

## CONTENTS

---

0. INTRODUCTION	1
I. GETTING STARTED WITH XBASIC	
1. First steps - Direct & Program mode	3
2. Numbers and Strings	4
3. Variables	5
4. Arrays	6
5. Expressions	6
II. THE SYSTEM EDITOR & SYSTEM COMMANDS	
1. Screen Control Codes	9
2. The Screen Editor	9
3. The Line Editor	10
4. System Commands:	10
MON, NEW, DEL, AUTO, LOAD, SAVE, VERIFY CLEAR, RUN, CHAIN, LIST, HOLD, MGE, RENUM	
5. Chaining and 'Semi-Chaining' Programs	15
III. COMMANDS, STATEMENTS AND FUNCTIONS	
1. Commands/Statements	16
2. Disc Handling Commands:	21
DIR, ERA, REN, LOCK, UNLOCK	
3. Standard Functions	22
4. Standard String Functions	25
5. User-Defined Functions	27
IV. INPUT/OUTPUT FACILITIES	
1. Devices and I/O assignment:	28
PRINT\$, INPUT\$	
2. Direct I/O Port access	30
3. Special Commands affecting I/O:	30
SEP, FMT, IOM, SPEED, NULL, WIDTH, ZONE	
V. XBASIC FILE MANAGEMENT SYSTEM	
1. General	34
2. File Naming Conventions	34
3. The File Descriptor	35
4. Sequential and Random Access Methods	36
5. File-Handling Commands:	37
DRIVE, OPEN, CREATE, CLOSE, APPEND, PRINT\$, INPUT\$, INCH\$	
6. File-Handling Examples	39
VI. ERROR MESSAGES	
1. List of Error Messages	45
2. Error handling within BASIC:	47
ON ERR GOTO/GOSUB, ON EOF GOTO/GOSUB, OFF ERR, OFF EOF, ERR, ERR\$, ERL	
3. Error Message Construction/Extension	48
VII. MACHINE-CODE LINKAGE	
1. MC-code related Commands/Functions:	50
CALL, POKE, PEEK, DOKE, DEEK, PTR, HEX\$	
2. Loading and Saving MC-code Files	52

<b>VIII. COMMAND/FUNCTION EXTENSION</b>	
1. Program storage	53
2. Reserved Word Construction	54
3. The Auxiliary Tables	55
4. Commands and Functions	55
5. How to enter extra Reserved Words	55

#### **APPENDICES**

---

Appx. A.	Index of Reserved Words and Error Messages	58
Appx. B.	The Hardware Configuration, including: MEMORY MAP, SCRATCH-PAD addresses I/O Device assignments, Graphics, incompatibilities with other versions, etc	61
Appx. C.	Useful Subroutines in XBASIC	65
Appx. D.	Examples of Extra Commands and Functions	76
Appx. E.	Translator for Nascom ROM and tape Basic	81

## 0. INTRODUCTION

Nascom Extended Basic (XBASIC) is an interpreter written in Z80 machine code which has been developed by Crystal Research. It is based on experience gained with earlier versions of Xtal BASIC and Nascom ROM BASIC. Extended BASIC is significantly larger than both the earlier Xtal BASIC and Nascom ROM BASIC, but includes many new features, and existing features have been extended.

For those with some experience of machine-code programming, the ability to create user-defined reserved words must be one of the most outstanding features of this BASIC. By writing appropriate sub-routines and by inserting your own defined words in an auxiliary reserved word table, you will be able to expand this interpreter to give the type of BASIC most suited to your own needs. We believe that, for the time being at least (and we have not heard of any equivalent in over two years), this feature is unique to BASIC's from Crystal, and it makes it potentially one of the most powerful BASIC's ever available.

Extended BASIC is designed to allow the incorporation of disc handling commands, as well as handling cassette tape, and the file handling system has been designed with a view to dealing with both. Although we use the terms 'disc' and 'cassette tape' throughout the manual, it is as well to remember that some forms of tape, such as the 'stringy floppy' or 'floppy tape' are theoretically capable of random-access, and may hold separate 'file directories', i.e, to all intents and purposes they behave as disc drives. We therefore include all such devices under the broad term 'disc drives', to distinguish from the sequential-only 'cassette tape' drives.

