Luxor

Manual Basic II

ABC 800®

Luxor
Computers

Preface

This manual describes the BASIC II programming language of ABC 800. The reader should have programming experience, since the manual is not meant to be a BASIC textbook.

Chapter 1 introduces the BASIC programming language. Chapter 2 deals with the structure of BASIC II computer programs.

Chapters 3, 4, and 5 describe the data that can be processed by a program.

Chapter 6 describes how to operate BASIC II. This chapter contains plenty of advice and tips on how to type and edit a program

Chapter 7 deals with the direct usage of instructions and commands, without any program. This method is particularely useful when a program is being debugged.

Chapters 8, 9, and 10 contain detailed descriptions of all commands, functions, and instructions that are part of BASIC II. Most of the descriptions are completed by examples which show the structure of each program part.

Chapters 11 and 12 deal with the ABC 800 graphics. Both the TELETEXT graphics and the high resolution graphics with animation mode are described.

Chapter 13 describes the use of the function keys.

Chapter 14 describes the differences between ABC 800 and ABC 80.

Chapter 15 contains a list of error messages with comments.

Chapter 16, marked with grey edges, contains short descriptions of all instructions, functions, and commands arranged in alphabetical order.The list in chapter 16 is meant for use as an index register, where the syntax can be found together with references to the detailed descriptions earlier in the manual (chapters 8, 9, and 10).

Chapter 17 is a list of literature references and chapter 18 contains a number of appendices. The last chapter of the manual is an alphabetical index.

N.B.
Differencies in the BASIC programming language used for ABC 802 and ABC 806 are indicated in the margin as ABC 802 and ABC 806, respectively. The applicable text explanations are found in appendix 5 (for ABC 802) and appendix 6 (for ABC 806).

Contents

1 BASIC - the Programming Language

A formal language - a programming language - is used to give instructions to the computer. The formal language consists of certain key words in English.

BASIC is a very simple programming language. Each instruction, command and function is easy to understand and to use. Nevertheless, the language is comprehensive enough to allow versatile and efficient solutions to most problems.

The name BASIC stands for Beginners' All-purpose Symbolic Instruction Code. BASIC was originally designed for elementary programming education. However, the language turned out to be so efficient that now it is used in a wide variety of applications.

The concept behind many programming languages is that the whole program is fed to the computer which translates it into the appropriate machine language; the program is compiled. The compiler program looks for formal errors in the user program. The computer then prints a list of the errors. The programmer corrects the program and feeds it once more to the computer. The program is compiled and the errors - if any - will be printed.

On the other hand, when you program in BASIC, a BASIC interpreter program is resident in the computer. The interpreter checks every program line as soon as you have written it. A formal error will result in an immediate error message on the screen. You may run the program at any time to test the parts of it that you have written. This is called interactive programming and is in many cases the most efficient way of programming a computer.

Naturally, interactive programming does not solve all problems. When the formal errors have been eliminated from the program, logical errors may still remain which can only be detected when the program is executed with the proper data.

Like any other language, BASIC has grammatical rules. The grammar of a programming language is much simpler than the corresponding rules of a natural language. The example below shows a program which computes the mean value of five numbers, given by the user. Here you can see the structure of the language.

Example

```
10 INPUT A,B,C,D,E
20 LET S=(A+B+C+D+E)/5
30 PRINT S
40 END
```

BASIC II contains all the elementary instructions needed for simple programs as well as the instructions and functions which make possible the writing of more advanced programs with greater efficiency. The key word in this kind of programming is efficiency. As the programmer gains more experience, his efficiency increases and he will want to use more advanced data processing. BASIC II is comprehensive enough to solve virtually any problem.

BASIC II offers an AUTOSTART function, which is described in detail in the disk drive manual.

2 The Program

A program consists of program lines containing statements. The statements contain instructions for the BASIC interpreter. Every program line begins with a unique line number. The line number is followed by one or more BASIC statements. Two statements in the same line should be separated by a colon (:). The line numbers indicate the order of execution. Each statement begins with a key word, which indicates the operation to be performed.

The statement gives an instruction to the computer (in this example **PRINT**):

```
30 PRINT S
```

Some instructions need operands to specify on which variable or which part of a program the instruction should operate. The operand in the example above is "S".

The last statement in a program is the **END** statement.

```
10 INPUT A,B,C,D,E
20 LET S=(A+B+C+D+E)/5
30 PRINT S
40 END
```

The **END** statement, which tells the computer that the program is finished, is not mandatory. The **END** statement should be the last instruction that is executed. When an **END** statement is executed, all files are closed but the variables still have their current values.

2.1 Line Numbers

Every program line starts with a line number. Some of the effects of the line number are given here:

1. Denotes the order of execution. The statements may be written in any order.
2. Makes possible changes in the normal order of execution by means of the instructions **GOTO, GOSUB** etc. The line number serves as a label and a jump address.
3. Makes it possible to alter (edit) any line without affecting the rest of the program.

You, the programmer, chose the line numbers. Any integer from 1 up to and including 65 535 may be used.

Each line must have a unique line number. The computer uses the line numbers to identify and keep track of the instructions. If a new line is written with an existing line number, this new line will replace the existing one.

The statements may be entered in any order. The computer will arrange them by their line numbers. If you write e.g. the lines 30, 10, 20 in that order, the computer will rearrange them :10, 20, 30.

The lines should be numbered by fives or tens so that new statements can be easily inserted. There are commands for automatic line numbers (**AUTO**) and for renumbering (**REN**).

2.2 Comments

A comment or remark can be denoted in two ways in BASIC II; by means of the standard **REM** statements or with the text preceded by an exclamation point.

1. **REM** statement (according to the BASIC standards)

```
10 A=7: REM seven
```

2. Exclamation point. Does not require a colon.

```
10 A=7 !seven
```

Remarks are part of a BASIC program. They are printed when the program is listed on the screen or printer. These comments are not executed. Any character (except RETURN) can be used in a remark. The remarks are usually marked with some clearly visible character, so that you will notice them in the program.

```
10 REM ***** Data from transducer *****
200 GOSUB 3100 ! Search for peak value
3240 RETURN !&&&&&& X7=peak value &&&&&
```

NOTE:

A comment cannot be terminated by a colon. The colon is treated as part of the remark.

```
150 REM **** Calculations **** : LET R1=3.52E2.1+Y5
```

The last statement will not be executed. The entire line is considered to be a non-executable comment.

2.3 BASIC Statements

The line begins with a line number, then follows a BASIC statement. The key word of the statement identifies the statement type. The BASIC interpreter is thereby informed as to which operation to perform and how to treat the data - if any - that follow the key word.

The user is allowed to write more than one BASIC statement on a single line. These statements must be separated by a colon. A line consisting of several statements is executed a little faster than the same statements if they are each written on one line. A shorter execution time can be important in some applications.

```
100 PRINT A,B,C
```

is an ordinary, single program line

```
200 LET X=X+1 : PRINT X : IF Y=1 THEN 100
```

is a multiple statement line containing the three statements **LET**, **PRINT** and **IF-THEN**

As a rule, any statement can be used anywhere in a multiple statement line. The exceptions to the rule have been explicitly specified in the descriptions of the instructions.

NOTE: It is good programming practice to write only one statement on each line.

2.4 Expressions

An expression is a group of symbols that represent constants, variables, functions or a combination of these separated by arithmetic, relational or logical operators. Examples:

Arithmetic expressions

```
4.123
3%+A%
B6*(C**3+1.0)
```

Relational expressions

```
X > Y
Y8 >= 0
A = B
```

Logical expressions

```
(A< 1.0) AND (B=5)
((B<A) OR (D=C)) AND B/A< >D/C
```

Arithmetic expressions yield either floating point or integer values.

Relational expressions yield a truth or false value that reflects the result of a comparison of two values.

Logical expressions yield a truth or false value that reflects the existance or nonexistance of conditions.

String expressions are explained in chapter 5.

2.5 Logical Units

BASIC II ensures independence from physical input/output devices through the use of file numbers.The file number can be treated as a logical unit and is handled with the instructions **OPEN, PREPARE** and **CLOSE.** The file number may for instance represent a printer or a file on a tape cassette/flexible disk.

Example:

```
10 - -
20 OPEN "PR:" AS FILE 2 ! Open the printer
30 - -
40 - -
50 CLOSE 2 ! Close the printer
60 END
```

NOTE: **CON:** is the standard device. **CON:** stands for console (keyboard and screen).

2.6 Error handling

Certain errors can be detected by BASIC when it executes a program. These errors can for instance be computational errors (such as division by 0) or input/output errors (reading an end-of-file code to an **INPUT** statement). Normally, the occurrence of any of these errors will cause termination of program execution and the printing of a diagnostic message.

Some applications may require that program execution continues after an error has occurred. To accomplish this, the user can include an **ON ERROR GOTO** <line number> statement in the program. The program will then jump to the user's error handler which begins at the specified line number. The error handler will analyze the error.

The **ON ERROR GOTO** statement should be placed before all the executable statements, with which the error handling routine deals.

When an error occurs in a program, BASIC checks to see if the program has run through an **ON ERROR GOTO** statement. If no such statement has been encountered, a message is printed at the screen and the program execution is terminated. If an **ON ERROR GOTO** statement was run through, program execution will continue at the line number specified by that statement. The error handler at that line number can e.g. test the function **ERRCODE** to find out precisely what error has occurred and decide what action is to be taken.

If there are portions of the program in which any errors detected are to be processed by the system and not by the error handler of the program, the error handler can be disabled by executing the following statement:

line number **ON ERROR GOTO**

The computer will then attend to all errors as it would do if no **ON ERROR GOTO** <line number> had ever been executed.

The error handling routine is terminated by a **RESUME** statement. The function of **RESUME** resembles the one of the **RETURN** statement at the end of an ordinary subroutine. The program jumps to the entry point - if any - in the statement that caused the error. If the program execution should continue at another line number, the line number in question should be given in the **RESUME** statement.

Example of error handling:

```
10 ON ERROR GOTO 100 !At error go to line 100
20 INPUT "Age, Weight " A,W
30 ON ERROR GOTO !Disable the error handler
40 STOP
100 PRINT !Error handler
110 PRINT " Erroneous input! "
120 RESUME !Jump to line 20
```

3 Data

3.1 Range of Values

Floating Point

The range of values for floating point is the largest range of values in BASIC.
±1E-38 ±1E+38

There are seven significance digits in single (**SINGLE**) and sixteen digits in double precision (**DOUBLE**). All numbers are rounded internally to fit this precision.
Numbers may be entered and displayed in three formats:

Example: 153, 34.52, 136E-2

Integers

The range of integer values is:
-32768 through 32767 inclusive.

Character strings

A character string can contain any number of characters.

NOTE
Strings used in string arithmetic have a maximum size of 125 characters including the sign and the decimal point.

