 4
150 DRAW Car1$
```

This stores the car as a screen dump in Car1$, clears screen 2 and dumps it onto screen 4 in the same place.

If the screen dump or the line description have been previously stored on a disk file, the file (which may contain several screen dumps) must have previously been opened in the program.
For Example:-

```
100 OPEN OLD "DIAGRAMS" AS 1
110 CLS : SELECT 2 : CLS : DISLAY 2
120 FOR Number = 1 TO 5
130 DRAW #1
140 NEXT
150 CLOSE 1
```

This reads 5 diagrams off the disk file DIAGRAMS.DAT and displays them on screen 2.

3.0.35. DRAW@

Type:- Command

Availability:- All

Syntax:- DRAW@(row, column) string-variable
or DRAW@(row, column) #channel

DRAW@ shifts the coordinates relative to the specified row, column and DRAWs the diagram in the new position. (See also FILL and DRAW).
For Example:-

```
50 CLS
100 Car$ = LDES$(115,0,115,20,100,20,100,
    70,130,70,130,0,115,0)
110 SELECT 2 : CLS : DISPLAY 2
120 FOR Col = 240 TO -2 STEP -10
130 COLOUR 1 : DRAW@(115,Col)Car$
140 COLOUR 0 : DRAW@(115,Col+10)Car$
150 NEXT
```

This causes the car to move from right to left across the screen.

3.0.36. DRIVE

Type:0 Statement

Availability:- All

Syntax:- DRIVE source-number

DRIVE reassigns the working drive for a particular POLY unit. The drive-number should be in the range 0 to 3.

This allows a particular POLY unit to be working on a different drive from otheunits in the system.
For Example:-

```
DRIVE 1
```

3.0.37. ELSE

Type:- Statement

Availability:- All

Syntax:- IF condition [THEN] [:stmt]...ELSE stmt [:stmt]...

ELSE must always be used in conjunction with an IF statement. The statement(s) following the ELSE are performed if the condition is FALSE. The ELSE must not be immediately preceded by a colon. If an assignment (such as A=24) is used following the condition, without a THEN, a space must follow the condition.

A line number immediately following ELSE is read as ELSE GOTO linenumber.
For Example:-

The following are all correct constructs:

```
100 IF A%>3 OR B%<2 THEN GOTO 200 ELSE GOTO 300

200 IF A=4 THEN B=0 :C=0 ELSE D=0 :E=0

300 IFC<X C=X:D=0ELSEIFC<YTHENC=Y:D=1ELSEC=Z:D=Z

400 IF A[%=4 PRINT A% ELSE 2000
```

The following are NOT valid constructs:-

```
500 IF A%>4 GOTO 1000 : ELSE GOTO 2000

600 IF A%=4 AND B%=6 ELSE C%=0
```

3.0.38. END

Type:- Statement

Availability:- Not available in Immediate Mode.

Syntax:- END

END terminates program execution. It may be placed anywhere in the program and is not essential. On reaching the last statement in a program, a program will terminate.
For Example:-

```
100 IF A% > 100 END
```

Following END, a program cannot be restarted by using CONT, but it may be RUN again.

3.0.39. ERL

Type: Function

Availability: All

Syntax:- ERL

ERL returns the line number on which a program stops when an error is encountered. The error may have occurred in a previous line. If there is an ON ERROR routine, then the line number may be checked within the ON ERROR routine and the appropriate action taken.
For Example:-

```
5000 IF ERL<3000 OR ERL>3200 THEN RESUME
```

This line would probably be inside an ON ERROR suboutine and causes errors occurring between lines 3000 and 3200 to be ignored.

3.0.40. ERR

Type: Function

Availability: All

Syntax:- ERR

ERR returns the error number after an error has occurred. This is useful in an ON ERROR routine in sending back appropriate error messages. A full list of error numbers is given in Appendix 5.1.
For Example:-

```
1000 IF ERR<50 THEN RESUME 2200 ELSE PRINT ERL;ERR:END
```

This statement would probably occur within an ON ERROR routine. Errors with numbers less than 50 are sent to a routine at line 2200.

3.0.41. ERROR

Type:- Statement

Availability:- All

Syntax:- ERROR numeric-expression

ERROR allows an error to be simulated during testing. ERR and ERL are set up as if the error had occurred and the ON ERROR branch is taken as if it has been set.
For Example:-

```
ERROR 4
```

makes the program act as if error 4 had occurred.

In the error routine, RESUME NEXT rather than RESUME must be specified as RESUME will just cause repetition of the ERROR statement.

3.0.42. EXEC

Type:- Statement

Availability:- Not available in Standalone

Syntax:- EXEC "filename"

EXEC loads and executes the machine language program contained in filename. Care must be taken to ensure that the program loaded does not interfere with the BASIC program running. Utilities which run in the utility command space may be executed with this command. Following execution of the program, control is returned to the BASIC program.
For Example:-

```
100 EXEC "CAT 1"
```

This displays the catalogue on screen.

A complete description of the utilities is given in the POLY SYSTEM UTILITIES MANUAL.

3.0.43. EXP(X)

Type:- Function

Availability:- All

Syntax:- EXP(numeric-expression)

EXP returns e to the power of X. The maximum value of X allowed is 88. EXP is the inverse of the LOG function i.e. X=EXP(LOG(X)). Where LOG(X) is the natural logarithm of X.
For Example:-

```
Y=EXP(1)
```

sets Y to 2.7182818, the value of e.

3.0.44. FETCH

Type: Statement

Availability:- Not available in Standalone

Syntax:- FETCH variable1 [,variable2]...

When a BASIC program is run from another program by the use of CHAIN, all variable values are reset unless saved by specifying a variable list with CHAIN. These variables are reloaded in the called program using FETCH. The variables in the CHAIN and FETCH lists must match by type, but need not have the same name. FETCH must be the first executable statement in the program.
For Example:-

```
CHAIN "NEXTPROG",Address$,NO%
```

could be put in the first program and

```
FETCH Address$,NUMBER%
```

could be put as the first line of the second program.

3.0.45. FIELD

Type:- Statement

Availability:- Not available in Standalone or Immediate Mode.

Syntax:- FIELD #channel, fieldsize AS stringname ...

FIELD is used with random access file statements PUT and GET. FIELD is used to specify string subfields within the record buffer.
For Example:-

```
10 OPEN OLD RANDOM "TRIAL" AS 1
20 FIELD#1, 10 AS A$, 4 AS X$, 8 AS D$
30 GET #1
```

Line 10 opens the file TRIAL on channel 1.
Line 20 assigns A$ as the label for the first 10 bytes (characters) in the record buffer of channel #1, whatever these may contain, X$ as the next 4 characters, and D$ as the last 8 characters.
Line 30 reads the next record on the file into the record buffer.
A$,X$ and D$ have now been assigned as above.

After GETting the record, a FIELD statement may reassign the fields within the record.
For Example:-

```
40 FIELD#1, 23 AS Z$, 10 AS A$, 4 AS X$, 8 AS D$
```

will reassign the labels Z$, A$, X$ and D$, while the data in the buffer is unchanged.

If it is required to store numeric fields in the record, the CVT function be used to convert them.
For Example:-

```
5 DIM A%(125)
10 OPEN OLD RANDOM "TRIAL1" AS 9
20 GET #9
30 FOR I=0 to 125
40 FIELD #9, I*2 AS X$, 2 AS A$
50 A%(I)=CVT$%(A$)
100 NEXT I
```

This program GETs a single record from TRIAL1.DAT into the record buffer and breaks it into 126 2 character fields. These are then stored as integers in the array A%. X$ is a dummy string to put successive A$ labels in the right places. Further processing can then be done with the values stored in the array.

To store strings in a random access file we use FIELD, LSET and PUT.
For Example:-

```
10 OPEN NEW RANDOM "TRIAL2" AS 12
20 FIELD #12,20 AS A$, 30 AS B$
30 LSET A$="AAA"
40 LSET B$="BBB"
50 PUT #12, RECORD 3
60 CLOSE 12
```

Lines 30 and 40 assign the strings "AAA + 17 spaces" and "BBB + 27 spaces" into A$ and B$ respectively. The FIELD statement in line 20 assigns these to the first 50 bytes of the buffer. Line 50 PUTs the contents of the buffer into RECORD 3 of the new file TRIAL2.DAT.

3.0.46. FILL

Type:- Command

Availability:- Not available in Standalone

Syntax:- FILL [USING pixels[,shift];] line-description

FILL fills a defined area on a graphics screen with a specified pattern in the current colour. The boundary is specified using LDES$, the points being described in a clockwise direction. The pattern is described by two values, the pixels, and the shift. The pixels are set up in groups of 6 which are represented by the decimal value of their binary pattern. If the pixels are not specified, 63 is used. If the shift is not specified no shift is assumed.
For Example:-

The pattern 101010 is changed to its binary value 42 and this is entered. The easiest way to do this conversion is by assigning values to each digit and adding them up.

```
32 16  8  4  2  1
 1  0  1  0  1  0 = 42
 0  1  1  1  0  1 = 29
 0  0  0  0  0  1 = 1
```

Shift is the number of pixels each row is shifted to the right.
For Example:-

For pattern 000001 a shift of 1 gives the following patterns on following rows

```
Row1 000001
Row2 100000
Row3 010000
Row4 001000
Row5 000100
```

this gives a diagonal effect.
For Example:-

```
10 SELECT 2
20 CLS
30 DISPLAY 2
40 A$=LDES$((2,0)(2,100)(50,100)(50,10)(2,10))
50 COLOUR 4
60 FILL USING 1,1;A$
```

This selects screen 2, clears it, displays it and fills the rectangle defined with blue diagonal lines.

The FILL is executed in two stages, the line description is first converted to a boundary description and then the area is filled. If the same line description is to be filled repeatedly, a time saving may be made by converting the line to a boundary description using CONVERT, and storing this in a string.
For Example:-

```
10 A$ = LDES$(2,0,2,100,50,100,50,10,2,10)
20 CONVERT A$ TO A$
30 SELECT 2 : CLS : DISPLAY 2
40 FOR flash = 1 TO 100
50 Col = RND(4) : IF Col = 3 then Col = 7
60 COLOUR Col
70 FILL A$
80 NEXT
```

This fills the box with a solid random colour 100 times.

3.0.47. FILL@

Type:- Statement

Availability:- Not available in Standalone.

Syntax:- FILL@(row,column)[USING pixels[shift];]line description

This transposes the area to be filled to start at the row, column specified. This acts the same as DRAW@.
For Example:-

```
100 FILL@(10,30) USING 21,1;A$
```

If A$ is as specified in the FILL example, then all points are moved by the difference between (10,30) and (2,0). That is the rows have 8 added to them and the columns 30.

(10,30)(10,130)(58,130)(58,80)(10,40)

This area is then filled.

3.0.48. FOR

Type:- Statement

Availibility:- All

Syntax:-
FOR variable = start-value TO end-value [STEP increment]

FOR starts a loop so that a sequence of program statements can be executed over and over again. The loop must be terminated by a NEXT statement. The specification of the variable in the NEXT statement is optional.
For Example:-

```
10 FOR Tom = 1 TO 6
20 PRINT Tom;
30 NEXT Tom
```

is the same as writing

```
10 PRINT 1;
20 PRINT 2;
30 PRINT 3;
40 PRINT 4;
50 PRINT 5;
60 PRINT 6;
```

The start and end values may be either constants, variables or expressions. The variable in the FOR statement is to count the number of times the loop is passed through and care must be taken if this is changed within the program.
For Example:-

```
100 FOR N%=1 TO 10
110 N%=N%*2
120 NEXT
```

Lines like 110 which change the value should be used with caution.
Note that the variable name need not be specified in line 20.

STEP specifies the amount to be added to the counter variable at the end of each loop, it may be positive, negative or contain a decimal point. (If not specified, 1 is assumed). At the end of each loop, the counter is incremented or decremented. If the counter variable is greater than the final value the program goes on and executes the statement after the NEXT. (If the STEP is negative, the test is less than the final value)

The NEXT statement may specify the counter variable. A FOR loop may be terminated by a GO TO from within the loop but a GO TO should never reenter the middle of a FOR loop unless it has previously exited from it with a GO TO.

FOR loops may be nested, i.e. placed one inside another.
For Example:-