Nascom Extended BASIC is available in three forms - tape cassette, NAS-DOS and CP/M. The differences involve only the media and the provision of appropriate disc access commands.

### LOADING EXTENDED BASIC ON NASCOM MICROCOMPUTERS (TAPE VERSION)

---

XBASIC will run on any of the Nascom computers, as long as one of the NAS-SYS monitors is being used. It is supplied on tape in CUTS format, to load at 1200 baud.

To load XBASIC, type R and then press the <ENTER> key. Next, press the PLAY button on the cassette recorder. The program should be observed to load block by block until, after about two minutes, loading should be complete. XBASIC occupies the area 1000H to 40FFH (about 12 1/4 K).

To run, type in E1000 and press the <ENTER> key. This is the initialising, or 'COLD', entry to XBASIC. A 'WARM' entry is also allowed from the monitor into XBASIC by typing E1003 <ENTER>. This preserves any current BASIC program and variables. This entry point should not, however, be used unless XBASIC has already been previously entered by a COLD start.

## NOTATION

---

In order to simplify the use and understanding of this manual and, in particular, the command and function descriptions, we have adopted a notation that explains the syntax requirements of each command/function. This consists of a single letter, which may or may not be followed by a number, enclosed thus: < >. If a command/function name has to be followed by an expression, this notation will show the type of expression that is allowed:

- J     An expression, which must evaluate to a number in the range 0 to 255. If it is not an integer, the decimal part of the number will be chopped off, the integer part only being used.
- I     An expression, which must evaluate to a number in the range -65535 to +65535. Again, only the integer part is actually used. In some cases, the range is restricted to 0 to 65535, or even 0 to 32767 (e.g., Array elements), but mention is made only when those cases apply.
- L     A line number, in the range 0 to 65535. This must be a number only, and so may not be given as a variable.
- N     Any numeric expression.
- E     Any expression, whether numeric or string.
- S     Any string expression.
- F     A string expression, which must evaluate to give a legal file name (as defined in Chapter V.2).
- U     A numeric variable, which may not be an array element.
- V     A variable name, which may be of numeric or string type, and may be an array element.
- SV    A string variable name, which may not be a string array element.
- X     A complete Xtal BASIC statement.

### Examples:

1. In Chapter III.4 we find the LEFT\$ function described thus: LEFT\$(<S>,<J>)

This means that LEFT\$ must have two arguments separated by a comma, and enclosed within parentheses. The first argument must be a legal string expression, and the second argument must be a number in the range 0 to 255 (reasonable, since we cannot have strings longer than 255 characters).

e.g. LEFT\$("NAME "+X\$,7) is legal.

2. In Chapter III.1, the ON..GOTO command is shown:

ON <J> GOTO <L1>,<L2>,...,<Ln>

This means that ON must be followed by a number in the range 0 to 255 followed by the word GOTO followed by one or more line numbers <L1> to <Ln>. Each of these line numbers (if more than one) must be separated by a comma.

e.g. ON X GOTO 1000,2000,3000 will simply drop to the next line if X is 0 or greater than 3, otherwise a GOTO will be executed to one of lines 1000, 2000 or 3000 according to the value of X being 1, 2 or 3 respectively.

## I. GETTING STARTED WITH XBASIC

## 1. RUNNING UP XBASIC

Having loaded XBASIC from your tape or disc, you should be rewarded with the 'sign-on' message, i.e:

```
Nascom Enhanced BASIC Rev xx
(C)1982 Xtal
Size: yyyyy
Ok
```

where xx represents the sub-version for your machine, and yyyyy the memory size available for storage of BASIC program and variables. The Ok prompt shows that XBASIC available and is not running a program, but is waiting for a command to be typed in at the keyboard. Virtually any of the commands or statements listed in the following chapters may be typed in and executed, along with a number of special commands given in Chapter II, known as 'SYSTEM' commands. For example, we may use the machine as a calculator:

```
Example:
PRINT ATN(1)*4      You type this line
3.14159             Computer prints the result of 4 times the arc-
                    tangent of 1
```

This is known as DIRECT execution mode, since commands are executed DIRECTly they are typed.