3.2 Constants

Numeric constants retain a constant value throughout a program. They can be positive or negative. Numeric constants can be written as follows:

```
Example:      +3%
              -4.765
              12345.6
              -.0001
```

The three last constants of this example would be stored as floating point, since they have no % suffix. The use of an explicit decimal point or percent sign is recommended in all numeric constants to avoid unnecessary data conversion and to improve documentation.

3.3 Variables

A variable is a data item the value of which can be changed during program execution. A variable is denoted by a specific variable name.

Variable names consist of a single letter or a single letter followed by a single digit. It is possible, by means of **EXTEND**, to use long variable names (letters and digits, starting with a letter).
These characters are allowed:

A,B,C,...,Z	(letters)
0,1,2,...,9	(digits)

A name can also have an **FN** prefix (denoting a function name), a . suffix (denoting floating point), a % suffix (denoting integer), a $ suffix (denoting string), or a subscript suffix that consists of a set of subscripts enclosed by parentheses.

A string expression has a value that consists of a sequence of characters, each character occupying one byte. A string expression can be expressed either as a sequence of characters enclosed by quotation marks or as a variable using a variable name with a $ suffix.

Mixing of data types in a statement should be avoided. Use integers whenever possible. Integers need less storage space and are processed faster by the computer.

The same name can appear in combination with various prefixes and suffixes in the same program and generate mutually independent variables. For example, the floating point variable A is entirely different from the integer variable A%. The name A can be used as follows:

A	floating point variable A
A%	integer variable A%
A$	string variable A$
A(d)	floating point array A with dimension specification d
A%(d)	integer array A% with dimension specification d
A$(d)	string array A$ with dimension specification d
FNA	floating point function A
FNA%	integer function A%
FNA$	string function A$

In the **EXTEND** mode a name can be used as follows:

Signal	floating point variable Signal
Signal%	integer variable Signal%
Signal$	string variable Signal$
Signal(d)	floating point array Signal with dimension specification d
Signal%(d)	integer array Signal% with dimension specification d
Signal$(d)	string array Signal$ with dimension specification d
FNSignal	floating point function Signal
FNSignal%	integer function Signal%
FNSignal$	string function Signal$

Variables are assigned values by **LET**, **INPUT**, and **READ** among other statements. The variables are set to zero before program execution, unless they have been protected by a **COMMON** statement. It is necessary to assign a value to a variable only when an initial value other than zero is required.

3.4 Subscripted Variables and the DIM Statement

In addition to the simple variable, the use of subscripted variables (arrays) is allowed. Subscripted variables provide the programmer with additional computing capability for dealing with lists, tables, matrices, or any set of related variables. Variables are allowed with two numbers of subscripts.

The name of a subscripted variable is any acceptable variable name followed by a number of integers enclosed by parentheses. For example, a list may be described as A(I) where I goes from 0 to 5:
A(0), A(1), A(2), A(3), A(4), A(5)
This allows the programmer to refer to each one of the six elements in the list, which can be considered a one-dimensional algebraic vector as follows:

A(0)
A(1)
A(2)
A(3)
A(4)
A(5)

A two-dimensional matrix B(I,J) can be defined in a similar manner and displayed graphically:

B(0,0)	B(0,1)	B(0,2)	...	B(0,J)
B(1,0)	B(1,1)	B(1,2)	...	B(1,J)
B(2,0)	B(2,1)	B(2,2)	...	B(2,J)
B(3,0)	B(3,1)	B(3,2)	...	B(3,J)
....	....	....	......	....
B(I,0)	B(I,1)	B(I,2)	...	B(I,J)

Matrix structure

Subscripts used with subscripted variables may have integer values only. If a subscript is a floating point value, it will be rounded off to an integer.
A dimension (**DIM**) statement is used to define the maximum subscripts of an array. If a subscripted variable is used without a **DIM** statement, the maximum value of each subscript is assumed to be 10. It is possible to change the minimum value for array subscripts by means of the statement **OPTION BASE.** Normally, the minimum value is 0 but it can be changed to 1, so that a standard array would have 10 elements in each dimension instead of 11. All **DIM** statements should be placed at the beginning of the program.

3.5 File Storage

BASIC II provides facilities for the definition and manipulation of data on cassette or disk.

A data file consists of a sequence of data items transmitted between a BASIC program and an external input/output device. The external device can be a printer, a cassette, or a disk. The **OPEN** statement specifies the devices available and their references. The device has a name by which it is identified within the system (DR0: for disk drive 0).

Each data file is identified by a unique name; the file name. For example, ABC123.BAC is the name of a disk file. The file is accessed internally in the user program by means of its file number. The file number is given in the program by means of one of the instructions **PREPARE** or **OPEN.** These statements will open the file, i.e. set up a channel for the data transfer. To close such a data transfer channel the instruction **CLOSE** is used. The instructions **INPUT** and **PRINT** or **GET** and **PUT** are used for the data transfer.

A buffer area is created by the system when a file is opened. All data transfer to and from a file is buffered.

3.5.1 Opening a File

To open an existing file the **OPEN** statement is used. If the file is new, it should be opened with a **PREPARE** statement.

Example

```
10 OPEN 'FILE1.AAA' AS FILE 1
```

opens the existing file named FILE1.AAA for input/output with file number 1.

3.5.2 Data Transfer To/From a File

The transfer of data takes place directly between the internal channel (the file number) and the string variable or the value of the expression in question. All data transfer refers to either one byte or one character string (the characters followed by a carriage return).

The following instructions can be used:

INPUT £ reads a value to a variable or a string from the position of the file pointer to a carriage return

INPUT LINE £ reads a value to a string variable including the carriage (CR) return and line feed (LF)

PRINT £ writes the contents of a variable into the file

GET £ COUNT reads one byte or the given number of bytes from the position of the file pointer

PUT £ writes one record into the file

POSIT £ moves the file pointer to the desired position

If no file number is given in the **GET** statement, it will attempt to read from the keyboard. If the **COUNT** option is not used, **GET** will read one byte, i.e. one character.

Example

```
20 GET £1,D2$ COUNT 6%
```

will read six characters from the file with file number 1 from the position of the file pointer. The characters are put in the string D2$.

The instruction **POSIT** is used to position the file pointer at the given position in the file. The number of characters always refers to the beginning of the file (position 0). **POSIT** can be used together with any one of the other file handling instructions.

Example file 1 contains ABCDEFGHIJK

```
50 POSIT £1,5
60 GET £1,A$ COUNT 3
70 PRINT A$
80 END
RUN
FGH
```

The function **POSIT**(file number) reads the position of the file pointer. In the example above, **POSIT**(1) has the value 8, when the example has been executed. **POSIT** returns a floating point value, and can thus operate on long files.

WARNING **POSIT** should not be used in conjunction with sequential files, i.e. files which are handled by **PRINT** and **INPUT/INPUT LINE.** If you want to use **POSIT** with a sequential file, every **PRINT** statement should be followed by a **GET** statement, else an end-of-file (EOF) mark will be written at the position of the file pointer, the next time that **POSIT** or **CLOSE** are used. A dummy **GET** looks like this:

```
40 GET £2,Q$ COUNT 0
```

When **POSIT** is used with sequential files, you should look upon **PRINT** and **GET** as a sequence of instructions that belong together.

3.5.3 Closing a File

The data transfer to or from a file will not be correctly terminated until the file is closed. The contents of the buffer area are then transferred, and the file is given an end-of-file (EOF) mark.

There are two ways of closing a file:

CLOSE 2 closes the file with file number 2

CLOSE closes all files

4 Integers and Floating Point Numbers

Normally, all numeric values (variables and constants) specified in a BASIC program are stored internally as floating point numbers. For integer numbers significant economies in storage space can be achieved by the use of the integer data type. Also, integer arithmetic is faster than floating point arithmetic. A constant, variable or function can be specified as an integer by ending its name with the % character.
Example: A%, **FN**X%(Y), -8%, Z3%

The user always has to specify with the % character that an integer is to be generated, otherwise a floating point value will be produced.

When raising to an integer power, the power value should be explicitly indicated as an integer.

The computer will act as described above when BASIC II operates in its normal mode, i.e. **FLOAT.** The default value can be changed to integer by the **INTEGER** instruction.

4.1 Mathematical Operations

When more than one operation is to be performed in a single formula, certain rules are observed as to the precedence of the operators. The arithmetic operations are performed in the following sequence, where the operation described in item 1 has precedence:

1. Any formula within parentheses is evaluated first. The parenthesized quantity is then used in further computations. Where the parentheses are nested as follows:

   (A+(B*(C**3)))

   the innermost parenthesized quantity is calculated first.

2. In the absence of parentheses the following precedence is performed:

   a. Intrinsic or user defined functions
   b. Exponentiation (**)
   c. Multiplication and division (*, /)
   d. Addition and subtraction (+, -) and unary minus
   e. Relational operators (=, <>, >=, <, <=, >)
   f. **NOT**
   g. **AND**
   h. **OR** and **XOR**
   i. **IMP**
   j. **EQV**

   Thus, for example, -A** B with a unary minus is a legal expression and is the same as -(A** B). This implies that -2** 2 evaluates as -4. A**-B is not allowed, but A**(-B) is allowed.

3. In the absence of parentheses, operations on the same level are performed left to right, in the order that the formula was written.

4.2 Integer Arithmetic

All arithmetic with integer values is performed in modulo 2** 16. A BASIC integer can be between -32 768 and +32 767 inclusive. The integer representation can be regarded as a continuous circle with -32 768 following +32 767.

Integer division forces truncation of any remainder. However, the function **MOD** makes the remainder available.

Examples: 3%/4%=0% and 283%/100%=2%

When an operation is performed on both integer and floating point data, the result is stored in the format indicated by the resulting variable.

Example:

```
LET B%=Z%+3/X
```

The result is rounded to give B% an integer value.

4.3 Input/Output of Integer and Floating Point

Input and output of integer variables is performed in exactly the same manner as the corresponding operations on floating point variables.

Any number, which can be represented by up to seven significant digits in **SINGLE** mode or sixteen digits in **DOUBLE** mode, is printed without use of the exponential form.

Any floating point variable that has an integer value, is automatically printed as an integer but is internally still a floating point number.

If more than seven/sixteen digits are generated during any computation, the result will automatically be printed in the format:

[-].nE[-]m

where n is a number with seven digits, at the most, and m is an exponent with one or two digits.

Input allows all the formats used for output. When a floating point value is assigned to an integer variable, the value is rounded off to an integer.

4.4 User Defined Functions

An integer function is defined as being of integer type by the % suffix following the function name.

Example

```
10 DEF FNC%(X%)=X%*(Z%+X%)
```

A floating point function can be written like this:

Example

```
10 DEF FNV(X%)=X%*(Z+X%)
```

4.5 Integers as Logical Variables

Integer variables or integer valued expressions can be used within **IF** statements in any place where a logical expression can appear. Any non-zero value is defined as being true and an integer value of 0% corresponds to the logical false value. The logical operators **(AND, OR, NOT, XOR, IMP, EQV)** operate on logical (or integer) data in a bitwise manner.