```
5 X=0.4
10 FOR I=1 TO X*7 STEP 0.5
20 PRINT "OUTER LOOP"; I
30 FOR J=1 TO 2
35 PRINT "INSIDE LOOP"
40 NEXT
50 NEXT
```

This would print

```
OUTER LOOP 1
INSIDE LOOP
INSIDE LOOP
OUTER LOOP 1.5
INSIDE LOOP
INSIDE LOOP
OUTER LOOP 2
INSIDE LOOP
INSIDE LOOP
OUTER LOOP 2.5
INSIDE LOOP
INSIDE LOOP
```

Using integer values for variables saves time and memory space. A FOR loop is always executed at least once.

3.0.49. FRE

Type:- Function

Availability:- All

Syntax:- FRE(0) or FRE(1) or FRE(-1)

FRE(0) returns the number of bytes of free memory. To get the maximum amount available with no program loaded type

```
NEW <ENTER>
PRINT FRE(0) <ENTER>
```

The memory displayed is the amount available for BASIC programs. When a BASIC program is loaded, it takes up part of this area. As it runs it uses further memory for storing strings and other pointers and may run out of memory during operation. Should this occur it is necessary to reduce the size of the program and/or the amount of memory used in strings, arrays etc. Suggestions to achieve this are:

-Remove all REM statements from the program.
-When a string is no longer needed reset it to null e.g.
A$=""
-Put multiple statements on lines using a colon to separate statements.
-Improve your coding.
-If screen 4 is used, 8142 bytes can be saved by doing all graphics on screen 2.

FRE(1) returns the value of the current high address for memory. (See MEM).

FRE(-1) returns the maximum number of characters that a new string may have. This will always be less than the maximum amount of memory left as returned by FRE(0). By checking FRE(-1) it is possible to stop errors occurring due to lack of free space.

3.0.50. GET#

Type:- Statement

Availability:- Not available in Immediate Mode or in Standalone Mode.

Syntax:- GET #channel [,RECORD record-number]

GET reads a specific record from a file and places it in the record buffer for that channel. To access the data in that record, a FIELD statement must be used. If a record number is specified, that record is read from the file. If it is not specified, then the next record on the file is read.
For Example:-

10 OPEN OLD RANDOM "TEST" AS 4	File TEST.DAT is opened on channel 4.
20 GET #4, RECORD 24	Record 24 is read into the buffer.
30 FIELD #4, 100 AS A$, 10 AS B$	The 1st 100 bytes are labelled as A$, the next 10 as B$.
40 CLOSE 4	The file is closed.

See FIELD for further examples.

3.0.51. GOSUB

Type:- Statement

Availability:- Not available in Immediate Mode.

Syntax:- GOSUB linenumber

GOSUB transfers program control to the subroutine beginning at the specified line number. When a RETURN statement is encountered, control is returned to the statement following the GOSUB.

For Example:-

```
150 IF I%=1 GOSUB 1000
160 PRINT "RETURN FROM SUBROUTINE"
170 END
1000 PRINT "SUBROUTINE"
1010 RETURN
```

With I% set to 1 this would display

```
SUBROUTINE
RETURN FROM SUBROUTINE
```

A subroutine may be used over and over again from various places in a program.

3.0.52. GOTO

Type:- Statement

Availability:- Not available in Immediate Mode.

Syntax: GOTO linenumber

GOTO transfers program control to the specified line number. The line number may not be held in a variable.
For Example:-

```
100 PRINT "A";
120 GOTO 100
```

This would result in a continuous loop with the computer displaying A's. To stop the program it would be necessary to press the <EXIT> key.

```
100 PRINT "A";
110 GO TO 1000
     .
     .
     .

1000 PRINT "B"
```

This simply transfers control to line 1000 and would display AB, because of the ";". Lines between 110 and 1000 would not be actioned.

3.0.53. HEX

Type:- Function

Availability:- All

Syntax:- HEX (string-containing-Hex-value)

HEX converts a hexadecimal string into its decimal equivalent. Unless assembler programming is being undertaken, this function will probably not be necessary.
For Example:-

```
PRINT HEX("100")
```

displays 256

3.0.54. IF THEN

Type:- Statement

Availability:- All

Syntax:- IF condition(s)[THEN]action(s)

IF instructs the computer to test the condition. If the condition is TRUE, control proceeds to the action clause(s) following o n the same line. If the expression is FALSE, control jumps to the matching ELSE statement (if there is one) or down to the next line.

If there are multiple statements (separated by :) following the THEN, all of these will be actioned if the condition is TRUE.

If an assignment statement is used, a space must be left after the condition. See ELSE and GOTO.
For Example:-

```
100 IF A%<4 AND B%=2 THEN GOSUB 1000
200 IF X>127 THEN B%=3 : C% = 0 : GOTO 300
210 IF Z>256 INPUT A$
220 IF X%=1 X%=9999
```

Note the space in line 220 before X%=9999 is necessary.

The following lines all perform the same action

```
300 IF A$ = "END" THEN GOTO 2000
400 IF A$ = "END" GOTO 2000
500 IF A$ = "END" THEN 2000
```

3.0.55. INCH$

Type:- Function

Availability:- All

Syntax:- INCH$ [(channel)]

INCH$ reads a character from the specified channel. If a character has been input then that character is returned, otherwise CHR$(0) is returned. If no channel number is specified, then the keyboard is scanned to see if a key has been pressed. The most useful way to use this function is to test the keyboard during a loop while some other action is going on.
INCH$ neither displays a cursor nor the character received.
For Example:-

This example continues looping until any key is pressed.

```
100 IF INCH$ = CHR$(0) THEN 100
200 Continue processing
```

The following example repeatedly tests for characters A,B,C, throwing away any other characters received.

```
100 A$ = INCH$
200 IF A$ < "A" OR A$ > "C" THEN 100
300 ON ASC (A$)-64 GOSUB 500, 600, 700
400 GOTO 100
500 REM "A" SUBROUTINE
550 RETURN
600 REM "B" SUBROUTINE
650 RETURN
700 REM "C" SUBROUTINE
750 RETURN
```

3.0.56. INCH$(0)

Type:- Function

Availability:- All

Syntax:- INCH$(0)

INCH$(0) is a special form of INCH$. It also scans the keyboard, but waits until a key has been pressed before returning a value. The cursor is displayed, the character is printed, and the cursor is incremented. It is not necessary to press <ENTER> to terminate the entry.
For Example:-

```
10 CLS
20 A$ = INCH$(0)
30 GOTO 20
```

needs a print

is a program which allows continuous typing.

3.0.57. INPUT

Type:- Statement

Availability:- Not available in Immediate Mode.

Syntax: INPUT ["message";] variable [,variable]...

INPUT causes the computer to stop execution until the specified number of fields are entered on the keyboard. The input statement may specify a list of string or numeric variables to be input. The items in the list must be separated by commas. When typing the Strings, fields need not be enclosed in quotes unless leading or trailing spaces are to be included or the string includes a , or a :. When the INPUT is complete <ENTER> must be pressed.

When a numeric value is required and a non-numeric value is entered, an error is returned and the program is terminated. To trap these errors and return to the INPUT statement, ON ERROR must be used and the error tested.

INPUT displays a ? followed by the cursor in the next PRINT position. To position an INPUT to a specified place on the screen, the INPUT statement should be preceded by a PRINT@(row, column) and a semi-colon.
For Example:-

```
100 PRINT@(20, 15);
200 INPUT A$
```

A "prompting message" may also be included in the INPUT statement. This is printed before the ? prompt. The statement must be enclosed in quotes and followed by a ";".
For Example:-

```
100 INPUT "NAME"; N$
110 INPUT "AGE IN YEARS, HOUSE NUMBER"; A%,H%
```

The following would be displayed on the screen. The data typed in is underlined.

```
NAME? JIM <ENTER>
AGE IN YEARS, HOUSE NUMBER? 15,113 <ENTER>
```

If insufficient fields are entered, further ? prompts are given until all the requested values have been entered.
Pressing <ENTER> without typing in any other data, leaves the previous value in the variable unchanged.
Entering spaces only followed by <ENTER> sets a string variable to a single space. In all other cases leading spaces are ignored.
For Example:-

```
6 <SPACE> <SPACE> 0 <ENTER>
```

is the same as typing

60<ENTER>

3.0.58. INPUT#

Type:- Statement

Availability:- Not available in Standalone Mode

Syntax:- INPUT #channel, variable [,variable]...

INPUT# reads the next sequential record from a disk file on the specified channel and places it in the variables specified in the variable list. INPUT#0 works the same as the ordinary INPUT statement except that no ? prompt is given.

The disk file must first be OPENed and later CLOSEd. The disk file must previously have been created to match the form in the variable list. See PRINT#. To test for the end of the file, an ON ERROR trap should be set and a test an end of file error, made.
For Example:-

INPUT and list all the records from file "NAMES.TXT". The records each contain 3 fields.

```
5 ON ERROR GO TO 1000
10 OPEN OLD "NAMES.TXT" AS 1
20 INPUT #1, A$,B%,C
30 PRINT A$
40 PRINT B%,C
50 GO TO 20
1000 IF ERR=8 AND ERL=20 THEN END
```

3.0.59. INPUT LINE

Type:- Statement

Availability:- Not available in Immediate Mode.

Syntax:- INPUT LINE string-variable

INPUT LINE inputs all data typed in up until <ENTER> is pressed and places it in the string variable. Only one string is allowed. Data typed in may include commas unlike INPUT where the comma specifies the start of a new field. Any prompting text must be printed by a previous statement.
For Example:-

```
100 PRINT@(10,0) "TYPE IN YOUR NAME AND ADDRESS"
110 INPUT LINE A$
```

On the screen this would appear as:

```
TYPE IN YOUR NAME AND ADDRESS
? JOHN SMITH, 246 BERKLEY GROVE, WADESTOWN <ENTER>
```

All of the name and address typed in by the user is placed in A$, including the commas.

3.0.60. INPUT LINE#

Type:- Statement

Availability:- Not available in Immediate Mode

Syntax:- INPUT LINE# channel , string-variable

INPUT LINE# works exactly like INPUT LINE but the string comes from the disk file which is OPEN on the specified channel. The input takes all characters up to the first line feed(CHR$(13)).

3.0.61. INSTR

Type:- Function

Availability:- All

Syntax:- INSTR(start-position, string-name, sub-string)

INSTR looks for a sub-string in a string starting at the start-position. The character position at which the sub-string is found is returned. Zero is returned if the sub-string is not found.
For Example:

```
10 A$="ABCDEFGHIJ"
20 B%= INSTR(3,A$,"G")
30 PRINT B%
```

A$ is searched for "G" starting at the third character. The value 7 put into B%.

```
40 PRINT INSTR(1,A$, "DEF")
```

would print the value 4.

3.0.62. INT(X)

Type:- Function

Availability:- All

Syntax:- INT(numeric-expression)

INT truncates a numeric expression to the integer less than or equal to the expression.
For Example:-

```
INT(9)              returns  9
INT(6.7)            returns  6
INT(-1.1)           returns -2
INT(X*10+.5)/10     returns X accurate to one decimal place.
```

Note:- In division, DIV may be used in place of / and INT to give the integer result. Similarly, division performed with integer variables gives an integer result.
For Example:-
The following all give the same integer result.