Alternatively, commands and statements may be entered without being executed, by typing a line number in front of it. A sequence of one or more lines entered in this way forms a PROGRAM, which may be executed by means of the RUN command (see below). This is known PROGRAM mode.

Line numbers may range between 1 and 65535, and may be followed by one or more commands. Each line so entered is automatically placed in order, with line 1 at the front. Line numbers may be selected arbitrarily by the user, but it is recommended that reasonable gaps be left (say, 10) between lines, so that extra lines may be inserted if these are later found to be necessary, in the development of a program. The program may begin with any line number, but the first line to be interpreted will always be the lowest line number entered.

A line may be deleted by entering its number followed by <ENTER>, with no other information.

Several commands may be entered on a single line by separating them with colons:

```
Example:
10 PRINT 2*13 : PRINT 5+6      You type these
RUN                             lines
26                             Computer responds with the answers
11
```

Separation in this manner allows several commands to be entered in DIRECT mode as well as in a program.

## 2. NUMBERS AND STRINGS

---

There are two types of quantity allowed in XBASIC - numbers and strings. Numbers may also include floating-point numbers, integers, and hexadecimals.

### 2.1 Numbers

These can be whole numbers (integers) or floating-point numbers (reals). A number is stored internally as four bytes, one of which represents a signed exponent, while the other three represent a signed mantissa. This gives an exponent range from -38 to 38, with a seven-digit signed mantissa. Although the full seven digits are available for internal calculation, they are rounded off to six figures when output, the seventh figure being known as a 'guard' digit. When accuracy is at a premium, the seventh digit should always be used, if known, since Xtal BASIC can make use of it, even though only six significant figures are displayed.

Example:

The PI function actually uses 3.141593, even though it displays as 3.14159 (to show this, try PRINT PI-3).

Leading and trailing zeroes are suppressed on output, so that integers are actually printed as such without long rows of zeroes.

Examples:

3 3.14159 314.159 .0314159 3.14159E+08 -3.14159E-37

These are all possible forms in which numbers may be output. The last two, for those not familiar with them, are in SCIENTIFIC notation, a form only used when the output is too large or too small to be conveniently printed in any other way. Numbers may be INPUT in this form, if required.

### 2.2 Integers

XBASIC supports 16-bit integers, i.e., whole numbers in the range -32768 to +32767. Integers outside that range may only be stored in ordinary numeric variables (see next section, on variables), but integers in this range may be stored internally in two bytes instead of four (for simplicity of design of the interpreter, we actually store integers in four bytes for simple variables, and two bytes per element within arrays, where the greatest savings may be made).

We use an additional convention with integers, however, that numbers in the ranges -65535 to -32769, and 32768 to 65535, may be accepted by integer variables, since it is often useful to do so. In these cases, the values are internally converted to lie in the ranges 1 to 32767 and -32768 to -1 respectively (since we should otherwise need 17 bits to store such numbers).

### 2.3 Hexadecimal numbers

The allowance of hexadecimal numbers in XBASIC greatly increases the ease of linkage to machine-code routines and locations, and they may normally be used in any expressions requiring numeric quantities. The only limitations are that only integers are allowed for, in the range &0 to &FFFF. To indicate a

hexadecimal number, a leading ampersand '&' symbol is supplied, followed by a string of characters, each of which may be a number in the range 0 to 9, or a letter in the range A to F. When encountered within a numeric expression, a hexadecimal number is internally converted into a decimal integer and the result of such an expression will still always be a normal number. The hexadecimal number may consist of 1 to 4 digits. More may be entered, but all except the last four will then be ignored.

Examples:

&1F34 represents 7988 in decimal.

&A7 represents 167 in decimal.

&91F34 still represents 7988 in decimal (the first digit is ignored).

To obtain the hexadecimal equivalent of a decimal number, the HEX\$ function may be used (see Chapter VII.1).

## 2.4 Strings

These are combinations of ASCII characters representing letters, numbers and symbols, useful for storing names, titles and text, although their intrinsic data can be extracted by the interpreter and they are frequently used to hold numeric values as well. A string can be any combination of up to 255 characters, usually shown in quotes " " in order to prevent confusion with numbers or variables.