NOTE:
The integer -1% is normally used by the system for a true value. Logical values generated by BASIC II always have the values -1% (true) and 0% (false).

4.6 Logical Operations on Integer Data

BASIC II allows a user program to combine integer variables or integer valued expressions using a logical operator to give a bitwise result.

The truth table below is valid for the logical operations. A is the condition of one bit in one integer value and B is the condition of the corresponding bit of another integer value.

A	B	A **AND** B	A **OR** B	A **XOR** B	A **EQV** B	A **IMP** B	**NOT** A
1	1	1	1	0	1	1	0
1	0	0	1	1	0	0	0
0	1	0	1	1	0	1	1
0	0	0	0	0	1	1	1

The result of a logical operation is an integer value generated by the combination of the corresponding bits of two integer values according to the rules shown above.

The result of any logical operation can be assigned to an integer or a floating point variable.

Example:

```
10 A%=13% OR 14%
20 PRINT A%
RUN
15
```

AND, OR, XOR, EQV, IMP, and **NOT** can operate on variables and valued expressions to give a bitwise integer result. If logical operations are done on float variables or float valued expressions, conversion to integer format is done before the execution of the logical operation.

5 Character Strings

BASIC II not only processes numerical information but also information in the form of character strings. A character string is a sequence of characters.

5.1 String Constants

Character string constants are allowed, just like numerical constants. The character string constants are delimited by either single (') or double (") quotes. If the delimiting character occurs twice in a string sequence it is considered to be part of the text.

The value Let's can be expressed in two ways: "Let's" or 'Let''s'.

5.2 String Variables

Any legal variable name followed by a dollar sign ($) character is a legal name for a string variable.

Example A$, B1$ are simple string variables.
A$(8), G5$(M,N), J$(I) are subscripted string variables.

NOTE
The same name, without the $, denotes a numeric variable, which can be used in the same program.

Example F, F$, and F% are allowed in the same program.

5.3 Subscripted String Variables

The **DIM** statement is used to define string arrays and string matrices. The following alternative **DIM** statements are available:

Example

DIM W$(2,4)=8 !String length 8; maximum subscript values 2 and 4
DIM R5$(9,9) !String length up to 80; maximum subscript values 9 and 9
DIM NAME$(7,6,3,2)=10 !String length 10; four-dimensional matrix with maximum subscript values7,6,3, and 2

5.4 String Size

The length of a non-dimensioned string variable is automatically set to the current length the first time that the string is assigned a non-null value (< >"").

If less than 80 characters are used, the string length will be the default value of 80 characters.

Each string, scalar or vector element, has two lengths:

1. The maximum length is the number of bytes allocated to the string.
2. The current length is the number of bytes currently in use. The current length may vary between zero and the maximum length. The current length is the only visible length; this length may be examined by the function **LEN.**

If a string is assigned a null value (=""), the current length will be set to zero. No further action is taken.

If a string is assigned a non-null value and has a non-zero max length, the string length is checked. If the string length is sufficient, a number of bytes will be allocated to store the data and the current length will be set to the number of allocated bytes. If the string length is not sufficient, an error message will be written.

5.5 String Functions

Various functions are used with character strings. These functions allow a program to perform arithmetic operations on numeric strings, concatenate two strings, access part of a string, determine the number of characters in a string, generate a character string corresponding to a given number or vice versa, search for a substring within a larger string and so on. See chapter 10.2.

5.6 String Arithmetic

String arithmetic functions process numeric strings as arithmetic operands. This is a way to perform calculations with greater precision. Numeric string variable names must be suffixed with a dollar sign ($) character. Numeric string constants must be bounded by quotation marks (") or apostrophes (').

The maximum size of a string arithmetic operand is 125 characters.

5.7 String Input

Just like other variables, the string variables can be assigned data by the instructions **READ, DATA** and **INPUT.**

Example

```
10 INPUT "Address, Name"A4$,A5$
```

is the same as

```
10 PRINT "Address, Name";
20 INPUT A4$,A5$
```

INPUT LINE is very useful for string input. It accepts one line from the keyboard.

Example

```
45 INPUT LINE D$
```

Example

```
210 READ A,B,C$
290 DATA 17,14,61
```

This gives the following assignments:

A=17
B=14
C$="61"

The **INPUT** statement is used to input character strings exactly as though accepting numeric values.

5.8 String Output

Only the characters within quotes are printed when character string constants are included in **PRINT** statements. The delimiters are not printed.

Example

```
10 PRINT "Of course!"
RUN
Of course!
```

Strings can also be stored in files on an output device.

5.9 Relational Operators

The relational operators, when applied to string operands, indicate alphabetic sequence.

Example

```
15 IF A$(I%)<A$(I%+1%) THEN GOTO 115
```

When line 15 is executed, the following will occur: A$(I%) and A$(I%+1%) are compared; if A$(I%) occurs earlier in alphabetical order than A$(I%+1%), execution will continue at line 115.

When two strings of unequal length are compared, the shorter string (length n) will be compared to the first n characters of the longer string. If they are not equal, that inequality serves as the result of the original comparison. If the first n characters of the string are the same, the longer string is greater than the shorter one.

Relational Operators Used with String Variables

Operator	Example	Meaning
=	A$=B$	The strings A$ and B$ are equivalent.
<	A$<B$	The string A$ occurs before B$ in collating sequence.
<=	A$<=B$	The string A$ is equivalent to or occurs before B$ in collating sequence.
>	A$>B$	The string A$ occurs after B$ in collating sequence.
>=	A$>=B$	The string A$ is equivalent to or occurs after B$ in collating sequence.
<>	A$<>B$	The strings A$ and B$ are not equivalent.

6 Working with BASIC II

6.1 How to Write Program Lines

6.1.1 Free Format in Statements

BASIC is a "free format" language. The computer ignores extra blank spaces in a statement. For example, these four statements are equivalent:

```
30 PRINT S
30 PRINT  S
30PRINTS
30P RINT S
```

The computer will always list the programs in its usual way independent of how the statements were written.

NOTE
The spaces are significant in the following cases:

- **EXTEND** mode
- **DATA** statements

6.1.2 Procedure

A line can either be executed immediately (direct mode) or stored in the user program area for later execution and eventually saved on an external device (disk or cassette).

The RETURN key must be pressed after each statement.

Example

```
10 INPUT A,B,C,D,E          (press RETURN)
20 LET S=(A+B+C+D+E)/5      (press RETURN)
30 PRINT S                  (press RETURN)
40 END                      (press RETURN)
```

The RETURN key indicates that the statement is complete. If the statement contains an error, an error message is written on the screen.

6.1.3 Corrections

The ← key acts as a backspace key, deleting the immediately preceding character.

```
Typing this:       20 LR←ET S=10
is equivalent to:  20 LET S=10
```

```
This line:         30 LET←←←PRINT S
is equivalent to:  30 PRINT S
```

When a terminated line gives an error message, it can be edited using the arrow keys (→ and ←).

Example

```
10 LE S=10
```

This will result in an error message. Press → to display the characters and make the necessary changes to the program line.

The **ED** command provides an ability to change characters after the line is completed.

6.1.4 Deleting a Line

To delete the statement being typed, press the CTRL/X keys or the CE key. The entire line being typed will then be deleted.

NOTE
To delete a previously typed statement, type the statement number and press the RETURN key. The line with that number will then be deleted.

Example

```
5 LET S=0
10 INPUT A,B,C,D,E
20 LET S=(A+B+C+D+E)/5
```

To delete line 5 you should type:

```
5
```

(press RETURN)

Use the **LIST** command to check.

6.1.5 How to Change a Program Line

One way of changing a program line is to retype it as it should be. The new line will replace the old one when you press the RETURN key.

To change line 5 in the above example you can type:

```
5 LET S=5
```

(press RETURN)

The old line 5 is replaced by the new one.

If only a few characters are to be changed use the **ED** command.

6.2 How to Edit a Program

Lines may be deleted, inserted or changed according to the procedures described in chapter 6.1. The **MERGE** command lets you combine the program with a set of statements loaded from a file. The **ERASE** command deletes a block of lines. The **ED** command facilitates corrections of an existing line on a character basis.

When editing a program you may want to tidy up the line numbering. This is done by means of the **RENUMBER** command.

6.3 Executing a Program

The **RUN** command will start the execution of a program. When the command is entered and terminated by the RETURN key, BASIC starts to execute the program in the user program area at the lowest numbered line. Execution will continue until either one of these conditions is encountered:

STOP
END
Error

When the program executes a **STOP** or **END** statement, it halts and all the variables still keep their values. The user can examine the variables by simply addressing them by their variable names.

Example: You want to know the values of the variables A, S, and K%. Enter the following command:

```
PRINT A,S,K%          (press RETURN)
```

The computer will then write the current values of the variables when program execution was stopped.

Errors cause an error message to be written on the screen.

A running program can be halted by:

```
CTRL/C      (both keys simultaneously)
```

After that it is possible to single step the program by means of:

```
CTRL/S      (both keys simultaneously)
```

To continue execution, press any key.

To stop the program you have to press:

```
CTRL/C      again.
```

6.4 Guide to the Statements

To insert notes and messages into your program:
Use the instruction **REM** or !.

To assign a numeric value to a variable:
Use the instruction **LET**.

To assign values to a list of variables:
Use the instructions **READ, DATA, ON-RESTORE,** and **RESTORE.**

To transfer data to and from the system:

Use the instructions **INPUT, INPUT LINE, PRINT, GET, PUT, PREPARE, OPEN,** and **CLOSE.** Use **OUT** and **INP** to control input/output via the inports/outports of the ABC 800.

To control the program flow:

1. Unconditional jump to another part of the program
Use the instruction **GOTO.**

2. Conditional branch
Use the instructions **IF-THEN, ON...-,** and **WHILE-WEND.**

3. Program loops
Use the instructions **FOR-NEXT.**

4. Modularized programming with the use of subroutines
Use the instructions **GOSUB-RETURN** and **DEFFN-FNEND.**

To do your own error handling:

Use the instructions **ON ERROR GOTO-RESUME** and the **ERRCODE** function.

To combine your BASIC program with programs written in assembler language:

Use the instructions **CALL** and **POKE** and the functions **PEEK** and **VARPTR.**

To define and manipulate blocks of data: *ABC 802, 806*

Use the instructions **PREPARE, OPEN, CLOSE, PRINT, GET, PUT, INPUT,** and **INPUT LINE.**

Miscellaneous statements: *ABC 802, 806*

COMMON and **DIM** sets the size of variables.
STOP, TRACE, and **NO TRACE** facilitate the debugging of a program.