```
Z = INT(B/C)
Z% = B/C
Z = BDIVC
```

3.0.63. KILL

Type:- Statement

Availability:- All

Syntax:- KILL "filename"

KILL deletes a file from disk. The defaults are the working drive and BAS extension.
For Example:-

```
100 KILL 1.XXXX.AAA
```

Line 100 KILLs file XXXX.AAA which is found on drive 1.

3.0.64. +KILL

Type: DOS Command

Availability:- Immediate Mode only

Syntax:- +KILL filename

+KILL may be used as an alternative to KILL to delete a file from disk. +KILL gives an option to confirm the deletion and reports if the file is not there. The filename is not placed in quotes. The default extension is DAT.
For Example:-

```
+KILL MYPROG.BAS
```

3.0.65. KVAL

Type:- Function

Availability:- All

Syntax:- KVAL

KVAL returns the ASCII value of the key pressed to cause transfer to an ON KEY routine. KVAL gives the value of the key pressed. if <A> was pressed KVAL would give the value 65.
For Example:-

```
10 ON KEY GOTO 1000
.
.
.
1000 IF KVAL=65 THEN END
1010 RESUME NEXT
```

3.0.66. LDES$

Type:- String Function

Availability:- Not Standalone

Syntax:- LDES$ (line-description)

LDES$ sets up a graphics line description as a string for use with DRAW, DRAW@, FILL, FILL@ and CONVERT. The line-description consists of a list of coordinates, each pair optionally surrounded by brackets.
For Example:-

```
100 A$ = LDES$((0,0),(100,0),(100,100),(0,100),(0,0)
```

describes a box 100 pixels each way.

This could also be written as:-

```
100 A$ = LDES$(0,0,100,0,100,100,0,100,0,0)
```

to save space. If a ; is used in place of a , between coordinate pairs, then the line is broken at that point and a new line started.
For Example:-

Store the line description of the box given above, in a single string, such that only the left and right sides are drawn.

```
100 A$ = LDES$((100,0)(100,100);(0,100),(0,0))
```

If the line is to be used as a boundary in FILL or FILL@, the points must be specified in a clockwise order. The coordinates need not be inside the screen area.

3.0.67. LEFT$

Type:- String function

Availability:- All

Syntax:- LEFT$(string, numeric-expression)

LEFT$ returns the first n characters of the given string.
For Example:-

```
10 A$="ABCDEFGHIJKLM"
20 X$= LEFT$(A$,5)
```

This puts the value "ABCDE" into X$.

The numeric expression must be between zero and 32,767. Non integer expressions are truncated. See RIGHT$ and MID$.

3.0.68. LEN(X$)

Type:- Function

Availability:- All

Syntax:- LEN(stringname)

LEN returns the length of the specified string including spaces and control characters.
For Example:-

```
10 D$="asd  fghjk"
20 L=LEN(D$)
```

This sets L to the length of D$, i.e. 10.

3.0.69. LET

Type:- Statement

Availability:- All

Syntax:- [LET] variable = expression

LET assigns a value to a variable. The word LET may be omitted. The variable and the expression must be of the same type.
For Example:-

```
 5 DIM G(8,8)
10 LET X%=7
20 LET C$="ABCDEF"
30 D=SQR(5)*7.8+9    LET left out.
40 G(0,8)=6          dimensioned array element.
50 A%="ABCDE"        invalid statement!
```

3.0.70. LINE

Type:- Statement

Availability:- All

Syntax:- LINE line-description

LINE is used to draw lines on to the current screen.

LINE may be used on any of the screens but if used on the text screens, the programmer must ensure that the graphics control characters are printed on the rows before using this command.

All points are specified as coordinates in the form (row, column). The coordinates may be specified with or without surrounding brackets. If brackets are used, the comma between points may be omitted. If brackets are omitted, then all commas must be included.
For Example:-

```
LINE (10,10)(20,15)(30,20)(40,10)
LINE (10,10),(20,15),(30,20),(40,10)
LINE 10,10,20,15,30,20,40,10
```

are all legitimate forms of specification and draw a line joining the four points.

LINES may be stopped and started by using a ; in place of the comma between points. The points on either side of the ; are not joined.
For Example:-

```
   LINE (10,10) (20,15);(30,20)(40,10)
or LINE 10,10,20,15;30,20,40,10
```

would draw 2 lines, one joining (10,10) to (20,15) and a second line joining (30,20) to (40,10).

Use of LINE on the graphics screens

COLOUR 0 may be use to remove a line. The colours YELLOW, MAGENTA and CYAN can only be obtained by using MIX, either by adding a colour to the current screen or by mixing the two screens. (See MIX and COLOUR)

If points are specified outside the screen, then the line is drawn to the edge of the screen as if it was joined to that point.

Use of LINE on the text screens

On teletext screens, the user must ensure that graphics control characters have been written to the lines prior to using LINE. A simple loop to insert these could be:-

```
10 PRINT@(0,0);:FOR ROW=0 TO 23: PRINT "↓r":NEXT
```

which prints "↓r" or CHR$(18) in column 0 of all rows on the screen.

Because the graphics control character is placed in column 0, chunky pixels cannot be placed in chunky graphic columns 0 or 1 on the text screen.
For Example:-

```
10 SELECT 1
20 CLS: FOR I%=0 TO 23: PRINT "↓r":NEXT
30 LINE (14,10),(42,78),(0,0)
```

draws a line joining the 3 points screen 1 in green.
For Example:-

```
10 SELECT 2
20 COLOUR 1
30 DRAW (0,0),(203,0),(203,239),(0,239),(0,0)
```

draws a line around the boundary of the screen.

Lines may also be defined using LDES$ and areas of the screen saved using STORE and saved in strings or in files using STORE.

3.0.71. LIST

Type:- Command

Availability:- All

Syntax:- LIST [start-line[- end-line]]

LIST displays a single program line, a group of lines, or the whole program. A long listing may be stopped for examination at any time by pressing <PAUSE>. Use of the <SPACE BAR> or the <PAUSE> key allows stepping through the listing, line by line. Pressing <EXIT> terminates the listing.
For Example:-

LIST	Lists the whole program.
LIST10-90	Lists lines 10 to 90.
LIST83	Lists line 83.
LIST-200	Lists lines 1 to 200.
LIST200-	Lists from line 200 to the end of the program.

3.0.72. +LIST

Type:- DOS Command

Availability:- Available only in Immediate Mode.

Syntax:- +LIST filename

+LIST lists the contents of the specified file onto the screen.
For Example:-

```
+LIST 1.AAAA.TXT
```

File AAAA.TXT is listed from drive 1. If the file extension is not given .TXT is used. If the drive is not specified the working drive is used.

Within programs, this may be executed by:-

```
EXEC "LIST 1.AAAA.TXT"
```

The <PAUSE> acts as described in LIST.

3.0.73. LOAD

Type:- Command

Availability:- Not available in Standalone Mode.

Syntax:- LOAD "filename"

LOAD loads a BASIC program source file from disk into memory. The defaults are the working drive and .BAS.
For Example:-

```
LOAD"AAAAAA"        loads AAAAAA.BAS from the working drive.
LOAD"1.BBBB.TXT"    loads BBBB.TXT from drive 1.
```

3.0.74. LOG(X)

Type:- Function

Availability:- All

Syntax:- LOG(numeric-expression)

LOG is the natural logarithm. To calculate logs for other bases use:

```
LOG of X to base Y = LOG(X)/LOG(Y)
```

In particular, for common logs:

```
LOG of X to base 10 = LOG(X)/LOG(10)
```

For Example:-

```
10 A%=2
20 L=LOG(A%)
30 PRINT L
```

This prints .693147181 .

The inverse of LOG(X) is EXP(X) i.e. X= LOG(EXP(X))

3.0.75. LOGOFF

Type:- Statement

Availability:- All

Syntax:- LOGOFF

LOGOFF returns the POLY unit to the LOGON state and removes the current program from memory.

3.0.76. LSET

Type:- Statement

Availability:- All

Syntax:- LSET string-variable = string-expression

LSET stores a new string value in an existing string storage location. The value is either truncated if it is too long, or has spaces added to it if it is too short. (Compare RSET.)
LSET is usually used to put a value into a variable in the I/O buffer before writing it to a disk file.
For Example:-

```
10 OPENOLDRANDOM"XXXX" AS 9
20 FIELD#9,5 AS A$,6 AS B$,6 AS C$
30 LSET A$="AAAAAXXX"
40 LSET B$="BBB"
50 LSET C$="CCC"
60 PUT#9,RECORD 10
70 CLOSE 9
```

Line 10 OPENs file XXXX.DAT on channel 9. Line 20 assigns positions in the buffer to A$,B$ and C$. Lines 30-50 put the values into the strings in the buffer as follows:

A$="AAAAA", B$="BBB ", C$="CCC "

Line 60 PUTs the contents of the buffer into RECORD 10 in the disk file XXXX.DAT .

3.0.77. MEM

Type:- Command

Availability:- All

Syntax:- MEM [memory-high-address]

MEM resets the top of memory address of the BASIC programs working area. This is useful when a protected area is required for a machine language subroutine. The current memory address can be obtained using FRE(1). If the address is not specified it is reset to the initial system value.

MEM should not be used within GOSUB routines, ON ERROR routines, ON KEY routines or ON SEC routines as the stack containing the return address is stored just below high memory.
For Example:-

```
10 MEM FRE(1)-100
20 REM SAVES 100 BYTES
.
.
.
10000 REM END OF PROGRAM
10010 MEM
```

This "protects" 100 bytes of memory in line 10 and resets memory back to the original address in line 10010.

3.0.78. MID$

Type:- Function

Availability:- All

Syntax:- MID$(string,start-number[,no-of-chars])

MID$ takes a middle part of a string and places it in a variable.
For Example:-

```
10 X$="ABCDEFGHI"
20 M1$=MID$(X$,3,5)
30 M2$=MID$(X$,4)
```

The result is: M1$="CDEFG", and M2$="DEFGHI"

See LEFT$ and RIGHT$.

3.0.79. MIX

Type:- Command

Availability:- All

Syntax:- MIX [ON]
or MIX [OFF]

or MIX screen, colour

MIX is used for controlling:-
a. The mixing of the colours on screens 2, 3 and 4.
b. The adding of a colour to all pixels ON within screens 2 or 4.

MIX ON specifies that the beams on screens 2, 3 and 4 are to be MIXed. If the current COLOUR is either YELLOW, CYAN or MAGENTA, then DRAW, FILL and LINE will draw on both screens 2 and 4, and if both screens 2 and 4 are displayed, then these colours will be displayed MIXed.

MIX OFF switches MIX mode OFF so that the screens display in their priority order.

MIX screen, colour, adds the colour to all pixels ON on the specified graphics screen. See COLOUR for examples.

Mixing Colours

The colour of each pixel on the screen is determined by which of the 3 colour beams - red, blue, and green - are switched on. These are the primary colours. The secondary colours have 2 or more beams on such that:-

```
Red   + Green       = Yellow
Red   + Blue        = Magenta
Green + Blue        = Cyan
Red + Green + Blue  = White
```

All 7 colours are available on the text screens, but the graphics screens have only the primary colours immediately available. The secondary colours are created using the MIX command. When two colours are mixed the resulting colour is the composite of the two colours.
For Example:-

```
Red + Magenta = Red + Red + Blue
              = Red + Blue
              = Magenta
```

Note that if a beam is repeated in the MIX, it is not doubled in intensity.

3.0.80. NEW

Type:- Command

Availability:- All

Syntax:- NEW

NEW deletes the current program from the POLY unit.

3.0.81. NEXT

Type:- Statement

Availability:- All

Syntax:- NEXT[loop-variable]

NEXT terminates a FOR loop. The loop variable name need not be specified.
For Example:-

```
10 FOR N%=1 TO 12
20 PRINT N%
30 NEXT
```

This prints a list of the first 12 integers.

This could have been written

```
10 FOR N% = 1 TO 12
20 PRINT N%
30 NEXT N%
```

3.0.82. ON ERROR GO TO

Type:- Statement

Availability:- Not available in Immediate Mode

Syntax:- ON ERROR GOTO linenumber

ON ERROR GO TO allows the user to trap errors and carry out whatever action is required. When an error occurs in a program without an ON ERROR GO TO , or the line number given is 0, the program terminates with a message like

```
ERROR #41 IN LINE 210
```

When an ON ERROR GO TO routine is included, the program will go to the specified line when an error occurs. The error number is put into ERR. The line number on which the error occurs is put into ERL. The ON ERROR GO TO statement must come somewhere before the line causing the errors. See ERR and ERL. After an ON ERROR GO TO routine, control is passed back to the main program with a

RESUME statement.
For Example:-