Examples:

"TREVOR" "Trevor" "12345.6" "Oh! \*%" are all valid strings.

## 3. VARIABLES

---

A variable is a combination of letters and/or numbers, the first character being a letter. XBASIC distinguishes the first FIVE characters (most BASICs distinguish only the first two). Variables may be of either numeric, integer or string type, and hold numbers, whole numbers and strings respectively. Integer variables must be suffixed with a '%', and string variables with a '\$'. There is no theoretical limit to the length of a variable name, although the length of the input line will clearly limit it!

Examples:

A AA X9% Z9% X\$ F4\$ ABCD\$ AB123\$ KRAZY KRAZY10

are all valid variables, although BASIC would be unable to distinguish between the names of the last pair, since their first five characters are the same.

Care must be taken to ensure that variable names do not contain reserved words, otherwise SYNTAX ERRORS will result.

Examples:

TONE LETTER COST EXPENSE PINCH TERROR TO LET COS  
EXP INCH ERR (and OR)

will all cause problems.

Keeping variable names to two characters will solve this problem (the only 2-character names in XBASIC are IF, TO and FN) and this also saves space.

Integer variables may contain only integers in the range -65535 to +65535, but they may also contain hexadecimal numbers. However, values returned by integer variables are in the range -32768 to +32767, using the most-significant bit as a SIGN flag (these may be regarded as sixteen-bit numbers).

Example:

```
%=65535: PRINT %      will display the value -1
LOC=&9678: PRINT LOC   will display the value 38520
LOC%=&9678: PRINT LOC% will display the value -27016
```

#### 4. ARRAYS

---

In addition to simple numeric and string variables, we can use numeric and string arrays. An array is, in effect, a table full of variables, each of which can be uniquely identified. Naming of arrays takes exactly the same form as for simple variables, except that they are followed by a set of one or more subscripts, each subscript representing one of the dimensions of that variable.

Examples:

```
A(0) TABLE%(5,6) NAME$(1,0,2)
```

are all valid arrays, where A is an array of one dimension, the subscript (0) referring to the FIRST element. TABLE% is a two-dimensional integer array and NAME\$ is a three-dimensional array holding strings, each of which may be up to 255 characters in length.

In XBASIC all array subscripts number from zero.

In order for BASIC to know how much space to allocate to an array, the array in question must be dimensioned with a DIM statement (see Chapter III.1) before being brought into use. However, if all subscripts in an array have maximum values of 10 or less, then that array may be used without a DIM statement.

Example:

```
AA(7,4,6)=56
```

will dimension that array exactly as though the following had been written:

```
DIM AA(10,10,10):AA(7,4,6)=56
```

assuming that no previous DIM statement has been used for AA. This array will have 11\*11\*11=1331 elements, requiring over 5320 bytes to store it! NOTE: If we had used an integer array A%, we should only require about 2670 bytes to store it.

#### 5. EXPRESSIONS

---

Expressions consist of variables, numbers, string variables or strings in any combination, and related by means of arithmetic and/or logic operations.

## 5.1 Arithmetic operators

The arithmetic operations allowed in XBASIC are as follows:

+	(add)	-	(subtract)	*	(multiply)	/	(divide)
↑	(raise to power)			MOD	(remainder)		

The  $\uparrow$  operator has the following conventions:

$X \uparrow 0 = 1$  for  $X >= 0$ , and  $0 \uparrow Y = 0$  for  $Y > 0$  (so  $0 \uparrow 0 = 1$ !).  
 $X \uparrow Y$  is undefined for  $X < 0$  or for  $X = 0$  and  $Y < 0$ .

The MOD operator is the remainder from a division, and can be defined as follows:

$$X \text{ MOD } Y = X - Y * \text{INT}(X/Y)$$

Examples:

5 MOD 3 returns 2

-5 MOD 3 returns 1 (see definition of INT, Chapter III.3).