Chapter 10 contains the many mathematical, logical and other functions available, which extend user programming and provide it with advanced features.

6.5 Declarations

There are the following declarations:

- **FLOAT/INTEGER**

- **SINGLE/DOUBLE**

- **NO EXTEND/EXTEND**

- **OPTION BASE (0/1)**

If any declaration statements are used, they should be placed at the very beginning of the program. **COMMON** and **DIM** statements, if any, should follow directly after the declarations.

7 Direct Mode

BASIC II facilitates computer utilization for the immediate solution of such problems, generally mathematical ones, which do not require iterative program procedures.

To clarify: BASIC II allows the use of the computer as a sophisticated electronic calculator by means of its ability to provide direct statement execution.

When BASIC II is in the command mode, a BASIC statement may be entered without a line number. Such a statement, when terminated by RETURN, will be executed immediately. This is called the direct mode of execution.
Most BASIC statements can be used in direct mode.

Example:

```
A=1.5 : B=3
PRINT "(A+B*A)=";(A+B*A)
```

Statements which are entered with line numbers are considered to be program lines which will be executed later.

Direct execution is very useful as an aid in program development and debugging. Through the use of direct statements, program variables can be altered or read, and the program flow may be monitored and controlled.

Direct statements operating on program variables can be used in the following cases:

- when CTRL/C has been pressed twice

- when an error has occurred

- after a **STOP** or **END** statement

8 Commands

When a command is written and terminated by RETURN it causes BASIC to take immediate action. A BASIC program, by contrast, is first entered into the memory and then executed later, when the **RUN** command is given.

When the BASIC interpretor is ready to receive a command, the text ABC 800 is displayed on the screen. Commands should be typed without any line numbers.

Different commands control the editing and execution of programs and allow file manipulation. Each command is identified by a key word at the beginning of the line.

The following definitions are used in this manual:

- key words - in thick print e.g. **LOAD, SAVE,** and **RUN**
- optional items - within square brackets e.g. [device:]
- alternative items - separated by a slash e.g. "data"/string variable
- additional items - represented by dots e.g. ["data"/string variable, ..., ...]

Generally:

ABC 802, 806

- the devices are addressed as DR0:, DR1:, CAS:, PR:, and CON:.
- the primary default device is disk drive 0 (DR0:) and the secondary one is disk drive 1 (DR1:). If both a disk drive and a cassette recorder are connected, the device CAS: must be given if a command is to act on the cassette recorder.
- a file name should consist of up to eight alphanumeric characters, the first of which is a letter. In addition, an extension (3 characters) may be used to clarify the file name.
- the extension of a file name need not be given explicitly. However, there are some exceptions. All such exceptions are mentioned in the syntax rules. If no extension is given, the command will act first on files with the extension .BAC and then on files with the extension .BAS.
- the RETURN key should be pressed to terminate the entered command.

The following list shows the commands with a short description of each one:

AUTO	Generates line numbers automatically.
BYE	Transfers control to the DOS.
$BAS	Transfers control back to BASIC.

CLEAR	Clears all variables and closes all files.
CON	Continues the execution of a stopped program.
ED	Gives edit facility.
ERASE	Erases blocks of lines.
LIST	Lists the current program.
LOAD	Loads a BASIC program into the computer.
MERGE	Merges program files.
NEW	Clears the program storage.
REN **RENUMBER**	Changes the line numbering.
RUN	(Loads and) Executes a program.
SAVE	Stores the current program on a file.
SCR	Clears the program storage.
UNSAVE	Deletes a file.

Below follows a detailed description of all the commands.

AUTO

Format

AUTO [argument 1[, argument 2]]
where < argument 1 > specifies the first line number to be written and < argument 2 > specifies the line interval. Both arguments are optional. If no arguments are given, the line numbering starts with the first whole 10th number following the already existing lines. The step interval is then set to 10.

Function

The new line number is automatically entered after each carriage return. You do not have to enter the line numbers manually.

Use

This command is continuously available during programming work. Automatic line numbering can be stopped by pressing the RETURN key as the first character of a new line. If a line entered causes an error message, automatic line numbering is stopped and the line can be edited. The line numbering can be started by a new **AUTO** command.

Example

AUTO 10,5

The first line number will be 10 and the line number will be incremented by 5 for each line.

```
AUTO 10,5
10 LET A=1
15 - - -
20 - - -
25 - - -
```

etc

BYE

Format

BYE

Function

Transfers control to the disk operating system (DOS).

Action

Closes any files remaining open and loads the command interpreter CMDINT.SYS.

$BAS

Format

$BAS

Function

Transfers control to BASIC.

Action

Terminates the DOS work and transfers control to the BASIC interpreter.

CLEAR

Format

CLEAR

Function

Clears all variables and closes all open files.

Action

Does not affect the current program in the storage.

CONTINUE

Format

CON (or **CONTINUE**)

Function

Will continue program execution from where it was stopped.

Use

Variables may be changed or displayed by means of direct entry commands before **CON** is used.

Note

CON cannot be used if the program has been edited or an error has occurred.

ED

Format

ED [line number]

Where <line number> is the number of the line to be edited.

Function

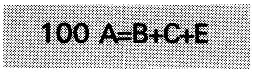
Allows editing of a program line.

Action

When the command has been terminated by the RETURN key, the → is used to display the contents of the line. Each time the → key is pressed, one character is displayed. If you want to alter the contents write the new characters where they should be. Use the ← key to back space.

Example

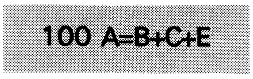
Line 100 has the following contents:



```
100 A=B+C+E
```

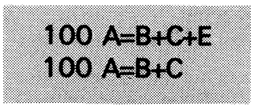
Assume that C is to be replaced by D and that you do not want to rewrite the line. Proceed as follows:

Type **ED** 100. Press RETURN. Line 100 is then displayed:



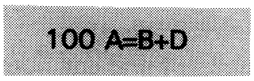
```
100 A=B+C+E
```

Press → to display the following text:



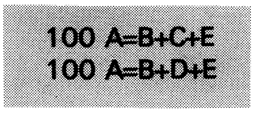
```
100 A=B+C+E
100 A=B+C
```

The second line will thus show your new text. Erase C by pressing ← once. Type D. The bottom line now looks like this:



```
100 A=B+D
```

Press → until the rest of the line is displayed.



```
100 A=B+C+E
100 A=B+D+E
```

Read the line to check if it is correct. Press RETURN to replace the old line with the new one.

Note

When an error occurs during programming, the erroneus line remains in the computer and can be edited by the → and ← keys as shown above without any **ED** command.
If **ED** is typed without a line number, the first line of the program will be displayed.

ERASE

Format

ERASE line number I [- line number II]
ERASE line number -
ERASE - line number

Function

Erases one or more program lines.

Action

All lines between line number I and line number II inclusive are removed.

Examples

ERASE 20-200	Erases lines 20 to 200 inclusive
ERASE -200	Erases all lines up to and including line 200
ERASE 20-	Erases line 20 and all subsequent lines up to and including the last line

LIST

Format

LIST [device:]file name[.extension][,line number[-line number]]
LIST [device:][,line number[-line number]]
LIST line number -
LIST - line number

Where <file name.extension> is the name of a program file. <Device:> can be, for instance, CAS:, PR:, DR0:, or DR1:.

Function

Lists the whole program or part of it.

Action

1. **LIST** [device:] file name [.extension]
Saves the program in the working storage in text format on an external storage. The program is stored under the specified <file name> . Compare with **SAVE**. The default extension is BAS.

2. **LIST**
The entire program is listed on the screen.

3. **LIST** line number
Just the line specified is listed

4. **LIST** line number I - line number II
The lines between line number I and line number II, inclusive, are listed on the screen.

5. **LIST** PR:
The entire program is listed on the printer.

6. **LIST** -line number
All lines up to an including the line specified are listed.

7. **LIST** line number-
All lines from the specified line number to the end of the program are listed.

ABC 802 Note

A long program is listed on the screen until it is filled. The next line will be displayed when you press the space bar. A listing can be stopped by CTRL/C, RETURN or any BASIC command.

Examples

LIST ABC800	Stores the file ABC800 on external storage
LIST	Lists the entire program on the screen
LIST 100	Lists line 100
LIST 100-500	Lists the lines 100-500
LIST PR:	Lists the entire program on the printer
LIST PR:, 100-200	Lists the lines 100-200 on the printer

LOAD

Format

LOAD [device:]file name[.extension]
Where <file name.extension> is the name of the program to be loaded. <Device> can be CAS:, DR0: or DR1:.

Function

Loads a BASIC program into the working storage of the computer from an external storage.

Examples

```
LOAD ABC200
LOAD DR1: PROG.BAS
```

Note

If no extension is given, the computer will first search for .BAC and then .BAS. The entire file is read, until EOF (end of file) and not only to the **END**. ***ABC802***

MERGE

Format

MERGE [device:]file name[.extension]
Where <file name> is the name of a file on an external storage. The file should be stored in text format (by **LIST**).

Function

Merges program files so that the program lines will be in line number sequence. The program from the external storage overlays the current program.

Action

The numbered BASIC lines from the file specified are inserted in line number sequence in the current program. Each new line is checked for errors. If a new line has the same number as an old one, the old line is replaced by the new one.

The external file is read until EOF (end of file).

Example

Existing program:

```
5 Y=1
10 PRINT
20 FOR L=1 TO 10
30 PRINT L;TAB(Y);"I";
40 READ Y
50 FOR I=1 TO Y
60 PRINT " * ";
70 NEXT I
80 PRINT
90 PRINT TAB(Y);"I"
100 NEXT L
```

The following program file is stored on an external storage (disk or cassette) under the name TABLE:

```
200 DATA 5,4,2,3,1
300 DATA 10,15,28,15,6
999 END
```

The following command will link the two files together:

```
MERGE TABLE
```

The command **MERGE** adds the lines 200, 300 and 999 to the existing program.

NEW

Format

NEW

Function

Clears the working storage.

Action

Clears the working storage and all variables and resets the pointers. The command erases all traces of the existing program from the storage and starts over again.

All open files are closed.

Use this command before typing a new program.

Note

The command **SCR** (scratch) can be used, as well.It works just like **NEW.**

RENUMBER
REN

Format

REN[UMBER] [line number[,interval[,from line -to line]]]
Where <line number> is the number to be given to the first line. The default is 10.

<Interval> is the increment number. The default is 10.

<From line> - <to line> specify which lines are to be renumbered. The default is that all the lines are renumbered.

If you want to specify <interval>, <line number> must be specified.

If you want to specify <from line> - <to line>, you have to specify both <line number> and <interval>.

Function

Changes the line numbering of the current program.

Action

All the line numbers of the program are changed as specified in the **RENUMBER** command.

Any references to line numbers in **GOSUB, GOTO, IF, ON,** and **RESUME** statements are changed to the new line numbers, so that the entry points still represent the same statements as before.