```
10 ON ERROR GO TO 1000
     .
     .
210 OPEN NEW "FILE" AS 3
220 ...
     .
     .
1000 IF ERL=210 AND ERR=41 THEN RESUME 220
```

This program tries to OPEN a file. If the file is already open then control is transfered to line 220 and the program continues.

The ON ERROR routine does not trap errors within itself.
ON ERROR routines may be tested by simulating errors using ERROR. (See ERROR)

3.0.83. ON GOSUB

Type:- Statement

Availability:- Not available in Immediate Mode

Syntax:- ON numeric-variable GOSUB line[,line]...

ON GOSUB allows different subroutines to be called from the same line. Control will RETURN to the line following the ON GOSUB statement.
For Example:-

```
100 ON N GOSUB 110,120,130,140,150
105 GOTO 200
```

If N = 1 control goes to the subroutine at line 110.
If N = 2 " " " 120 etc.
If N < 0 or N > 5 control goes to the next statement, line 105.

The statement automatically truncates any non integer values for N. ON GOSUB must be the last statement on a line. Values of N less than or greater than the number of items in the list cause the program to continue onto the following statement.

3.0.84. ON GOTO

Type:- Statement

Availability:- Not available in Immediate Mode

Syntax:- ON numeric-variable GOTO line[,line]...

ON GOTO is the same as ON GOSUB except that control is not RETURNed to the line following the ON GOTO statement.

If the numeric-variable has a value less than 1 or greater than the number of items in the list, then the program continues onto the following statement.
For Example:-

```
50 ON N% GOTO 200, 210, 220
```

If N% = 1 control goes to the statement at line 200.
If N% = 2 " " " 210.
If N% = 3 " " " 220.
If N% < 0 or N% > 3 the program continues onto the line following line 50.

3.0.85. ON KEY

Type:- Statement

Availability:- Not availabe in Immediate Mode

Syntax:- ON KEY [key-no-1][TO key-no-2] GOTO [linenumber]
or ON KEY keynumber = new-value

ON KEY allows the functions performed by specific keys to be programmed.

Programming the Special Keys
The special keys include the numeric keypad, the cursor keys, the editing keys and other special purpose keys. These are all assigned a special ON KEY number as follows:

Key(s)	ON KEY value
Numeric keypad numbers 0-9	0-9
EXIT	10
PAUSE	11
ENTER	12
NEXT	13
REPEAT	14
BACK	15
HELP	16
CALC	17
back arrow	18
forward arrow	19
down arrow	20
up arrow	21
INS CHAR	22
DEL CHAR	23
INS LINE	24
DEL LINE	25
. on keypad	26
SHIFT PAUSE	27
@	28
£	29
EXP	30
|	31

The HELP and CALC keys cannot be trapped in an ON KEY routine.

The ON KEY procedure works in a similar way to an ON ERROR procedure in that as soon as the key specified is pressed, a special routine in the program is performed. Control is passed back using RESUME.

During the ON KEY routine the ASCII value of the key pressed is in KVAL.
For Example:-

```
20 ON KEY 12 GO TO 1000
30 other statements
:
1000 CLS
1010 RESUME NEXT
```

This would cause the program to clear the current screen whenever <ENTER> was pressed. The program would RESUME normal operation at the statement following that during which <ENTER> was pressed. The keyboard is checked after every statement.
For Example:-

```
20 ON KEY 10 GO TO 1000
30 :
:
1000 RESUME NEXT
```

This disables the <EXIT> key. The ON KEY routine at line 1000 does nothing except continue when the <EXIT> key is pressed.
For Example:-

```
10 DIM a%(9)
15 Count = 0
20 ON KEY 0 TO 9 GOTO 1000
.
.
.
1000 a%(count) = KVAL
1010 count = count +1
1020 IF count > 9 ON KEY 0 TO 9 GOTO 0
1100 RESUME NEXT
```

This routine allows up to 10 numeric keys to be pressed during other processing. These are stored in the array a%. When 10 keys have been pressed, the ON KEY is switched OFF.

The whole keyboard may be trapped by ON KEY by omitting the key-number.
For Example:-

```
20 ON KEY GO TO 1000
:
1000 IF KVAL < 65 OR KVAL > 90 RESUME NEXT
:
:
1100 RESUME NEXT
```

In this example, whenever any key is pressed, the ON KEY routine is executed. In line 1000, if the key pressed is not a capital letter, processing continues.

Changing the ASCII values of the numeric keypad
ON KEY may also be used to assign a new ASCII value to the keys 0 to 9 of the numeric keypad.
For Example:-

It is required to enter from the keyboard the teletext characters 1/4, 1/2 and 3/4 as these are not on the keyboard. The numeric keypad numbers 1, 2 and 3 are assigned for entering these characters.

```
20 ON KEY 1 = 123
30 ON KEY 2 = 92
40 ON KEY 3 = 125
```

If the <HELP> AND <CALC> key ASCII values are given to one of the numeric keypad keys, then they would be actioned as for <HELP> and <CALC>.

3.0.86. ON SEC

Type:- Statement

Availability:- Not available in Immediate mode.

Syntax:- ON SEC [interval] GOTO line-number

ON SEC acts in a manner similar to ON ERROR or ON KEY except that the ON SEC routine is performed at specified time intervals. The interval is specified in seconds.
For Example:-

```
100 ON SEC 5 GOTO 1000
.
.
.
1000 ?@(0,32)TIME$;
1010 RESUME NEXT
```

During this program, the time would be displayed every 5 seconds on the top line of the screen.

3.0.87. OPEN

Type:- Statement

Availability:- Not available in Standalone Mode or Immediate Mode

Syntax:- OPEN OLD [RANDOM] filename AS channel
NEW

Before any file may be used in BASIC, it must be OPENed.
If a file is read from beginning to end using an INPUT# statement, access to that file is sequential. If the file is read using the GET# statement, access is random.

Sequential files are accessed using PRINT# and INPUT#.
For Example:-

```
10 OPEN NEW "AAAA" AS 7        OPENs a new file on channel 7
20 A$="DD" : B=9.2 : C%=99
30 PRINT#7,A$,",",B,",",C%     PRINTs data into the file.
40 CLOSE 7                     CLOSEs the file on channel 7
50 OPEN OLD "AAAA" AS 2        OPENs an existing file on
                               channel 2
60 INPUT#2,W$,X,Y%             INPUTs data from the file
70 PRINT W$,X,Y%               PRINTs the data on the POLY
                               unit
80 CLOSE 2                     CLOSEs the file on channel 2

The display shows:
DD      9.2     99
```

The variable names are not stored with the data. Default extensions are the working drive and .DAT. Channel numbers may be variables or expressions with a value less than 12.

Random files must have the word RANDOM. Random files are accessed using GET# and PUT#.

The use of PUT# and GET# is shown in the following example.
For Example:-

```
10 OPEN OLD RANDOM "EXAMPLE" AS 11
20 GET#11,RECORD 2
30 FIELD#11,20 AS A$,30 AS B$
40 PRINT A$;B$
50 LSET A$="AAA"
60 LSET B$="BBB"
70 PUT#11,RECORD 2
80 CLOSE 11
```

The old file EXAMPLE.DAT is opened, record 2 is read into the buffer and the field s within it are defined. Strings A$ and B$ are printed then new data is moved to them in the buffer. Then the whole contents of the buffer is PUT back in to the same record on the disk. Lastly the file is closed.

3.0.88. PEEK(X)

Type:- Function

Availability:- All

Syntax:- PEEK(address)

PEEK returns the single character value held at the address. The address must be between 0 and 65536. The value returned will be between 0 and 255 inclusive. See DPEEK.
For Example:-

```
A% = PEEK(1161)
```

This puts into A% the ASCII decimal value of the data stored at address 1161.

3.0.89. PI

Type:- Function

Availability:- All

Syntax:- PI

PI returns the value 3.14159265
For Example:-

```
A = PI*R*R
```

The area of a circle is calculated.

3.0.90. POKE

Type:- Statement

Availability:- All

Syntax:- POKE address, numeric-expression

POKE stores a single character integer value at the specified address. The address must be between 0 and 65535 The numeric-expression must have a value between 0 and 255 inclusive. See DPOKE.
For Example:-

```
POKE 1161,1
```

This puts the value 1 into address 1161.

3.0.91. POS(X)

Type:- Function

Availability:- All

Syntax:- POS(channel)

POS returns the current column POSition of the cursor on the specified channel. Positioning starts at zero. Channel numbers must not be greater than 12.
For Example:-

```
10 CLS
20 PRINT TAB(10);
30 X=POS(0)
40 PRINT X
```

Channel 0 is the screen, so this program displays the number 10 starting in column 10 (Numbers have preceding and trailing spaces). Note that the ";" is essential otherwise the position returns to 0 on the next line. The POS is calculated from the beginning of the line, or from the start column of a PRINT@ statement.
For Example:-

```
10 CLS
20 OPEN NEW "EX3" AS 12
30 A$="AAAAAA"
40 PRINT#12,A$;
50 X=POS(12)
60 PRINT X
70 CLOSE 12
```

This OPENs a NEW sequential file, PRINTs A$ into it, then displays the number 6. Where the number is displayed depends on the current screen position.

3.0.92. POS(-1)

Type:- Function

Availability:- All

Syntax:- POS(-1)

POS(-1) returns the current row number of the cursor on the screen.
For Example:-

```
20 row = POS(-1)
30 column = POS(0)
40 PRINT row, column
```

This print the position of the cursor on the screen at the start of the program.

3.0.93. PRINT

Type:- Statement

Availability:- All

Syntax:- PRINT print-list

The print-list is a list of items to be printed. PRINT causes the screen to display the items in the print-list starting at the current PRINT position. This position is varied by the punctuation in the PRINT statement.

No punctuation after an item causes the current position to move to the start of the next line.
For Example:-

```
10 PRINT "AAA"
```

A comma after an item causes the next item to be PRINTed starting in column 0,8,16,24, or 32 (whichever is the next column).
For Example:-

```
20 PRINT "0","8","16","24","32"
```

This displays the numbers each starting in the given column. i.e.

```
0       8       16      24      32
```

A semi-colon after an item causes the next item to be PRINTed immediately following the previous one.
For Example:-

```
30 PRINT "A";"B"
```

This will display

```
AB
```

with no spaces between.

An exclamation mark causes the next data to be printed on the next line starting in the same column as the start of the last data printed.
For Example:-

```
20 PRINT@(10,10)"Line 1"!
30 PRINT"Line 2"!
40 PRINT "Line 3"!
```

would be printed as, starting on line 10

```
Line 1
Line 2
Line 3
```

Commas, semi-colons and exclamation marks may be mixed in a single PRINT statement.

Numbers are printed with one trailing space, and one leading space unless this is filled with a minus sign. All strings, including 'number' strings, have no leading or trailing blanks.
For Example:-

```
100 X$="ABC" : B=4.2 : C%=7
110 PRINT X$ : PRINT B : PRINT C%
120 PRINT X$,B,C%
130 PRINT X$;B;C%
```

This program displays:

```
ABC
4.2
7
ABC    4.2     7
ABC 4.2  7
```

3.0.94. PRINT@

Type:- Statement

Availability:- All

Syntax:- PRINT@(row,column) [,]print-list

PRINT@ specifies the point where the PRINTing starts. The rows are numbered from 0 to 23, and the columns from 0 to 39. When in row 23 use a semi-colon to stop scrolling of the text. A character printed in the bottom right hand portion of the screen does not scroll, if followed by a ;.
For Example:-

```
10 X=555
20 PRINT@(10,4)"X =";X
30 PRINT@(23,0)"This is the bottom line";
```

This displays: "X = 555" starting at position (10,4), and "This is the bottom line" starting at position (23,0).

3.0.95. PRINT USING

Type:- Statement

Availability:- All

Syntax:- PRINT USING string, print-list

PRINT USING uses various special strings to format the data displayed. The string is an image of the required output, but with special characters in place of the actual characters. The print list is similar to that of the PRINT statement. Literal characters, i.e. ones that are not special, may be inserted in

front of the string of special characters. Special characters are: !,",#,$,*,!, and sometimes a comma.