## 5.2 Relational and logical operators

RELATIONAL operators are used for comparisons and the evaluation of conditions, particularly for IF statements. The ones allowed are:

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

LOGICAL operators allowed are:

NOT	AND	OR	XOR (exclusive-OR)
-----	-----	----	--------------------

Example:

10 IF (X+Y-Z)>3 AND Y<=20 THEN 100

Expressions involving relational operators and logical operators are normally used within IF statements (Chapter III.1), but can also be used within normal arithmetic expressions, since a relational expression returns a value -1 if it is TRUE, and 0 if FALSE. In some cases, quite a lot of space can be saved.

Example:

IF X>15 THEN A=0: ELSE A=1      can be replaced by:  
 A= -(X>15)

## 5.3 Bit manipulation

Logical operators may also be used for bit manipulation, provided that the sub-expressions on either side evaluate to results in the range -65535 to 65535 (i.e., they can be thought of as sixteen-bit quantities). Then AND, OR, XOR and NOT will all work upon the individual respective bits of the two expressions.

Example:

PRINT 1234 AND 3412      outputs the result 1104:  
 0000 0100 1101 0010 (1234 = &04D2)  
 0000 1101 0101 0100 (3412 = &0D54)  
 0000 0100 0101 0000 (1104 = &0450)

## 5.4 Operator precedence

Operator precedence follows the usual mathematical order. We also include the relational and logical operators here, so that they may be used within arithmetic expressions with the correct precedence:

Highest precedence:	( )	(parentheses).
	$\frac{a}{b}$	
	* / MOD	
	+ -	
	< <= = <> >= >	
	NOT	
	AND	
Lowest precedence:	XOR OR	

## 5.5 String expressions

XBASIC also allows string expressions, but the only operator is **CONCATENATION**, represented by '+'.  
 C

Example:  
 A\$="ABC": B\$="DEF": C\$=A\$+B\$: PRINT C\$  
 outputs the result  
 "ABCDEF"

String comparison may also be performed for alphabetic sorting, since "B">"A", for example. In comparing two strings, the comparison is done character by character, until a position is found in which the two differ. The 'greater' string is then the one whose character has the greater ASCII code. If no differences are found, but one string is longer than the other, the longer string is considered to be the greater.

Examples:  
 "GOLIATH" is greater than "DAVID"  
 "ANDY" is greater than "ANDREW"  
 "BROTHERHOOD" is greater than "BROTHER"  
 "Hello" > "Goodbye" returns the numeric result -1 (true).

## II. THE SYSTEM EDITOR AND SYSTEM COMMANDS

### 1. SCREEN CONTROL CODES

---

The following VDU control codes are used by XBASIC on the standard 48x16 VDU. Note, incidentally, that all 16 lines scroll when running XBASIC.

Ctrl-A	&01	HOME cursor to top left corner of screen.
<TAB>	&09	TAB cursor to next print ZONE, by printing spaces. However, see also IOM command in Chapter IV.3.
<LF>	&0A	LINE FEED, or move cursor DOWN. Scroll screen at bottom.
<CS>	&0C	CLEAR SCREEN and Home cursor to top left corner.
<CR>	&0D	CARRIAGE RETURN, without line feed.
Ctrl-P	&10	PRINT SCREEN to printer (device #1, see Chapter IV.1).
Ctrl-Q	&11	Move cursor LEFT.
Ctrl-R	&12	Move cursor RIGHT.
Ctrl-S	&13	Move cursor UP.
Ctrl-T	&14	Move cursor DOWN (same as <LF>).

### 2. THE XBASIC EDITOR

---

This powerful facility, available to you the moment that XBASIC is entered, has been designed in an attempt to make program entry and debugging more of a pleasure rather. Input lines may be up to 127 characters long, and note is kept at all times of where the start and finish of the line is. So, if you have several lines in a listing, you may move the cursor up the screen to that line and make modifications to it, even if it occupies two or more rows on the screen. If the line is extended so that it will apparently run into the next one, the lines below simply move down one row to make room for it. Note that the modified line is only entered into the program when the <ENTER> key is pressed while the cursor sits in one of the rows of the screen containing that line.