If a statement in the program refers to a line number, which does not exist, an error message is printed and no renumbering is done.

Examples

```
REN
REN 10
RENUMBER 10,5
REN 10,5,10-50
REN 10,5-100
REN 10,5,100-
```

RUN

Format

RUN [device:][file name[.extension]]
Where <file name> is the name of the program file to be loaded and executed. If no extension is given, the computer will search for an extension of first .BAC and then .BAS.

Function

Loads and executes a BASIC program or executes the current program.

Action

1. **RUN**
All variables and arrays in the program area are erased and all buffers are cleared. The execution of the current program is started at the lowest numbered line.

2. **RUN** file name[.extension]
The program <file name[.extension]> is loaded through the action of a **LOAD** command. The program is then executed, starting at the lowest numbered line.

Examples

Example 1

```
10 READ A,B
20 LET A=A+B
30 PRINT A
40 DATA 2,3
50 END
RUN
5
```

Example 2
If the same program had been stored on an external storage under the name AADDB, the screen would look like this:

```
RUN AADDB
5
```

SAVE

Format

SAVE [device:]file name[.extension]
Where the <file name> is a string literal specifying the name of a new file. The default file extension will be .BAC.

Function

Creates a program file on an external storage and stores the current program on that file.

Action

The command causes the program, which is currently in the working storage, to be saved under the given file name with the extension .BAC, if no other extension is specified.
The program is saved in internal code to ensure fast loading.

Example

```
10 - - -
900 - - -
999 END
SAVE TEST
```

ABC 802 Note

If the file exists already on the disk, the old file will be destroyed and replaced by the new program, unless the file or the disk is write protected.

SCR

Format

SCR

Function

Clears the storage.

Action

Clears the working storage and all variables and resets the pointers. The command erases all traces of the existing program from the storage and starts over again.

All open files are closed.

Note

The command **NEW** can be used, as well.It works just like **SCR.**

UNSAVE

Format

UNSAVE [device:]file name[.extension]

Function

Erases a file from a disk.

Example

When you have completed all work with the file XYZ, the file can be erased by the following command:

```
UNSAVE XYZ
```

ABC 802 Note

When the extension is omitted, the computer will look for .BAC first and then .BAS.
The command **UNSAVE** cannot be used on an erase protected file.

9 Instructions

This chapter describes the program statements of BASIC II. A program statement is an instruction, which tells the BASIC interpreter to perform a certain operation. Most instructions can also be used as commands.

Below is a list of the instructions with a short description of each one:

CHAIN	Loads a program file and executes it
CLOSE	Closes files
COMMON	Transfers variables to the next **CHAIN**ed program
DATA	Data statement; **READ** fetches the value
DEF FN	Defines a user function
DIGITS	Denotes the maximum number of digits to be printed
DIM	Denotes the size of a vector/matrix or string
DOUBLE	Double precision (16 digits)
END	The logically last instruction of a BASIC program
EXTEND	Allows the use of long variable names
FLOAT	All variables are represented as floating point ones
FNEND	Terminates multiple line user functions
FOR	Program loop (with **NEXT**)
GET	Reads a character
GET COUNT	Reads the specified number of characters
GOSUB	Calls a subroutine
GOTO	Jump to a specified line number
IF-THEN-ELSE	Controls conditional execution
INPUT	Reads data
INPUT LINE	Reads data
INTEGER	All variables are represented as integers
KILL	Erases a file
LET	Assigns a value to a variable
NAME	Renames a file

NEXT	Increments the variable in a **FOR** loop
NO EXTEND	Disables **EXTEND** mode
NO TRACE	Disables **TRACE** mode
ON ERROR GOTO	Error handling
ON-GOSUB	Conditional jump to subroutines
ON-GOTO	Conditional jump to one of several lines
ON-RESTORE	Conditional **RESTORE** statement for data pointer
ON-RESUME	Conditional jump from error handler
OPEN-AS FILE	Opens a file and assigns a file number
OPTION BASE	Sets the minimum value for vector subscripts
POSIT	Positions the file pointer
PREPARE-AS FILE	Creates and opens a new file and assigns a file number
PRINT	
PRINT USING	Prints data with the specified format
PUT	Writes a record
RANDOMIZE	Provides a new initial value for the random number generator
READ	Assigns a value from a **DATA** statement to a variable
REM (!)	Inserts a comment (remark)
RESTORE	Sets the data pointer
RESUME	Returns from error handler
RETURN	Returns from subroutine to calling program
SINGLE	Changes the precision to seven digits
STOP	Stops the program execution
TRACE	Allows tracing
WEND	Terminates a **WHILE** function
WHILE	Defines the branching condition of a **WHILE-WEND** function

ABC 802, 806

Here follows a comprehensive description of the instructions.

CHAIN

Format

CHAIN "file name.extension"/string variable
Where <file name.extension> is the name of the program to be loaded. If the file name refers to a disk file and no extension is given, the computer will search for .BAC and secondly .BAS.

Function

Loads and executes a program.

Use

If the program is too large to be loaded into the computer storage and run in one operation the program can be segmented into several programs. The **CHAIN** instruction is used as a logical termination of one program to call the next one. Each program is called by its name. The program in the computer is erased and the new one is loaded. The lowest numbered program line is executed first as though a **RUN** command had been used. The **CHAIN** instruction is the last instruction to be executed. The last program in a chain does not need any **CHAIN** statement, but control is often transferred by **CHAIN** back to a program that allows the user to select the program to be run.

Note

Variables can be passed on to a **CHAIN**ed program by means of the **COMMON** instruction. ***ABC 802***

Example

```
550 CHAIN "PROGRAM4.BAC"
```

CLOSE

Format

CLOSE [file number, ...]
<File number> specifies the file to be closed.

Function

Terminates I/O instructions and closes the file(s).

Use

The instruction **CLOSE** is used to close one or more files. If no file number is given, all files will be closed.

Note

The instruction **END** closes all open files.
Ordinary output with the **PRINT** instruction will cause the last buffer to be output when the file is closed.

COMMON

Format

COMMON variable[(n,...)] [,variable, ...]
COMMON string variable[(n,...)]=length [,stringvariable=length, ...]

<n>=the vector index and <length>=the maximum string length

Function

A declaration of the variables whose values are to be transferred to another program when the programs are **CHAIN**ed.

Use

The declarations must look alike in all programs. The programs must be alike regarding precision and integers or floating point values.

Note

The length of common string variables must be declared. The **COMMON** statements must follow immediately after the declaration statements.

Example

```
10 COMMON A%,B(7),C$(15)=25
```

DATA

Format

DATA value [value, ...]
Where all the **DATA** statements, placed anywhere in the program, form a list of data. The **DATA** statement should be the only statement on the line, i. e. the line must not contain, for instance, **REM** since this instruction is interpreted /0/as data.

Function

Assigns values to variables (used with **READ**).

Use

DATA is used only in conjunction with **READ** and vice versa. See **READ.**

Example

```
100 DATA ....a.text...,"..a.text."
110 DATA ' "...a.text..." '
120 FOR I=1 TO 3
130 READ A$
140 PRINT "!";A$;"!"
150 NEXT I
```

yields the following display

```
!....a.text...!
!..a.text.!
!"...a.text.."!
```

DEF FN

Format

Single line function:
DEF FN identifier[(argument)]=function

Multiple line function:
DEF FNidentifier[%/$] [(argument)] [**LOCAL** variable
[, variable, ...]]

Function

Defines single line and multiple line functions.

Use

A multiple line **DEF** function differs from the single line functions due to the absence of an equal sign following the function designation on the first line. Any number of arguments of any type or any mixture of types may be used. Within the multiple line function definition there must be a statement of the form:

RETURN expression

FNEND

When the **RETURN** statement is encountered, the expression is evaluated and used as the value of the function, and exit is performed from the definition. The definition may contain more than one **RETURN** statement, as can be seen from the example below.

Note

ON ERROR GOTO line number, **GOSUB** line number, and **RESUME** line number may only call lines within the function. Only **RESTORE** to data statements is allowed to point on lines outside the function.

Examples

Single line function:

```
10 DEF FNA(X,Y)=X+X*Y
```

Multiple line functions:
The function below determines the larger of two numbers and returns that number. Such a use of the **IF - THEN** instruction is frequently found in multiple line functions:

```
 5 EXTEND
10 DEF FNM(X,Y)
20 IF Y<=X THEN RETURN X
30 RETURN Y
40 FNEND
```

The next example shows a recursive function that computes the N-factorial. (However, there are more efficient, non-recursive routines for the computation of N-factorial.):

```
5 EXTEND
10 DEF FNFak(M%)
20 IF M%=1% THEN RETURN 1% ELSE
RETURN M%*FNFak(M%-1%)
30 FNEND
35 INPUT "Value for factorial: ";X
40 PRINT X;"-factorial equals ";FNFak(X)
50 END
RUN
Value for factorial: 4
4-factorial equals 24
```

Any variable referred to in the body of a function definition which is not an argument or a local variable of that multiple line **DEF** function keeps its value in the calling program. Multiple line **DEF** functions can be nested; one multiple line function definition can refer to itself or another multiple line function definition. The same rules apply here as for the nesting of program loops. There must be no transfer from within the definition to outside its boundaries or from outside the definition into it. The line numbers used by the definition must not be referred to elsewhere in the program. If **ON ERROR GOTO** is used inside the function it will be disabled as the function exits to the calling program.

If temporary variables are needed within a function definition they should be declared local to the function in order to protect the global variables from being disturbed. This eliminates the need for variable names that are free for usage.

The **LOCAL** modifier makes possible the local variable name option. Vectors cannot be declared **LOCAL** and string variables must have explicit length.

```
10 DEF FNA(X) LOCAL A,A$=10
20 A=33: A$="Local"
30 PRINT A$
40 PRINT A
50 RETURN 5*X
60 FNEND
100 A=22: A$="Global"
110 PRINT A$
120 PRINT A
130 PRINT FNA(8)

 RUN
 Global
 22
 Local
 33
 40
```

The next example shows a string function:

```
100 PRINT FNV1$("AaBaCcDdEeFf",5%,10%)
110 END
120 DEF FNV1$(A$,B%,C%)
130 IF B%=C% THEN RETURN LEFT$(A$,B%) ELSE
RETURN RIGHT$(A$,C%-B%)
140 FNEND
RUN
CcDdEeFf
```

FNEND

Format **FNEND**

Function Terminates a multiple line function.

Use See the instruction **DEF FN**.

Note This statement must not be executed. The function shall exit by a **RETURN** statement before **FNEND** is encountered.

DIGITS

Format **DIGITS** number of digits

Function Gives the number of digits to be printed.

Action A number printed by means of **PRINT** is rounded off to the nearest value for the last digit. Values too great to be displayed in this form are printed in exponent form with the specified number of digits.