```
10 PRINT USING"! ! !","Joe","E","Bloggs"
```

This prints the initial letters with a single space between each pair. i.e. J E B. The "!" denotes a single character string field.

```
20 PRINT USING"!!!","Joe","E","Bloggs"
```

This prints the initial letters without spaces. i.e. JEB

```
30 PRINT USING"%1234%","ABCDEFGH"
```

This prints the first 6 letters of the string i.e. ABCDEF. The pair of "%"'s denote a string field with a length equal to the total length of the image string. The middle characters are arbitrary.

```
40 PRINT USING"####.###",123.4567
```

This prints 123.457 The "#" characters denote a number field. Decimal points are lined up and the number is rounded to fit the format.

```
50 PRINT USING"$$##.##",987.659
```

This prints $987.66 . Two dollar signs are needed to position $ immediately before the first digit. The second $ also acts like a #, allowing another digit before the point.

```
60 PRINT USING"$$###.###-",-10.1234
```

This prints $10.123- . If a "$" or"*" must precede a negative number, then the minus sign must follow the number.

```
70 PRINT USING"**###.#",1.23
```

This prints ***1.2 . Two asterisks cause all leading spaces to be filled. The second asterisk also acts like a "#".

```
80 PRINT USING"$**###.##",67.89
```

This prints $***67.89 . The $ is a 'literal' character preceding the asterisks.

```
90 PRINT USING"###,#.##",2345.67
```

This prints 2,345.67 . A single comma embedded somewhere in the numeric field will cause the number to be printed out with commas separating every three digits.

```
100 PRINT USING"#.###!!!!",123456
```

This prints 1.235E+05. Four up arrows are needed after the numeric field to print in scientific notation.

Note:- If the number is too big to fit into a format a % sign will be printed followed by the unformatted number.

3.0.96. PRINT@ USING

Type:- Statement

Availability:- All

Syntax:- PRINT@(row,column) USING string, print-list

PRINT@ may be combined with USING. Refer to PRINT USING.
For Example:-

```
10 A$="###.##" : X=4.5
20 PRINT@(10,15)USING A$,X
```

This will print 4.5 in row 10 with the decimal point in column 18.

3.0.97. +PRINT

Type:- DOS Command

Availability: Not available in Standalone or Program Modes.

Syntax:- +PRINT filename

The specified file is printed out on the printer. The default drive is the working drive, and default extension is .DAT.
For Example:-

```
+PRINT MYPROG.BAS
```

This prints the BASIC source program MYPROG out on the printer.

3.0.98. PRINT#

Type:- Statement

Availability: Not available in Standalone Mode or Immediate Mode.

Syntax:- PRINT#channel, print-list

PRINT# outputs the data specified to a sequential access disk file. The print-list format is as specified in INPUT#.
For Example:-

```
10 OPEN NEW"AAAA" AS 5
20 PRINT#5,A$,",",B,",",C%
30 CLOSE 5
```

Be careful to insert ",", between variables. This is very easy to overlook. See INPUT#.

3.0.99. PTR

Type:- Function

Availability:- All

Syntax:- PTR(variable)

PTR returns the address of the variable or variable pointer in the memory.
Floating point variable values are stored as 8 bytes and PTR returns the start address of their 8 bytes.

Integers are stored as 2 bytes, PTR returning the start address of these.

A string is held as a 4 byte pointer and the actual string in a different area. The first 2 bytes of the pointer contain the start address of the string and the last two bytes the length of the string. PTR returns the start address of the string pointer.
For Example:-

```
10 A$="ABCDEFGHIJ"
20 A%=PTR(A$)
30 PRINT A%
40 PRINT DPEEK(A%);DPEEK(A%+2)
```

This could display

```
4206
4215  10
```

The 4 byte string descriptor is stored in bytes 4206-4209 inclusive. The string itself starts at 4215 and is 10 bytes long.

3.0.99. PUT#

Type:- Statement

Availability:- Not available in Immediate or Standalone Modes.

Syntax:- PUT #channel [, RECORD numeric-expression]

PUT# is used to PUT data from a 252 character I/O buffer into a random access disk file. FIELD is used to define the buffer area and LSET and RSET must be used to place the data in the buffer.
For Example:-

```
10 OPEN NEW RANDOM "NNNNN" AS 3
20 FIELD#3,6 AS G$,5 AS H$,4 AS I$
30 LSET G$="GG" : LSET H$="HHH" : LSET I$="IIIIII"
40 PUT#3,RECORD 8
50 CLOSE 3
```

This has now PUT into record 8 of file NNNNN the data:
"GG HHH IIII"

Each time a PUT or GET statement is actioned and the RECORD number is not specified, the RECORD number is automatically increased by one.
For Example:-

```
10 OPEN NEW RANDOM "PPPP" AS 6
20 FIELD#6, 5 AS X$, 6 ASY$, 7 AS Z$
15 FOR N%=1 TO 10
30 LSET X$="XX" : LSET Y$="YY" : LSET Z$="ZZ"
40 PUT#6
45 NEXT N%
50 CLOSE 6
```

This PUTs the same three strings into the first 18 bytes of RECORDs 1 to 10.

GET# is used to read the data from the file.

3.0.99. RANDOM

Type:- Statement

Availability:- All

Syntax:- RANDOM

RANDOM generates a new seed for the RND function. This means that each time RANDOM is used, a new sequence of RND values is started.
For Example:-

```
20 ? RND(10)
```

will always give the same result each time after switching the POLY unit ON

```
10 RANDOM
20 ? RND(10)
```

ensures that the value is different each time.

3.0.99. READ

Type:- Statement

Availability:- Not available in Immediate Mode

Syntax:- READ variable-list

READ is used to READ data specified in a DATA statement and place it in specified variables. The data must match the variable type. For Example:-

```
300 READ A,B%,C$
500 DATA 32.4,55,AAAA
```

This will take the next 3 items in a DATA list, call them A,B%, and C$ respectively, and store them in memory.

Trying to READ DATA items when there is no more data causes an error. This may be avoided by using ON ERROR GO TO and testing for ERR = 71. To use the same data again use the RESTORE statement.

3.0.99. REM

Type:- Statement

Availability:- Not available in Immediate Mode

Syntax:- REM [remarks]

Any text in a REMark statement is ignored by the program and is used purely for documentation to make the program more understandable.
For Example:-

```
230 REM Put explanations here !!!!!!!!!
```

3.0.99. RENAME

Type:- Statement

Availability:- Not available in Standalone Mode

Syntax:- RENAME filename1, filename2

This is used to change the name of files on disk. The defaults are the working drive and BAS extension.
For Example:-

```
RENAME "1.AAAA.DAT","1.BBBB.DAT"
RENAME"AB123","AB100"
```

File 1.AAAA.DAT is renamed 1.BBBB.DAT and file AB123.BAS is renamed AB100.BAS.

3.0.99. RESET

Type:- Statement

Availability:- All

Syntax:- RESET [(]row,column[)]

RESET switches off the specified pixel. On text screens, RESET switches off the Chunky graphic pixel. (See SET)
For Example:-

```
100 RESET 24,26

200 RESET (24,27)
```

switches off the 2 pixels specified.

3.0.99. RESTORE

Type:- Statement

Availability:- Not available in Immediate Mode

Syntax:- RESTORE [line-number]

If the same DATA is to be READ in more than once, a RESTORE statement will allow the program to go to the beginning of the DATA, or to the first DATA item after the specified line number.
For Example:-

```
200 READ A,B,C
210 READ D,E,F
220 RESTORE 280
230 READ G,H,I
240 RESTORE
250 READ J,K,L
260 PRINT A,B,C,D,E,F,G,H,I,J,K,L
270 DATA 2.3,3.4,4.5
280 DATA 6.7,7.8,8.9
```

This program reads in the 6 data items on lines 270 and 280 and assigns them to variables A to F. The data is RESTOREd to line 280, so G, H, and I take on the values 6.7, 7.8, 8.9 respectively. The data is then RESTORED to the first DATA statement so J, K, and L take on the values 2.3, 3.4, and 4.5.

3.0.99. RESUME

Type:- Statement

Availability:- Not available in Immediate Mode

Syntax:- RESUME [line-number]
or RESUME NEXT
or RESUME LINE

RESUME is the only way to transfer control back to the main program after an error routine, an ON KEY routine or an ON SEC routine.

If the line-number is omitted or is 0, control is returned to the start of the statement containing the error.

If the line-number is specified, then control is returned to the line specified

If LINE is specified, then control is returned to the start of the line on which the error occured.

If NEXT is specified, then control is returned to the next statement following that on which the error occured.
For Example:-

```
20 ON ERROR GO TO 300
30 READ A$
40 . . . . .
     .
     .
300 IF ERR=71 THEN PRINT "NO MORE DATA"
310 RESUME 40
```

For Example:-

```
20 ON KEY 0 GOTO 1000
30   .
40   .
1000 RESUME NEXT
```

This disables the 0 on the numeric keypad.

3.0.99. RETURN

Type:- Statement

Availability:- Not available in Immediate Mode

Syntax:- RETURN

When a subroutine is completed, control is returned to the main program with RETURN. This sends control to the next statement after the GOSUB.
For Example:-

```
100 GOSUB 2000
     .
     .
2000 REM SUB 2000 STARTS HERE
     .
     .
2060 IF R=99 THEN RETURN ELSE STOP
```

3.0.99. RIGHT$

Type:- String Function

Availability:- All

Syntax:- RIGHT$(string, no-of-characters)

RIGHT$ returns the last no-of-characters in a string.
For Example:-

```
10 A$="ABCDEFGH"
20 AA$=RIGHT$(A$,4)
```

The value placed in AA$ is "EFGH".
See LEFT$, and MID$.

3.0.99. RND(X)

Type:- Function

Availability:- All

Syntax:- RND(numeric-expression)

If X is the numeric expression, for X = 0 to X = 1 this function produces random numbers between 0 and 1.
For X = 1 or more, a random number between 1 and X is generated.
For Example:-

RND(5) returns values of 1,2,3,4,5 each with 20% probability.

Negative integer values for X set up a defined random string. RANDOM sets the random string randomly so that after switching on, a new RANDOM string is used.
For Example:-

```
100 Z%=RND(-10) : REM TEMPORARY LINE DURING TESTING
```

This sets the random generator to give the same results each time the program is run.

For Example:-

```
100 RANDOM
```

This ensures that each run gives different results.

3.0.99. RSET

Type:- Statement

Availability:- Not available in Immediate Mode.

Syntax:- RSET string-variable = string-expression

RSET is similar to LSET except that the inserted string is right justified, i.e. it is packed on the left with spaces to make it fit. Truncation is the same as for LSET.
For Example:-

```
10 OPEN OLD RANDOM "FFFF" AS 4
20 FIELD#4, 6 AS A$, 6 AS B$
30 X$="123" : Y$="12345678"
40 RSET A$=X$
50 RSET B$=Y$
60 PUT#4
70 CLOSE 4
```

This sets A$=" 123" and B$="123456" in the disk file buffer area. This is then written into the file FFFF.DAT.

3.0.99. RUN

Type:- Command

Availability:- All

Syntax:- RUN ["filename"]

After LOADing a BASIC source program from disk into a POLY unit it may be RUN without specifying the filename. Specifying the filename causes a COMPILEd program to be loaded from disk and started automatically. The filename extension defaults are .BAC and the working drive.
For Example:-

RUN	Executes the BASIC source program in the POLY unit.
RUN"AAAA"	Loads the compiled program AAAA.BAC from disk and executes it.

3.0.99. SAVE

Type:- Command

Availability:- Not available in Program or Standalone Modes.

Syntax:- SAVE ["filename"]

After creating or editing a source program it may be SAVEd on disk. Any existing program of the same name is automatically overwritten. The filename extension defaults are the working drive and BAS extension.
For Example:-