The following special key functions are available, the ones in brackets indicating the equivalents for the Nascom 1 keyboard:

Ctrl-A (@A)	HOME cursor to top left corner of screen.
s<BS> or <CS>	CLEAR screen and Home cursor.
' ' (@R)	Move cursor RIGHT.
' ' (@Q)	Move cursor LEFT.
' ' (@S)	Move cursor UP.
' ' (@T)	Move cursor DOWN (scroll screen at bottom).
<BS>	DELETE character to the LEFT, but moving rest of line one place to the left.
s' ' (@U)	DELETE character from the RIGHT, moving rest of line one place to the left.
s' ' (@V)	INSERT space at cursor, moving rest of line one place to the right, and moving lines below it one row down, if required. NOTE: An insertion done at the bottom line of the screen will cause an immediate scroll, moving the cursor up with it. This has no ill effects, apart from being a bit disconcerting when first observed.

Ctrl-W (@W) ERASE whole line. This differs from Ctrl-X in that the cursor is returned to the start of the line before clearing it.

Ctrl-X (@X) ERASE to end of line (even if it occupies 2 or more rows), the current cursor position.

Ctrl-O (@O) ERASE to end of screen from current cursor position.

Ctrl-P (@P) PRINT SCREEN contents to printer.

<ESC> (s<NL>) Abandons a line (though you could just use an arrow key or a Ctrl-W!) and prints the 'Ok' prompt.

<CR> or <NL> ENTER the current line on which the cursor sits into BASIC. cursor will end up sitting at the start of the next line (i.e, not necessarily the next ROW of the screen). Leading and trailing spaces are ignored, and lines of greater than 127 characters will be truncated to 127 (this being the size of the buffer area).

If this is all as clear as mud(!), the best thing to do is to 'play'!

### 3. THE LINE EDITOR

---

In addition to the screen editor, a 'Line edit' mode is also available, primarily for use within programs, when to use the screen editor could cause some irritation (since the INPUT prompt would also be assumed to be part of the input line! On the other hand, this could also be very useful in certain applications).

In this mode, cursor movement keys are not available, except that ' ' and <BS> both delete the last character from the line, Ctrl-P still gives a screen dump to printer, <ESC> abandons the line, and <CR> enters it into BASIC.

For the reasons outlined above, screen edit mode is 'switched on' automatically in direct mode, and LINE EDIT mode turned on for programs. In addition, the user may use the IOM command (see Chapter IV.3), inside or outside a program to change the editing mode: IOM 0,1 gives SCREEN EDIT mode, IOM 0,0 gives LINE EDIT mode. In direct mode, IOM 2,0 should be used before IOM 0,0, otherwise screen edit mode will be reselected on completion of the statement.

LINE EDIT mode shows itself by means of a prompt at the start of the line (']' in direct mode and '?' in an INPUT statement with no specified prompt string).

SPECIAL NOTE: In spite of the declaration above that lines are limited to 127 characters in length, it is possible to move the buffer area to other areas in the memory space, and to change the buffer length up to 254 characters maximum! This may be done by means of the PTR command (see Chapter VII). Care must then be taken over selection of the area used to contain the buffer, and it is recommended that an area created by means of a CLEAR command be used.

### 4. SYSTEM COMMANDS

---

The following commands are normally intended for use in direct mode, although some (such as RUN and LIST) can also be used to advantage within programs, and CHAIN is used almost entirely within programs. Because they all affect modification and overall control of programs and of the system, they are all referred to as SYSTEM commands:

**MON** Takes control back to the operating system, or the monitor. This is the command to use when you wish to leave X BASIC.

**NEW** Causes all program lines and variables, if any, to be deleted.

**DEL <L1>,<L2>** Deletes all lines from the program in the range <L1> to <L2>. Both start and finish lines should be specified, but will default to 10 if not given! If <L1> is larger than <L2>, or if <L1> is larger than the largest line present, a RANGE ERROR will occur.

Example:

DEL 100,199 deletes all lines with numbers from 100 to 199 inclusive

**LIST <I1>,<I2>,<I3>** Lists the program to the current output device. The listing starts from the first line after <I1> and ends at line <I3> or the first line after <I3> if that line is not present. <I2> gives the number of lines to list at a time. After <I2> lines have been listed, there will be a pause. The user then presses a key, and the listing continues with another <I2> lines (except for some special keys, given in note (iii)). Any or all of the expressions may be omitted, but the appropriate commas should be present if <I2> and/or <I3> are specified.