Note The **DIGITS** instruction does not affect the accuracy of calculation.

Default: 6/16 digits depending on precision.

DIM

Format

DIM variable(n)[, variable(n,...), ...]
DIM string variable[(n,...)] [=length]
Where the value following the "=" denotes the string length and <n,...> is the greatest index value. The minimum index equals the lower limit unless stated otherwise. The lower limit is either 0 or 1 depending on the most recent **OPTION BASE** statement. The default value is 0.

Function

Gives the maximum number of elements and allocates space for strings and vectors.

Use

Any number of indices is allowed both for scalar and vector variables. All values used in **DIM** statements are rounded off to integers. If a subscripted value is used without a **DIM** statement it is assumed to be dimensioned as 10 for each index. All variables have a zero value until assigned a value. If a string variable has not been dimensioned, its max length is automatically set to the current length the first time that the string is assigned a non-null value (<>""). If less than 80 characters are used a standard length of 80 characters is assigned.

Examples

```
DIM A$(N)
```

A string vector with the strings A$(lower limit) -A$(N). Each string has a length of 80.

```
DIM A$(N)=14
```

As above but the maximum length of each string is 14 characters.

```
10 DIM X(5),Z(4,3),A(7,2,3,5)
12 DIM A4(100)
14 DIM A$(20),B$(10,20)
16 DIM C$(40)=4
18 DIM D$(10,10)=8
20 DIM Q$=253%
```

Advanced Programming:

The lower limit (0 or 1) indicated above can be overridden individually for each index. This is done by replacing the single maximum index for each dimension by two values separated by a colon.

Example

```
DIM A(-2:2)
```

The example will yield a vector with five elements A(-2), A(-1), A(0), A(1), A(2) which are totally independent of the current lower limit.
A dimensioned variable can be redimensioned only if the new **DIM** statement defines a smaller dimension.

DOUBLE

Format

DOUBLE

Function

All variables and expressions with floating point numbers are changed to double precision (16 digits).

Use

The **DOUBLE** declaration should be placed before the variables are used in the program and cannot be changed when the program has been started by **RUN.** This change can be made when a program line has been edited or the **CLEAR** command has been used. The default precision is **SINGLE.**

Note

SINGLE and **DOUBLE** cannot be mixed in the same program.

END

Format

END

Function

Terminates the program.

Use

The logically last instruction of a program. **END** closes all files.

Note

The variables keep their values after **END.**
END must be the first statement on the line.

EXTEND

Format

EXTEND

Function

Permits long variable names.

Note

In the **EXTEND** mode, BASIC requires spaces to delimit names and functions, unless the adjoining character is a line number or an arithmetic operator (-+*/). If key words are written without spaces they may be mistaken for long variable names. The variable names can have unlimited length and all the characters are significant.

Example

```
10 EXTEND
20 LET Subtotal=Units*Unitprice
```

FLOAT

Format

FLOAT

Function

All variables are interpreted as floating point. Integers must have a % suffix.

Use

See the **INTEGER** instruction.

Note

FLOAT is the default value. **FLOAT** and **INTEGER** cannot be mixed in the same program.

Example

```
10 A=125.5 !Floating point number
20 A%=12% !Integer
```

FOR

Format

FOR variable=expression **TO** expression [**STEP** interval]

Where the variable in the **FOR** - **TO** statement is set initially to the value of the first expression. The statements following the **FOR** statement are then executed. When the **NEXT** instruction is encountered the variable is incremented by the value indicated as the **STEP** interval. See **NEXT.**

If the variable value exceeds the value of the **TO** expression, the next instruction executed will be the one following the **NEXT** statement.

The expressions within the **FOR** loop are evaluated once, when the loop is initially entered. The test for completion of the loop is made prior to each execution of the loop.

Function

The **FOR** and **NEXT** instructions are used together to create a program loop. A loop means that one or more instructions are executed a number of times.

Use

A program loop consists of four parts:

1. Initialization to set up the condition which must exist for the first execution of the loop.

2. The body of the loop; i.e. the instructions for the operations to be repeated.

3. The modification which alters a value and makes each execution of the loop different from the others.

4. The termination condition; an exit test which, when satisfied, completes the loop. Execution continues with the program statement following the loop.

If the **STEP** interval is omitted from the **FOR** statement, +1 is the assumed value. Since +1 is a common **STEP** interval, that part of the statement is frequently omitted.

The variable can be modified within the loop. When control falls through the loop, the variable will have its new value, i.e. the last value used plus the interval.

FOR loops can be nested but not overlapped. Nesting is a programming technique in which one or more loops are completely within another loop.

It is possible to leave a **FOR** - **NEXT** loop without the variable reaching the termination value. A conditional or unconditional branch can be used to exit from a loop. When reentering a loop which was left earlier without being completed, be careful to ensure that the correct termination and interval values are assigned.

Example 1

The following is a demonstration of a **FOR - NEXT** loop. The loop is executed 20 times. Before the exit from the loop A=20 is displayed. The **FOR** statement contains no **STEP** interval so the interval is assumed to be +1.

```
10 FOR A%=1% TO 20%
20 PRINT "A=";A%
30 NEXT A%
40 PRINT "A=";A%
```

The loop consists of the lines 10, 20, and 30. When A% has the value 20 and line 30 is executed, A% is incremented by 1 and line 10 is executed. Since A% is greater than the upper limit, line 10 will cause control to be passed to line 40 which causes A=21 to be displayed.

Example 2

Acceptable nesting

```
50 FOR A=1 TO 10
60 FOR B=2 TO 11
70 NEXT B
80 FOR C=1 TO 10
90 NEXT C
100 NEXT A
```

Unacceptable nesting

```
150 FOR A=1 TO 10
160 FOR B=2 TO 11
170 NEXT A
180 NEXT B
```

Note

The use of an integer variable in a **FOR** loop is recommended, since it results in a faster loop execution.

NEXT

Format

NEXT variable
Where <variable> is the variable specified in the **FOR** statement. The **FOR** and **NEXT** statements are the delimiters of the loop. When the **NEXT** statement is executed, the variable is incremented by the interval and the program determines if the variable value exceeds the maximum value given in the **FOR** statement. When the value of the variable is greater than the upper limit, control falls through the loop to the statement following the **NEXT** statement.

Function

NEXT terminates a program loop, which begins with a **FOR** statement. When **NEXT** is encountered the variable will be incremented by the interval.

Use

See **FOR.**

GET

Format

GET string variable

Function

Reads one character from the keyboard into a string variable.

Note

If the keyboard buffer is empty, the BASIC interpreter will wait until a key is pressed. Any character can be read.

GET £

Format

GET £ file number, string variable [**COUNT** number of characters]

<File number> is the file number defined by the **OPEN** instruction.
<String variable> is the string which receives the character(s).
COUNT <number of characters> denotes the number of characters to be read from the file.

Function

Reads one or more characters from the specified file into the specified string variable.

Use

See chapter 3.5.

GOSUB

Format

GOSUB line number
<Line number> is the first line number of the called subroutine. The program execution will go on from that line number.

Function

Unconditional jump to a subroutine.

Use

A subroutine is a sequence of program instructions, which perform a task that is repeated several times in a program. Subroutines and functions enable such a sequence of instructions to be called from several program lines.

A subroutine is a part of the program which can be called by means of a **GOSUB** instruction. When the subroutine has completed its task, a **RETURN** instruction is used to exit the subroutine and continue the program execution with the statement following the calling **GOSUB** statement.

Example

```
50 GOSUB 1300
I
I
1290 REM This is a subroutine
1300 LET K=1
I
I
1360 RETURN
I
9999 END
```

Note

The only instructions that may be used to exit a subroutine are **GOSUB** or **RETURN**.

GOTO

Format

GOTO line number

Where the <line number> is usually not the next sequential line in the program.

Function

Unconditional jump to the given line number.

Use

The **GOTO** instruction is used to accomplish an unconditional jump to another line than the next sequential line in the program. The **GOTO** instruction can be used to jump backward as well as forward in a program.

When written as part of a multiple statement line, **GOTO** should always be the last statement on the line. Any statement following the **GOTO** statement on the same line will never be executed.

Example

```
10 X=20
20 PRINT X
30 X=X+1
40 GOTO 20
50 END
```

Note

GOTO can be used in direct mode instead of **CON** if the execution is to be resumed at a certain line.

IF - THEN - ELSE

Format

IF condition **THEN** statement[s]/line number [**ELSE** statement[s]/line number]
Where the condition specified is tested. If the condition is met (the expression is logically true), control is transferred to the line number given after **THEN** or the statement given after **THEN** is executed. If the condition is not met (the expression is logically false), the program execution continues at the program line following the **IF** statement.

Function

Conditional control of the order of execution of the program lines.

Use

THEN may be followed by either a line number or one or more BASIC statements. If BASIC statements are given and the condition is met, these statements will be executed before the program continues with the line following the **IF** statement. The condition applies to all statements that follow on the same line as the **IF** statement.

ELSE is followed either by a line number which is used as a jump address or one or more statements which are executed before the line following the **IF** statement. If the condition is met, the instructions between **THEN** and **ELSE** will be carried out.
When relational expressions are evaluated, the arithmetic operations take precedence in their usual order. The relational operators have equal weight and are evaluated after the arthmetic operators but before the logical operators.

The Relational Operators

=	Equal
<>	Not Equal
<	Less Than
>	Greater Than
<=	Less Than or Equal
>=	Greater Than or Equal

A relational expression has a value of -1 if it is evaluated to be true and zero if it is evaluated to be false.

Example: 5+6*5>15*2 is true

Example

```
170 IF A<B+3 THEN 160
180 IF A=B+3 THEN PRINT "A has the value ";A
190 IF A>=B THEN T1=B
200 IF A=B THEN PRINT "Equal ":A=1/B
210 IF A>B THEN PRINT "Greater" ELSE PRINT "Not greater"
```

The condition in line 200 applies both to the **PRINT** statement and the assignment statement.

INPUT

Format

INPUT [£file number/"prompt text"] variable [,variable, ...]

Where <file number> is the number assigned by the **OPEN** statement.

<Variable> may be the name of an arithmetic variable, an element of a numeric vector, a string variable or an element of a string vector.

If no £<file number> is given, the system will assume that the data input comes from the keyboard. The data is read from the file or device assigned to the specified file number.

<"Prompt text"> is a character string delimited by quotes. <"Prompt text"> can only be used when the variable is to be entered from the keyboard.

Function

Fetches data for the current program.

Use

During program execution, the user can type data when the program asks for it. **INPUT** causes the computer to wait for an answer. If no prompt text is given, a question mark is displayed on the screen.

It is often convenient to display a prompt text to remind the user of the kind of input data required. See Example 1 below. No question mark is written after the prompt text.