```
SAVE"1.AAAAAA.TXT"
```

This SAVEs file AAAAAA.TXT on the disk on drive one.

3.0.99. SCROLL

Type:- Statement

Availability:- All

Syntax:- SCROLL ON
or SCROLL OFF

If SCROLL is ON, when the bottom of the page is reached, during printing or with the cursor arrow, then all lines move up one line and the top line is lost. Moving off the top of the screen with the up arrow causes the lines to SCROLL down.

When SCROLL is OFF, when the bottom of the screen is reached, printing continues at the top of the screen. The cursor arrows simply move the cursor from the bottom line of the screen to the top line (or vice versa).

SCROLL is turned OFF on a RUN, CHAIN, CLEAR or NEW.
For Example:-

```
100 SCROLL OFF
```

Also see examples in SPLIT.

3.0.99. SELECT

Type:- Command

Availability:- All

Syntax:- SELECT screen-number

SELECT sets the current screen for writing. The screens available are

```
1 Top Text Screen (20 by 40)
2 Top Graphics Screen (204 by 240)
3 Bottom Text Screen (24 by 40)
4 Bottom Graphics Screen (204 by 240)
5 Fine Graphics Screen (204 by 480)
```

At the start of a program, the current screen is always 1. SELECT does not DISPLAY the screen. This must be done with a separate command.

The graphics commands SET, RESET, LINE and DRAW use the graphics on the current screen. If the current screen is a graphics screen, PRINT may still be used to display text. It is placed onto screen 1.

At the END of a program, all screens except 1 are turned OFF but not cleared.

On an error or <EXIT>, all screens are left exactly as they were at the time. Commands may be typed in and these are displayed on screen 1, but if the current screen is screen 3, the displays tney produce are placed on screen 3.
For Example:-

```
10 CLS
20 SELECT 2
30 CLS
40 DISPLAY 2
50 COLOUR 1
60 LINE 0,0,0,239,203,239,203,0,0,0
70 PRINT@(10,10) "SCREEN 1"
80 GOTO 80
```

Line 10 clears screen 1
Line 20 selects the graphics screen 2
Line 30 clears screen 2
Line 40 displays screen 2
Line 50 selects the colour red
Line 60 draws a box around the screen in red on screen 2
Line 70 prints on screen 1, the words SCREEN 1
Line 80 loops so that display is not turned off

3.0.99. SET

Type:- Statement

Availability:- All

Syntax:0 SET [(]row, column[)]

SET switches on the specified pixel on the current screen. On graphics screens, the pixel is turned on in the current colour. If the colour is a secondary colour, the specified pixel is also turned on in the other graphics screen. MIX must be ON and both

screens 2 and 4 displayed for this to be displayed in the correct colour. (See COLOUR and MIX).

For pixels to be displayed using SET

Rows should be between 0 and 71 for screens 1 and 3.
Rows should be between 0 and 203 for screens 2, 4 and 5.
Columns should be between 2 and 79 on screens 1 and 3.
and Columns should be between 0 and 479 on screen 5.

On text screens, a teletext graphics control character must have been placed on the start of the line, previously. If this is missing the character equivalent of the graphics character is displayed. The colour is as set up in the graphics control character.

```
100 CLS
110 FOR row = 0 TO 23:PRINT@(row,0)" ,";:NEXT
120 SET 15, 20
130 SET (64, 70)
140 SELECT 2:CLS:DISPLAY 2
150 SET (120, 230)
160 COLOUR 1
170 SET 121, 230
```

Line 110 prints graphics green control characters on Screen 1.
Lines 120-130 set points on
Line 140 selects, clears and displays screen 2
Line 150 sets a point on in WHITE
Line 160 sets COLOUR to red
Line 170 sets a point on in RED

Note:- On graphics screens the colour must be the same for each group of 6 horizontal points. If a point is set on in a colour different from the colour in that group of 6 pixels, all other pixels in that group are turned OFF, the new colour is set, and only the one point in the group is displayed.

3.0.99. SGN(X)

Type:- Function

Availability:- All

Syntax:- SGN(numeric-expression)

SGN returns 1 if X is positive, 0 if X is 0, and -1 if X is negative.
For Example:-

```
10 A=-2 : B=0 : C=14
20 PRINT SGN(A),SGN(B),SGN(C)
```

This will print out the values: -1 0 1

3.0.99. SIN(X)

Type:- Function

Availability:- All

Syntax:- SIN(numeric-expression)

This is the SINe function with X in radians. Multiply degrees by PI/180 to convert to radians.
For Example:-

PRINT SIN(30 * PI/180) will print the value .5

3.0.99. SOUND

Type:- Command

Availability:- All

Syntax:- SOUND[pitch, length]

SOUND produces a pitched sound of the specified length. The pitch is calculated as 62,500/pitch. The equivalent note values for these are given in section 5.5. The length is specified in 10 millisecond lengths.
For Example:-

```
10 C=239:G=159:A=142:B=127:C1=120
.
.
.
100 SOUND C,50
110 SOUND C,50
120 SOUND G,50
130 SOUND G,50
140 SOUND A,25
150 SOUND B,25
160 SOUND C1,25
170 SOUND A,25
180 SOUND G,100
```

This plays the first line of "BAA BAA BLACK SHEEP" as GG.

3.0.99. SPLIT

Type:- Command

Availability:- All

Syntax:- SPLIT number-of-lines[,cursor-action]

Each of the text screens may be split into 2 independent scrolling screens.

The number-of-lines specifies the number of lines in the top half of the screen.

The cursor-action, if specified must contain either 0 or 1. If this is 0 or not defined, then the cursor is left in place after each PRINT. If 1 is specified, after each PRINT, the cursor is returned to the top half of the screen where it was when SPLIT was specified.
For Example:-

```
100 CLS : SPLIT 10
110 GOSUB 200
120 PRINT@(10,0);
130 GOSUB 200
140 END
200 FOR I = 0 TO 20
210 PRINT I
220 NEXT
```

Line 100 splits the screen
Line 110 Prints the numbers 0 to 20 in the top half
Line 120 moves the cursor to the top line of the bottom half
Line 130 Prints the numbers 0 to 20. This shows the independant scrolling of the two areas. If the line

```
105 SCROLL OFF
```

is inserted, then the return to the top of each screen can be seen.

With a split screen, CLS only clears the half of the screen on which the cursor is currently placed.
For Example:-

Add the line

```
230 CLS
```

to the program in the above example and only the bottom half of the screen is cleared.
For Example:-

```
100 CLS : SPLIT 23,1
110 FOR I = 0 TO 100
120 PRINT I
130 PRINT@(23,0)I
140 NEXT
```

Line 100 clears the screen, placing the cursor in (0,0) and then splits the screen specifying that the cursor is to return to the top half. In the loop (lines110-140), the number is printed in the top half and then in the bottom half. Following the PRINT@(23,0) in the bottom half, the cursor is returned to its position in the top half for the next PRINT in line 120.

On a RUN, CHAIN or CLEAR, SPLIT is reset to SPLIT 24,0.

3.0.99. SQR(X)

Type:- Function

Availability:- All

Syntax:- SQR(positive-numeric-expression)

SQR returns the square root of X, negative X values will cause error 107.
For Example:-

A = SQR(9) puts the value 3 into A.

3.0.99. STOP

Type:- Statement

Availability:- Not available in Immediate Mode

Syntax:- STOP

The program terminates and a STOP AT LINE message will be displayed. The program can be restarted from just after the STOP statement with a CONT command. STOPs and ENDs are optional, and may be conditional. STOPs are useful for diagnostic purposes.
For Example:-

```
250 IF X=10 THEN STOP ELSE PRINT X
```

If X = 10 the program will display STOP AT LINE 250 and will STOP.

3.0.99. STR$(X)

Type:- String Function

Availability:- All

Syntax:- STR$(numeric-expression)

STR$ turns a numeric expression into a string, the string is constructed exactly as it would be printed.
For Example:-

```
10 A = -.999 : B = 3.678
20 PRINT "X";STR$(A);"X";STR$(B);"X"
```

This prints:

```
X-.999 X 3.678 X
```

3.0.99. STRING$

Type:- Function

Availability:- All

Syntax:- STRING$(numeric-expression [, character])

STRING$ creates a single character string of the specified length, of either the specified character or spaces by default. The character may be specified either as a string or as its ASCII decimal value.
For Example:-

```
10 PRINT STRING$(30)          Prints 30 spaces
20 PRINT@(10,5)STRING[$(30,"*")]
                              Prints 30 asterisks
30 PRINT STRING$(30,42)       Prints 30 asterisks, 42 is the
                              ASCII decimal code for
                              asterisks.
```

3.0.99. SWAP

Type:- Statement

Availability:- All

Syntax:- SWAP variable1, variable2

SWAP exchanges the values of two variables of the same type. Virtual array elements cannot be SWAPped directly. SWAP is used when sorting.
For Example:-

```
                              10 SWAP A$,B$
```

This exchanges the contents of the two strings.

3.0.99. SWI

Type:- Function

Availability:- All

Syntax:- SWI (intno[para1[,para2[,para3]]]

SWI acts identically to USR except that it calls a software interrupt function in the POLY unit system ROM. A full list of these and their parameters is given in Section 4.

Also see USR.
For Example:-

```
10 Z% = SWI(30)
20 Z% = SWI(31)
30 CLS
40 PRINT "This is all double height"
```

Software interrupt 30 selects a 12 line screen, 31 selects the top half. CLEAR, RUN and CHAIN all restore the 24 line screen.

3.0.99. TAB

Type:- Function

Availability:- All

Syntax:- TAB(numeric-expression)

TAB is used in PRINT statements to move the cursor to column specified. If the cursor is the column then the cursor does not move. Commas and semi-colons mean the same as in any PRINT statement. TAB will work with PRINT# statements. TAB included in any other string statements will cause an error.

For Example:-

```
5 X=8.9 : Y=222222
10 PRINT TAB(10),X,Y,"RESULTS"
```

This displays:

```
          8.9    222222  RESULTS
```

The 8 is in column 17. The TAB moves the PRINT position to column 10, the "," moves it on to 16, and there is a leading blank before the number.

```
20 A$="AAA"+TAB(3)
30 PRINT A$
```

This will not work as line 20 contains an illegal expression, ERR = 73.

3.0.99. TAN(X)

Type:- Function

Availability:- All

Syntax:- TAN(numeric-expression)

This returns the TANgent of the numeric expression which is expressed in radians. To convert degrees to radians multiply by PI/180.
For Example:-

```
T=TAN(45 * PI/180)
```

will assign the value 1 to the variable T.

3.0.99. TEXT$

Type:- String Function

Availability:- All

Syntax:- TEXT$ (row, column[,length])

TEXT$ returns the block of data on the current Text screen starting at the specified row, column of the specified length. If the length is not given, a single character is returned.
For Example:-

```
10 A$ = TEXT(20,0,40)
```

places row 20 on the screen in A$.

3.0.99. TROFF

Type:- Command

Availability:- All

Syntax:- TROFF

This turns the TRace OFF. See TRON.

3.0.99. TRON

Type:- Command

Availability:- All

Syntax:- TRON

TRON turns the TRace ON. The trace displays the line numbers as the program is executed. This is useful during debugging. When the trace is on, text will scroll out of sight but fine graphics will not move. Use the <PAUSE> key and space bar to examine the execution of the program line by line.
For example, the screen might appear as follows:

```
<1500>
<1501>
<1502>
<400>
<401>
<402>
<1503>
```

The program has gone to and returned from a subroutine at line 400.

3.0.99. USR

Type:- Function

Availability:- All

Syntax:- USR(address [,[para1][,[para2][,[para3]]]
or USR (stringvar[,[para1] etc

USR calls a machine language subroutine stored at the specified address, or stored in the named string. Up to 3 parameters may be handed over to the subroutine. Each parameter, if not specified, defaults to value -1 (hex FFFF). Each parameter may be an integer value in the range -32768 to 65536.

Parameter 1 is placed in the D register.
Parameter 2 is placed in the X register.
Parameter 3 is placed in the Y register.

The return value must be placed in the D register. If the subroutine wishes to flag an error, the carry flag must be set and the error number returned in the D register. This causes an error to occur in the BASIC.
For Example:-

```
Z% = USR(A$, 48, B%)
```

calls a subroutine stored in A$ and hands over two parameters, 48 and B%. The return value is placed in Z%.

To place the subroutine in Z%, it may either be set up using DATA statements and the string formed, or be loaded from disk using LOAD.