Examples:

LIST	Lists whole program
LIST ,5	Lists whole program, 5 lines at a time
LIST 100,7	Lists 7 lines at a time starting from 100
LIST 200	Still lists 7 lines at a time, starting from line 200
LIST 100,,199	Lists 7 lines at a time from 100 to 199
LIST ,4,299	Lists 4 lines at a time from the start to line 299
LIST ,,199	Lists 4 lines at a time from the start to line 199
LIST 300,5,999	Lists 5 lines at a time from 300 to 999

Notes:

(i) BASIC remembers the last value of <I2> given and keeps using it until LIST is used with a different value. When BASIC starts up, <I2> is assumed to be 65535, until given.

(ii) A listing may be abandoned at any time, whether paused or not, by pressing <ESC>.

(iii) When paused, the cursor movement keys may be used to abandon the listing and, at the same time, move the cursor in the direction of the key pressed. This only works in SCREEN EDIT mode (see section 3 of this Chapter). The purpose of this is to allow quick exit to the Editor, without having to remember to press <ESC> first!

(iv) Unlike many BASIC's, the LIST command may be used within a program as a normal statement, and note also that <I1>, <I2> and <I3> may all be EXPRESSIONS. This can be extremely nice!

Example:

X=100: Y=50: LIST X,Y,X+99      Lists from line 100 to 199, 50 lines at a time.

**AUTO <L1>,<L2>** Automatic line-numbering while entering programs. This command requires a start line <L1> and increment <L2>, and both of these default to 10 if not given.

Examples:

AUTO 100,5        Starts from line 100 and continues 105, 110, 115, etc.  
 AUTO 100         Starts from line 100 and continues 110, 120, 130, etc.  
 AUTO             Starts from line 10 and continues 20, 30, 40, 50, etc.

Each line number is displayed just as if it had been typed from the keyboard, and the user may enter the usual program statements at that point. On pressing <CR> in that line, the text is entered with its line number in the usual way, and then the next line number appears. The user then continues with this line. When finished, just type <ESC> to abandon, whereupon normal direct mode will be re-entered. Any error (BRANCH ERROR is common, when the user just presses <CR> without entering any statement and the line does not exist) will also cause a return to normal direct mode. The editing mode is not affected by this command.

**LOAD <F>** Loads a file from tape OR disc (depending which is available, either if both are available) whose file name is <F>. The file name convention is described in full in Chapter IV.4, so the user is referred to that.

Examples:

LOAD "TEST"      Loads the program file "TEST.XBS" from the current default disc or tape drive. Any existing program in memory is deleted, but note that variables are NOT destroyed.

LOAD "B:TEST.ASC"   Loads the ASCII program file "TEST.ASC" from disc drive B, whatever the current default drive. In this mode, the user may actually observe the file loading, appearing line-by-line on the screen. Again, variables are NOT destroyed, but neither is the existing program. Thus the user may add extra routines to existing programs, and the added lines will appear at their correct positions in relation to those already present. Note, however, that if a new program is to be loaded as a .ASC file, a NEW command must first be executed.

LOAD "T:ROUTINES.OBJ"   Loads the machine-code routines or data from the file "ROUTINES.OBJ" on tape drive T, into the area previously reserved for them in the memory map (by means of the CLEAR command). The start address will be assumed to be the first location above this CLEARED area (e.g, if a CLEAR &9FFF has been done, the file will load starting at &A000). See also Chapter VII.2.

In all three cases, if the size of the file is larger than the area available, a MEM FULL ERROR will occur. If the file is not present, a NO FILE ERROR will occur if a disc drive is being searched, while no result will be returned if a tape drive is being searched - the user simply has to abandon the tape load, as explained in Appendix B.

If a type other than XBS, ASC or OBJ is specified, a FILE TYPE ERROR will occur (this also applies to SAVE. If it is desired to load or save data files, use the file access commands described in Chapter V).