Examples

Example 1

```
10 INPUT "Your name : ?"A$
20 INPUT "Your address : ?"B$
```

is equivalent to

```
10 PRINT "Your name : ";
20 INPUT A$
30 PRINT "Your address : ";
40 INPUT B$
```

Example 2

```
50 INPUT £3,C$
```

Data will be read from file 3 and placed in the string C$.

INPUT LINE

Format

INPUT LINE [£file number,]string variable

<File number> is the number specified in the **OPEN** instruction and stands for an external device or file as a logical unit.

Function

Accepts a line of characters.

Use

The program accepts a line of characters from the specified file. All characters belonging to the line are read; spaces, punctuation characters, and quotes. The line termination characters carriage return (CR) and line feed (LF) are read, as well.

No text can be output by the **INPUT LINE** statement; this facility is only available in the **INPUT** statement. Use the **PRINT** instruction to print out the prompt text.

Examples

Example 1

```
10 PRINT "Your address : ";
20 INPUT LINE A$
```

Example 2

```
10 INPUT LINE A$
20 A$=LEFT$(A$,LEN(A$)-2)
```

In Example 2 CR and LF are removed from the string A$.

INTEGER

Format

INTEGER

Function

At data input and program listing, all variables are supposed to be integer variables, unless otherwise declared.

Use

When a program is being typed and the **INTEGER** instruction has been given, the programmer need not type the integer suffix %. On the other hand, all floating point variables should be marked by a decimal point suffix (.). The strings should have the usual $ suffix.

A program which is stored in text format and contains floating point variables can be run as an **INTEGER** program if the command **INTEGER** is given prior to loading the program. Save the program and you have converted it into an integer program.

Note

The default format is **FLOAT.**
INTEGER and **FLOAT** cannot be mixed in the same program.

program.

Example

```
100 REM The listing formats of FLOAT/INTEGER
110 INTEGER
120 A.=12.345
130 B.=123
140 C=B.
150 D1=A.
160 PRINT A.,B.,C,D1
170 END
RUN
12.345 123 123 12
```

KILL

Format

KILL "[device:]file name[.extension]"

Where the file with the name <file name .extension> is not delete-protected. The user cannot erase a delete-protected file.

Function

The file in question is erased from the external storage.

Example

When the file XYZ.TXT on the disk is no longer needed, the file can be erased from the disk by means of the following statement:

ABC 802

```
460 KILL "XYZ.TXT"
```

LET

Format

[**LET**] variable=expression

The word **LET** is optional.

Function

Assigns a value to a variable.

Example

```
10 LET A=5.02
20 LET B9=5*(X/2)
30 D=(3*A)/2-B
```

NAME

Format

NAME "[device:]file name1.extension" **AS**
"file name 2.extension"

Function

Changes the name of a file.

Use

The file with the name <file name1.extension> will be given the new name <file name2.extension>.

No file name extension is assumed. The file name extension must be specified in both cases, if the file is stored under a name which includes an extension and if an extension should be included in the new name.

Examples

Example 1

```
100 NAME "DR0:OLD.BAC" AS "NEW.BAC"
```

Example 2
The following statement:

```
200 NAME "DR0:ABC.BAC" AS "XYZ.BAC"
```

changes the name of the file ABC.BAC on the disk in DR0:. The instruction **NAME - AS** cannot transfer a file from one device to another.

Example 3

```
120 NAME "NEW" AS "NEW1"
```

NO TRACE

Format

NO TRACE

Function

Terminates the printout of line numbers, which was started by the instruction **TRACE**.

Example

```
10 PRINT "Start "
20 K=-1
30 TRACE
40 IF K>1 THEN 80
50 K=K+1
60 PRINT "Number ";K
70 GOTO 40
80 A=K
90 NO TRACE
100 PRINT "Stop"
RUN
Start
40 50 60 Number 0
70 40 50 60 Number 1
70 40 50 60 Number 2
70 40 80 90 Stop
```

The **TRACE** function is disabled before line 40 and after line 90.

NO EXTEND

Format **NO EXTEND**

Function Disables **EXTEND** mode.

Use In **NO EXTEND** mode variable names may be composed of one letter and one optional digit. The default mode is **NO EXTEND.**

ON ERROR GOTO

Format **ON ERROR GOTO** [line number]

Function Branches to the indicated line number on an error.

Use See chapter 2.6

Note If < line number > is omitted, no jump will be executed at an error. **RESUME** is used to return from the error handler.

ON - GOSUB

Format **ON** expression **GOSUB** line number[,line number,...]

Where control is transferred to a subroutine beginning at one of the line numbers depending on the integer value of the expression. Execution is resumed at the line following the statement. If the value of the expression addresses a line number outside the range of the list, an error message will be displayed.

Function Conditional jump to one of several subroutines or to one of several entry points in a subroutine.

Example

```
10 FOR X=1 TO 5
20 PRINT X
40 ON X GOSUB 1300,200,1300,400,1300
50 PRINT A$
60 NEXT X
70 END
200 LET A$="Sub200"
210 RETURN
400 LET A$="Sub400"
410 RETURN
1300 LET A$="Sub1300"
1310 RETURN
```

Control is transferred to line	for X=
1300	1
200	2
1300	3
400	4
1300	5

ON - GOTO

Format

ON expression **GOTO** line number[,line number,...]

Where the integer value of the <expression> is used as a pointer in the list of line numbers.

Function

Jumps to one of several line numbers, depending on the value of the expression.

Example

```
100 ON A/B GOTO 1000,1500,1700
```

Control is transferred to:
1. line number 1000 if 0.5<=A/B<1.5
2. line number 1500 if 1.5<=A/B<2.5
3. line number 1700 if 2.5<=A/B<3.5
4. error if A/B<0.5
5. error if A/B>=3.5

ON - RESTORE

Format

ON expression **RESTORE** line number[,line number,...]

Where the integer value of the <expression> sets the **DATA** pointer to the specified line number.

Function

Sets the **DATA** pointer by the same selection routine as **ON - GOTO.**

Use

The **ON - RESTORE** statement can thus be used to set the **DATA** pointer to a specific position in the data buffer.

Example

```
10 FOR X=1 TO 3
20 READ A,B,C
30 ON X RESTORE 60,70,80
40 PRINT A,B,C
50 NEXT X
60 DATA 1,2,3
70 DATA 4,5,6
80 DATA 7,8,9
90 END
RUN
1 2 3
1 2 3
4 5 6
```

ON - RESUME

Format

ON expression **RESUME** line number[,line number,...]

Where the integer value of the <expression> is used as a pointer in the list of line numbers.

Function

Jumps to one of several line numbers, depending on the value of the expression. Error handling is resumed.

Use

The **ON - RESUME** instruction is used to accomplish a conditional return from an error handling routine.

Example

```
10 ON ERROR GOTO 100
I
I
100 REM Error handler
I
I
150 ON A RESUME 1000,2000
```

Note

ON - RESUME is used with **ON ERROR GOTO.** See chapter 2.6.

OPEN

Format

OPEN "[device:][file name[.extension]]" **AS FILE** file number *ABC 802, 806*

Where <device> may be for instance

DR0:	Disk drive 0
DR1:	Disk drive 1
PR:	Printer
CAS:	Cassette recorder
V24:	Serial channel
CON:	Keyboard and screen

The expression following **AS FILE** should be an integer value between 0 and 255.

Function

Opens a file with a file number internal to the BASIC program.

The optional <file name.extension> is not used when opening the printer.

Use

OPEN is used to open files which already exist.

Data files or devices have both external names, by which they are identified within the system, and file numbers, which refer to the files within the program. The **OPEN** statement associates the external file name with the internal file number.
Writing and reading from a file is done by means of instructions such as **INPUT, INPUT LINE, PRINT, GET,** and **PUT.**

Note

When data is to be read from an existing file, the file should be opened by the **OPEN** instruction. Up to seven files may be open at the same time. *ABC 802*

Examples

Example 1

```
50 OPEN "TEST.TXT" AS FILE 1
```

Example 2

```
 10 OPEN "DATA.TXT" AS FILE 2
 20 INPUT £2,A
 30 INPUT £2,B
 40 INPUT £2,C7$
```

The values of the variables A, B, and C7$ are read from the file, which was opened as file number 2. The values are read directly after the values last read. If reading is to be done from the beginning of the file, it must be opened again with the **OPEN** instruction.

OPTION BASE

Format

OPTION BASE n

Where n = 0 or 1.

Function

Denotes the lowest vector index value.

Note

The default value is 0. The **OPTION BASE** declaration must be placed before any **DIM** sinstructions or use of vectors.

POSIT

Format

POSIT £file number, position

Function

Positions the file pointer.

Use

POSIT is used to move the file pointer the specified number of positions from the beginning of the file (the first position). The first position = 0. **POSIT** can be used together with all file handling instructions. **POSIT**(file number) yields the current position of the file pointer. See chapter 3.5.

Examples

Example 1

```
10 POSIT £1,5
```

The file pointer is moved to position 5, i.e. it points to the sixth character of file number 1.

Example 2

```
50 A=POSIT(1)
```

A=the position of the file pointer. In Example 1 above, the file pointer is in position 5, i.e. A=5.

PREPARE

Format

PREPARE "[device:][file name.extension]" **AS FILE** file number

Function

Creates and opens a new file with an internal file number within the current program.

Use

PREPARE is used just like **OPEN** but will set up a new file. **OPEN** is used for existing files.

Example

ABC 802

```
10 PREPARE "DATA.TXT" AS FILE 2
20 PRINT £2,A
30 PRINT £2,B
40 PRINT £2,C$
```

The values of the variables A, B, and C$ are written on file 2 (DATA.TXT)

PRINT

Format

PRINT [£file number,] "data"/variable [,"data"/variable, ...]

Where £<file number> corresponds to the file number in the **OPEN** and **PREPARE** instructions. If no file number is given, the data will be displayed on the screen. A semicolon (;) can be used instead of **PRINT**.

If an element in the **PRINT** list is not a simple variable or a constant, the expression is evaluated before the data is printed. The instruction can also contain character strings within quotes.

Function

Prints data on an ASCII format.

Use

The positions on a line are numbered from 0 to 39/79. The line is subdivided into columns, fixed tabulator positions, starting in positions 0, 15, 30, 45, 60, and 75. A comma (,) after a variable or a string in the **PRINT** list means that the next element of the list will be printed in the next column. Two commas together in a **PRINT** statement cause a column to be skipped.

A semicolon (;) following a variable or a string in the list causes the next element in the list to be printed in the next position i.e. immediately after the previous character. If the list is terminated by a semicolon (;) no line feed will follow the **PRINT** statement.

When a line is filled, the printout continues on the next line. The **TAB** and **CUR** functions are used to cause data to be printed in certain positions.