```
DATA 10,20,47,126,130,48,57
FOR I = 1 TO 7
READ A%
A$ = A$+CHR$(A%)
NEXT I
```

or

```
LOAD "SUBA.DAT" AS A$
```

For Example:-

```
Z% = USR(32823)
```

calls a subroutine stored at 32823 with no parameters.

3.0.99. VAL

Type:- Function

Availability:- All

Syntax:- VAL(string-expression)

VAL takes a numeric string and converts it to its numeric VALue. The first non numeric character (other than +, -, or .) will cause the rest of the string to be ignored. All spaces are ignored. It is often safer to INPUT a number as a string, test its length and then use VAL to convert it into a numeric, otherwise the number typed in may be too long.
For Example:-

```
A = VAL("-2.3")
B = VAL("9ABC678")
C = VAL("r123")
```

These lines assign A = -2.3, B = 9, C = 0

3.0.99. WAIT

Type:- Statement

Availability:- All

Syntax:- WAIT length

WAIT suspends the program for the specified number of 10 millisecond intervals.ON KEY and ON SEC interrupts do not occur during a WAIT.
For Example:-

```
100 WAIT 100
```

suspends the program for 1 second.

4. SOFTWARE INTERRUPT FUNCTIONS.

The number on the left-hand-side of each description below is the "Software Interrupt Function number". The BASIC command "SWI(number,parameter)" calls the relevant Software Interrupt Function. This function is performed before the program continues.

For example:
To check the status of the keyboard, insert the code

```
X%=SWI(0)
```

at the appropriate position in your program.

Some Software Interrupt Functions require another one or more parameters to be specified.
For example:

```
X%=SWI(2,65,Z%,40)
```

REMEMBER,
WITH EACH SWI, A 2-BYTE VALUE IS RETURNED.

0. Check Status of Keyboard.

This function checks to see if a key has been pressed on the keyboard.

Input Parameters: None
Value Returned:
Only bits 0 and 7 of the second byte are used.
If bit 0 = 1, a keyboard character is waiting to be read for the first time.
If bit 7 = 1, the key is still depressed.
All other bits are zero.

2. Line Edit.

The facilities offered by the line edit function are:
initialise, insert character, delete character, scroll up, scroll down, bell, cursor left, cursor right, cursor down, cursor up, clear screen, clear to end of line, normal and reverse video, carriage return (<ENTER> key), print characters.
To perform "Edit" functions, the "SHIFT" and "CAPS LOCK" keys must not be depressed.

Input Parameters:
Parameter 1: The character to be processed.
Parameter 2: The start address of the line buffer.
Parameter 3: The maximum buffer length allowed.
Value Returned
None.

10. Set Relative Cursor Position on Current Teletext Screen.

This function moves the cursor by the specified number of positions, relative to its current location.

Input Parameters:

Parameter 1: A number in the range -23 to +23
which is the number of rows that the cursor is to be moved.
Positive is down, negative is up.
Parameter 2: A number in the range -39 to +39
which is the number of columns that the cursor is moved.
Positive is to the right, negative is to the left.

Value Returned:
None.

16. Read Display Mode.

This function reads the current screen display mode.

Input Parameters:
None.

Value Returned:
A 16-bit integer defined as follows:
bit 15 - not used
bit 14 - mix/priority bit: 1=mix, 0=priority
bit 13 - 1=Display 2 (Graphics 1 screen)
bit 12 - 1=Select 5, 0=Select 2
See Bit 13 for displaying these.
bit 11 - 1=Display 1 (Teletext 1)
bits 10 & 9 - Select 2 mix colour
00=blue, 01=green, 10=red,11=none
bits 7 & 8 - not used
bits 6 5 & 4 - Select Background colours
The colour is defined for teletext characters.
i.e.
bit 6 (4) - BLUE
bit 5 (2) - GREEN
bit 4 (1) - RED
bits 3 & 2 - Select Graphics 2 mix colour as above for Graphics 1
bit 1 - 1=Display 4
bit 0 - 1=Display 2

20. Set Timer Registers.

This function allows access to the registers in the MC6840 programmable timer module in the Poly. Typical uses would be for timing a signal on the user port or for using the three timers to generate output signals for Polyphonic music. Each register is one byte in length and may be accessed individually in any order.

Input Paramters:
Parameter 1: The first byte is zero.
The second byte contains the register number between 0 and 10.
0-7 refer to the 8-bit registers.
8-10 refer to the Timer Count Registers 1-3 as 16-bit values.
Parameter 2: The data to be written into the selected register.

Value Returned:
None.

21. Read Timer Registers.

This function is the complement of function 20 and is used to read the current byte values of the timer registers.

Input Parameters:

Parameter 1: The number of the register to be accessed (0-10).

Value Returned:

The value of the specified register.
If this is a single-byte value, bits 8-15 are zero.
An error is returned if an attempt is made to read a register numbered greater than 10.

22. Set User Port.

This function is used to set the User Port high or low.

Input Parameters:

Parameter 1: Zero if the User Port is to be set low.
If non-zero, the User Port is set high.

Value Returned:

None.

23. Read User Port.

This function reads the User Port.

Input Parameters:

None.

Value Returned:

Zero if User Port is set low.
1 otherwise.

24. Search for Tape Header and Return First Byte.

This function searches for the tape header start sequence of $89AF (decimal 35247) which is used at the beginning of each record on the tape to achieve data synchronisation. When $89AF is encountered, it clears the CRC and reads the next byte (which is usually the first byte of the record.)

Input Parameters:

None.

Value Returned:

The Ascii value of the byte read from tape.

25. Read Byte from Cassette Tape.

This function reads a single byte from the cassette tape input port. It updates the CRC checksum in memory. As the data on the tape is in synchronous format this function assumes that synchronisation has been achieved prior to its calling.

Input Parameters:

None.

Value Returned:

The Ascii value of the byte read from tape.

26. Write Byte to Cassette Tape.

This function writes 8 bits of data to the cassette tape.
Input Parameters:
The Ascii value of the byte to be written.
Value Returned:
None.

31. Select 24-Line Display for Text Screens.

Calling this function sets both text screens to display 24 lines.
Input Parameters:
None.
Value Returned:
None.

32. Select 12-Line Display for Text Screens.

Calling this function sets both text screens to display 12 lines.
Functions 33 & 34 determine whether the top 12, or bottom 12 lines are to be displayed.
Input Parameters:
None.
Value Returned:
None.

33. Display First 12 Lines.

When used in conjunction with 32 above, this function displays the top 12 lines (lines 0-11) on each text screen.
Input Parameters:
None.
Value Returned:
None.

34. Display Last 12 Lines.

When used in conjunction with 32 above, this function displays the bottom 12 lines (lines 12-23) on each teletext screen.
Input Parameters:
None.
Value Returned:
None.

36. Remote Control Chip Decoder.

This function enables the decoding of information provided by a variable-pulse-width remote-control encoding circuit connected to the User Port. Any number of channels may be used. Contact Polycorp for details of wave forms required.
Input Parameters:
Parameter 1: The address of the start of the table into which the values read from the recorder are to be placed.
Value Returned:
Each channel read results in two bytes being stored. These two bytes form an integer in

the range of approximately 0 to 3000.

5. APPENDICES

5.1. ERROR MESSAGES

NUMBER	MEANING
1	Illegal file request
2	Requested file is in use
3	File already exists
4	File could not be found
7	All disc space has been used
8	End of file error
9	Disk file read error
10	Disk file write error
11	File or disk is write protected
12	File is protected
15	Illegal drive number specified
16	Drives not ready
20	Unbalanced parentheses
21	Illegal file specification or illegal character
22	File close error or source not present
23	Too many files on disk or line too long
24	Non existent record number specified or syntax error on encode
25	Record number match error or file damaged
26	Error in command
30	Invalid syntax or data type mismatch
31	Invalid syntax in function or out of data in READ
32	Invalid character at line start or bad argument in ON statement
33	Invalid statement start
34	Invalid statement terminator or break key pressed
35	Label expected
36	Numeric result expected
37	String result expected
38	"(" expected
39	"," expected
40	")" expected or bad file number specified
41	Missing or invalid item in expression or file already open
42	Mixed mode or OPEN must have OLD or NEW
43	Too many temp strings or file has not been OPENed
44	Subscript negative or out of range or file status error
45	Incorrect number of subscripts or field size too large (>252)
46	Undimensioned array reference or can't extend a sequential file
47	Expression result <0 or >255 or RECORD 0 not allowed
48	Must use random type file
50	RETURN without GOSUB
51	NEXT without FOR
52	RESUME not in error routine
53	Cannot continue
54	Line not found
60	Arith overflow
61	Too large to convert to integer

```
 62   LOG of 0 or negative number
 63   SQR of negative number
 64   Division by 0
 65   Argument too large
 70   Argument out of range
 71   Out of data in READ
 72   Data type mismatch in PRINT USING
 73   Illegal format in PRINT USING
 74   PRINT@(Y,X) outside screen area
 75   Bad argument in SWAP
 76   Illegal parameter in SWI or USR call
 77   Array already dimensioned
 78   FN function not defined
 79   Dimension negative or too large
 80   Clock not running
 81   No room for stack
 82   Memory set too low or high
 83   No room for new string or array
 84   CRC error
 85   Verify error
 86   No trailer rec
100   Invalid data on input
101   Number too large for integer
102
 Number too large
```

5.2. TELETEXT SCREEN CONTROL CHARACTERS

ASCII Decimal Value		Representation In Strings	Function
0		@ or space	Not used
1		A or a	Starts RED characters
2		B or b	Starts GREEN characters
3		C or c	Starts YELLOW characters
4		D or d	Starts BLUE characters
5		E or e	Starts MAGENTA characters
6		F or f	Starts CYAN characters
7	*	G or g	Starts WHITE characters
8		H or h	Starts FLASHING
9	*	I or i	Ends FLASHING
10		J or j	Not used
11		K or k	Not used
12	*	L or l	Normal height
13		M or m	Double height **
14		N or n	Start ASCII characters
15	*	O or o	Ends ASCII characters
16		P or p	Reverse video on
17		Q or q	Starts RED graphics
18		R or r	Starts GREEN graphics
19		S or s	Starts YELLOW graphics
20		T or t	Starts BLUE graphics
21		U or u	Starts MAGENTA graphics
22		V or v	Starts CYAN graphics
23		W or w	Starts WHITE graphics
24		X or x	CONCEAL display on rest of line
25	*	Y or y	Contiguous graphics
26		Z or z	Separated graphics
27	*	← or ;	Reverse video off
28	*	½ or <	No background to characters
29		→ or =	Set background to current colour
30		↑ or >	Print graphics characters over control characters
31	*	# or ?	Print space for control characters

EACH CHARACTER MUST BE PRECEDED BY A ↑ .

To include a single ↑ in a print string use ↑↑ .
Each of the control characters take up ONE screen position. The reverse video on and off characters do not take up screen space. All control characters are reset at the beginning of each line to those with an * beside them. Reverse video is switched off at the end of each PRINT.
The control characters in strings are always converted to the first of the two options listed above, i.e. "↑a" is converted to "↑A".
** Double height may be used on screen 1 but not screen 3. Double height characters extend down to the following line. If double height is used anywhere on a line the following line is not displayed. Anything printed on screen 3 "behind" a line containing a double height character will be displayed in normal height on both rows.

5.3. ASCII SCREEN CONTROL CHARACTERS

ASCII Decimal Value		Function
0		Not used
1	*	Insert character
2	*	Delete character
3		Not used
4		Not used
5	*	Scroll up
6	*	Scroll down
7		BEEPs the speaker
8		Moves cursor 1 space to the LEFT
9		Moves cursor 1 space to the RIGHT
10		Moves cursor 1 space DOWN
	*	and scroll
11		Moves cursor 1 space UP
	*	and scroll
12		Clears screen and moves cursor to HOME position (0,0)
13		Moves cursor to start of screen line
14		Not used
15		Starts TELETEXT characters
16		Reverse video on
17		Not used
18		Not used
19		Not used
20		Not used
21		Not used
22		Not used
23		Not used
24		Not used
25		Not used
26		Not used
27		Reverse video off
28		Not used
29		Not used
30		Clear to end of line
31	*	Initialise line editor

NOTE

To use these, they must be preceded by Teletext control character 14 and followed by Teletext control character 15.

The * values are only available if the LINE EDITOR has been set on using SWI(2).

5.4. SPECIAL FUNCTION KEYS

Function Key	ON KEY Value	ASCII Decimal value
Numeric keypad 0	0	48 *
1	1	49 *
2	2	50 *
3	3	51 *
4	4	52 *
5	5	53 *
6	6	54 *
7	7	55 *
8	8	56 *
9	9	57 *
Numeric keypad .	26	46 *
EXIT	10	26
PAUSE	11	28
ENTER	12	13
NEXT	13	19
REPEAT	14	20
BACK	15	21
HELP	16	06
CALC	17	
◄	18	08
►	19	09
▼	20	24
▲	21	25
CHAR INS	22	01
CHAR DEL	23	02
LINE INS	24	17
LINE DEL	25	18
SHIFT/PAUSE	27	27
@	28	64
£	29	35
↑ or EXP	30	94
|	31	124

The function keys marked with * may be assigned "soft key" values by using ON KEY <exp> = <exp>

e.g. ON KEY 4 = 16

This assigns a new ON KEY value, key 4 now "looks like" key 16.

The HELP and CALC keys cannot be tested by checking their ASCII values. To disable or trap these keys use ON KEY.

5.5. SOUND FREQUENCIES AND THE MUSICAL SCALE

This scale has the A above middle C defined as having a frequency of 440 Hz.

Note	Frequency (Hz)	N1
G	784	80
F	698	90
E	659	95
D	587	106
C	523	120
B	494	127
A	440	142
G	392	159
F	349	179
E	330	189
D	294	213
C(middle)	262	239
B	247	253
A	220	284
G	196	319
F	175	357
E	165	379
D	147	425
C	131	477
B	123	508
A	110	568

To use the POLY to make sounds use the SOUND statement.

SOUND <N1> , <N2>

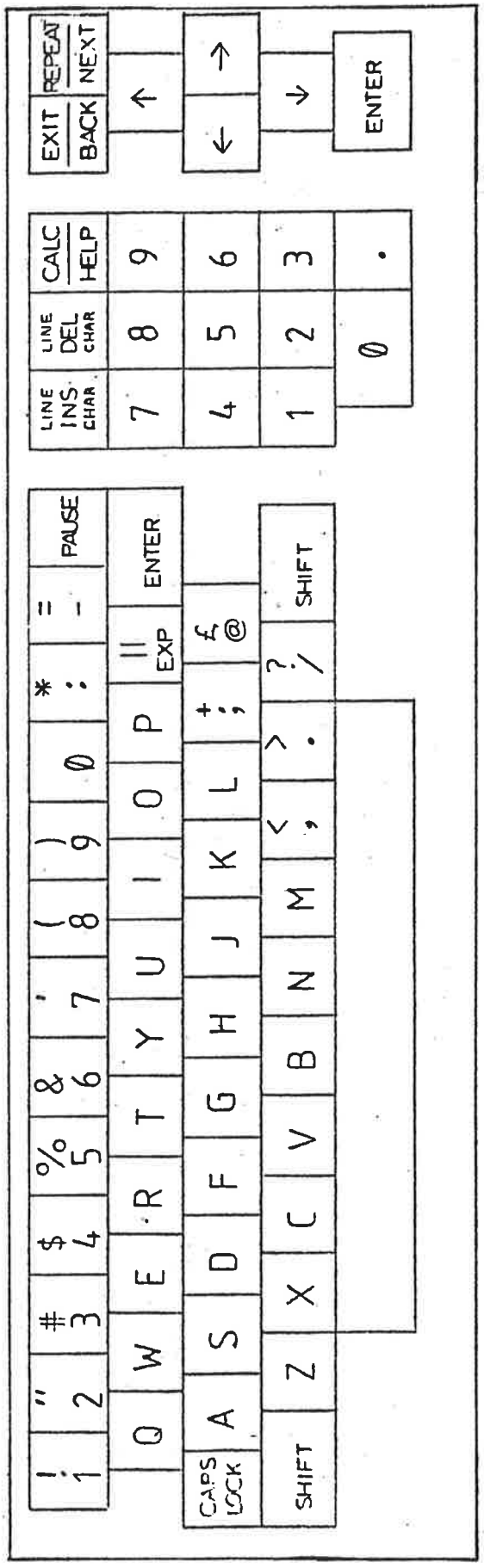
N1 = 62500/frequency and is tabulated above.

N2 is the duration of the sound in 10 ms intervals.

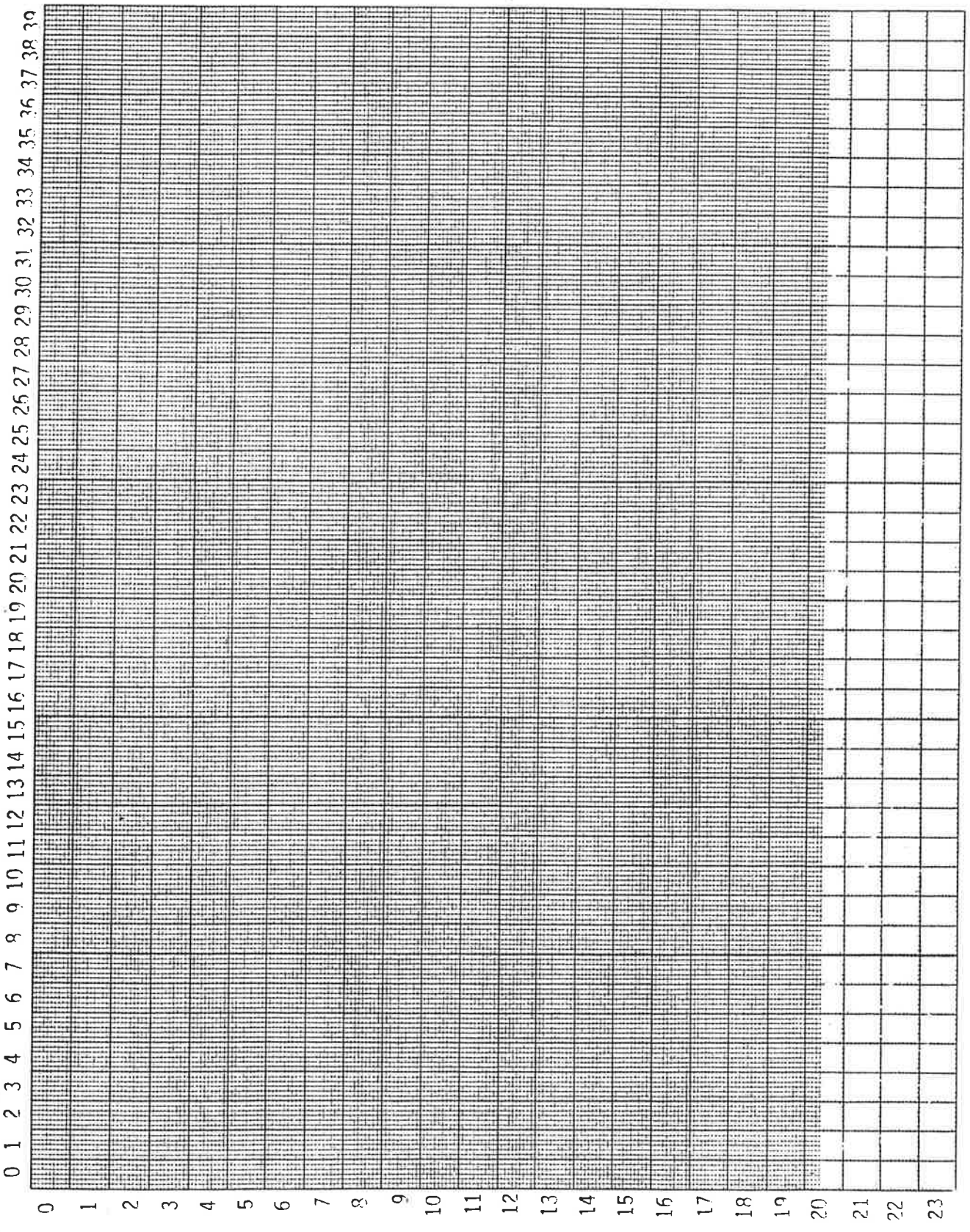
5.6. TELETEXT CHARACTERS AND GRAPHICS

ASCII DECIMAL VALUE	CHAR-ACTER	GRAPHICS	ASCII DECIMAL VALUE	CHAR-ACTER	ASCII DECIMAL VALUE	CHAR-ACTER	GRAPHICS
32	SPACE		64	@	96	-	
33	!		65	A	97	a	
34	"		66	B	98	b	
35	£		67	C	99	c	
36	$		68	D	100	d	
37	%		69	E	101	e	
38	&		70	F	102	f	
39	'		71	G	103	g	
40	(		72	H	104	h	
41	)		73	I	105	i	
42	*		74	J	106	j	
43	+		75	K	107	k	
44	,		76	L	108	l	
45	-		77	M	109	m	
46	.		78	N	110	n	
47	/		79	O	111	o	
48	0		80	P	112	p	
49	1		81	Q	113	q	
50	2		82	R	114	r	
51	3		83	S	115	s	
52	4		84	T	116	t	
53	5		85	U	117	u	
54	6		86	V	118	v	
55	7		87	W	119	w	
56	8		88	X	120	x	
57	9		89	Y	121	y	
58	:		90	Z	122	z	
59	;		91	←	123	¼	
60	<		92	½	124	‖	
61	=		93	→	125	¾	
62	>		94	↑	126	¯	
63	?		95	#	127	■	

5.7. DIAGRAM OF POLY KEYBOARD



5.8. SCREEN LAYOUT CHART



5.9. RESERVED WORDS

Words which are the names of POLYBASIC commands, statements or functions must not be included in variable names. Spaces are ignored by BASIC.

e.g. IF T1<TANDT1>0 GOTO100

This statement will cause an error because it contains the reserved word TAN.
The following table is a list of reserved words.

ABS	END	LOGOFF	RENAME
AND	EOF	LPRINT	RENUMBER
AS			
ASC	ERL	LPOFF	RESET
ATN	ERR	LPON	RESTORE
AUTO	ERROR	LSET	RESUME
BACKG	EXEC	MEM	RETURN
CHAIN	EXP	MID$	RIGHT$
CHR$	FETCH	MIX	RND
CLEAR	FIELD	MOD	ROTATE
CLOAD	FILE$	MOVE	RSET
CLOCK	FILL	NAME$	RUN
CLOSE	FN	NEW	SAVE
CLS	FOR	NEXT	SCALE
COLOR	FRE	NOT	SCROLL
COLOUR	GET	OLD	SELECT
		ON	SEC
COMPILE	GOSUB		SET
CONT	GOTO		SGN
CONVERT	HEX		SIN
COS	IF		SOUND
CSAVE	INCH$	OPEN	SPLIT
CVER	INPUT	OR	SQR
CVT		PEEK	STOP
			STORE
DATA		PI	STR
DATE$		POKE	STRING$
DEF	INSTR	POINT	SWAP
DELETE	INT	POS	SWI
	KEY		
	KILL		
DIGITS	KVAL	PRINT	TAB
	LDES$		
DIM	LEFT$		TAN
DISPLAY	LEN		TEXT$
			THEN
DIV	LET		TIME$
DOS	LINE		TROFF
DPEEK	LIST	PTR	TRON
DPOKE		PUT	UNLOCK
			USING
DRAW	LOAD	RANDOM	USR
DRIVE	LOCK	READ	VAL
ELSE	LOG	REM	WAIT