A **PRINT** statement without any argument causes carriage return and line feed to be printed, i.e. one blank line.

$$

A double dollar character causes a dollar character to be printed to the immediate left of the formatted number. The $$ specify two more digit positions, one of which is the dollar character. The exponential format cannot be used with $$. Negative numbers cannot be used unless the minus sign is trailing.

```
PRINT USING "$$££.££ ";456.78
$456.78
```

**$

The combination **$ at the beginning of a format string combines the effects of ** and $$. Leading spaces will be filled with asterisks and a dollar character will be printed before the number. **$ specify three more digit positions, one of which is the dollar character.

```
PRINT USING "**$££.££";2.34

***$ 2.34
```

A comma to the left of the decimal point in a formatting string causes a space to be printed to the left of every third digit to the left of the decimal point. A comma at the end of the format string is printed as part of the string. This comma serves as the delimiter between two numbers. A comma specifies one digit position. The comma has no effect if used with the exponential (↑↑↑↑) format.

```
PRINT USING "£££,.££";1234.5
1 234.50

PRINT USING "£££.££,";1234.5
1234.50,
```

↑↑↑↑

Four up-arrows may be placed after the digitposition characters to specify exponentialformat. The four up-arrows specify theposition of E+xx. Any decimal point position may be specified; the exponent will be adjusted. Unless a leading + or leading or trailing + or - are specified, one digit position at the beginning of the number will be used to print the minus sign.

```
PRINT USING "££.££↑↑↑↑";234.56
2.35E+02

PRINT USING ".££££↑↑↑↑";-888888
-.8889E+06

PRINT USING "+.££↑↑↑↑";123
+.12E+03
```

_

An underscore in the format string causes the next character to be printed out as a literal character.

```
PRINT USING "_!££.££!_";12.34
!12.34!
```

The literal character itself may be an underscore if the format string contains "_ _".

% If the number to be printed is larger than the specified numeric field, a percent character is printed before the number. A percent character is printed also if rounding causes the number to exceed the field.

```
PRINT USING "££.££";111.22
%111.22

PRINT USING ".££";.999
%1.00
```

PUT

Format **PUT** £file number, string variable

Where £<file number> is a file number defined by any one of the **OPEN** and **PREPARE** instructions. <String variable> may be a string variable or a string expression.

Function Writes a string variable on a binary format.

Use See chapter 3.5.

RANDOMIZE

Format **RANDOMIZE**

Function Sets a random starting value for the **RND** function (the random number generator).

Use The **RANDOMIZE** instruction should appear before the first random number generator call **RND** in the program. **RANDOMIZE** makes the random number generator produce different random numbers each time the program is run by initializing **RND** to start at a new value when **RANDOMIZE** is executed.

CAUTION Should only be used once in a program.

READ

Format **READ** variable[, variable, ...]

Function Used together with **DATA** statements as a way of assigning values to variables.

Use The **READ** instruction causes the variables listed to be assigned sequential values from the **DATA** statements. Before the program is run, BASIC creates a data block from all the **DATA** statements in the order of appearance. Each time a **READ** instruction is encountered in the program, the data block will supply the next value.

The **READ** and **DATA** statements are used together.

If it is necessary to use the same data several times in a program, the **RESTORE** or **ON RESTORE** instructions will set the data pointer within the data block. See **RESTORE** and **ON RESTORE.**

Examples

Example 1

```
100 READ A,B,C,D,X1,X2
I
I
150 DATA 3,6,1.8
200 DATA 6.83E-3,-86.4,3.14
```

When the program is run, the variables will be assigned the following values:

```
A=3
B=6
C=1.8
D=6.83E-3
X1=-86.4
X2=3.14
```

Example 2

```
10 READ A$,B$,C$
20 PRINT A$,B$,C$
30 DATA OSCAR,JOHN,"""PETER"""
RUN
OSCAR        JOHN        "PETER"
```

Note

If a comma, quote or apostrophe is to be part of a string, it must be delimited by quotes.

REM

Format

REM text
! text

Where the text can contain any printing characters on the keyboard. The BASIC interpreter ignores anything following the instruction **REM** or ! on a line.

Function

Inserts a comment in a program.

Use

Notes and messages should be inserted in a program to ensure easy referencing by anyone using the program.

Example

```
10 REM Calculates the mean value
2010 !**** Prints one table line. ****
```

RESTORE

Format

RESTORE [line number]

Function

Enables renewed use of the contents of **DATA** statements.

Example 1

```
60 RESTORE
```

Sets the data pointer to the beginning of the first **DATA** statement in the program.

Example 2

```
50 RESTORE 100
```

Sets the data pointer to the first data of the **DATA** statement with line number 100.

RESUME

Format **RESUME** [line number]

Function Returns from error handler.

Use When the error handling routine has been executed, you can resume execution of the program by means of a **RESUME** statement placed at the end of the error handling routine.

If execution is to be restarted at some other line in the program, the line number should be indicated in the **RESUME** statement.

Example

```
2000 RESUME
2010 RESUME 100
```

Line 2000 returns control to the line that caused the error. Line 2010 returns to line 100.

RETURN

Format **RETURN** [variable]

Function Returns from a subroutine or multiple line function.

RETURN causes a return from a subroutine to the statement immediately following the call.

RETURN <variable> causes a return from a multiple line function with the function value.

Use See **GOSUB** and **DEF FN**.

SINGLE

Format **SINGLE**

Function Changes all variables and expressions, which are floating point numbers, to single precision (7 digits).

Use The **SINGLE** declaration must be placed before the variables are used and cannot be changed once the program has been started by **RUN.** If a line is edited or the command **CLEAR** is given, **SINGLE** may be changed to **DOUBLE** or vice versa. The default is **SINGLE.**

Note **SINGLE** and **DOUBLE** cannot be mixed in the same program.

STOP

Format

STOP

Function

Stops program execution.

Use

The **STOP** instruction stops the execution of the program. The variables are not reset and the open files remain open. The **STOP** instruction is recommended for debugging. Several **STOP** instructions may be present in one program. A **STOP** instruction yields the following display:

```
STOP IN LINE <line number>
```

Note

The program execution can be continued by one of the commands **CON** or **GOTO**.

TRACE

Format

TRACE [£file number]

Function

Prints the line numbers of the program lines executed.

Use

When debugging a program to trace the execution of the program.

Example

```
100 OPEN "PR:" AS FILE 1%
110 A=12.345
115 TRACE £1%
120 B=123
125 IF A=0 THEN STOP
130 C%=B
135 X=A*2
140 D1%=A
145 NO TRACE
150 PRINT £1%A,B,C%,D1%,X
160 CLOSE 1%
170 END
RUN
```

The following text will be printed on the printer:

```
120 125 130 135 140 145
12.345 123 123 12 24.69
```

WEND

Format

WEND

Function

WEND terminates a loop that begins with **WHILE**.

Use

See **WHILE**.

WHILE

Format

WHILE expression

Function

Specifies the condition for the branching out of a program loop.

Use

In program loops where the values that determine the loop termination are modified when the loop is executed. Compare **FOR** loops, where the termination condition will be reached automatically, no matter what the loop contains.

It can often be desirable to execute the loop until a certain value is reached.

Example

```
10 WHILE X<10
20 X=X*X+1
30 WEND
```

Before the first execution of the loop and at the beginning of each new execution the condition X<10 is tested. The iteration will continue for as long as this is true.

ABC 802, 806

10 Functions

10.1 Mathematical Functions

Most programmers often meet with some relatively common mathematical operations. The results of these operations are likely to be found in mathematical tables; sine, cosine, log, square root, etc. Since the computer can perform such operations with speed and accuracy, some of the operations have been built into BASIC II. These intrinsic functions can be called whenever such a value is needed e.g.:

```
SIN(23*PI/180)
LOG(144)
```

The mathematical functions are detailed in the following table:

ABS(X)	The absolute value of X
ATN(X)	The arctangent of X
COS(X)	The cosine of X (X in radians)
EXP(X)	e**X where e=2.71828 (single precision)
FIX(X)	Truncated value of X; **SGN**(X)***INT**(**ABS**(X))
HEX$(X)	Converts a decimal number into a hexadecimal string
INT(X)	The greatest integer <=X
LOG(X)	The natural logarithm of X
LOG10(X)	The common logarithm of X
MOD(X,Y)	The remainder of the integer division X/Y
OCT$(X)	Converts a decimal number into an octal string
PI	Constant; value 3.14159 (single precision)
RND(X)	Random number between 0 and 0.9999999. **RND** will generate the same sequence of random numbers every time the program is run, unless a **RANDOMIZE** instruction is placed before **RND** in the program.
SGN(X)	0 if X=0, -1 if X<0, +1 if X>0.
SIN(X)	The sine of X (X in radians)
SQR(X)	The square root of X
TAN(X)	The tangent of X (X in radians)

ABS

Format **ABS**(argument)

Function The absolute value of the argument.

Example

```
10 Y=ABS(-3.1)
```

The result is Y=3.1

ATN

Format **ATN**(argument)

Function The arctangent of the argument.

Example

```
10 Y=ATN(PI/2)
```

The result is Y=1

COS

Format **COS**(argument)

Function The cosine of the argument (the argument in radians).

Example

```
10 Y=COS(0)
```

The result is Y=1

EXP

Format **EXP**(argument)

Function Gives e**argument where e=2.71828 (single precision).

Example

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

The result is Y=2.71828

FIX

Format **FIX**(argument)

Function Gives the truncated value of the argument (X), i.e. **SGN**(X)***INT**(**ABS**(X)).

Example

```
10 Y%=FIX(-.5)
```

The result is Y=0

Note Compare with **INT**(X).

HEX$

Format **HEX**$(argument)

Function Converts a decimal number into a hexadecimal string.

Example

```
10 Y$=HEX$(255)
```

The result is Y$="FF"

INT

Format **INT**(argument)

Function The value of the greatest integer less than or equal to the argument. Compare with **FIX.**

Use **INT** can be used to round off a number by means of **INT**(X+.5). The **INT** function can be used to round off a number to any given number of decimals using the formula:

INT(X*10**D%+.5)/10**D%

where D% is the required number of decimals.

If the number is negative, **INT** will return the largest integer less than the argument.

Examples

Example 1

```
10 Y=INT(34.67)
```

The result is Y=34

Example 2

```
10 Y=INT(34.67+.5)
```

The result is Y=35

Example 3

```
10 Y=INT(-23.15)
```

The result is Y=-24

LOG

Format **LOG**(argument)

Function The natural logarithm of the argument.

Example

```
10 Y=LOG(2)
```

The result is Y=.693147

LOG10

Format **LOG10**(argument)

Function The common logarithm of the argument.

Example

```
10 Y=LOG10(10)
```

The result is Y=1

MOD

Format **MOD**(argument 1, argument 2)

Function The remainder of an integer division of the arguments.

Example

```
10 Y=MOD(22,4)
```

The result is Y=2

OCT$

Format **OCT$**(argument)

Function Converts a decimal number into an octal string.

Example

```
10 Y$=OCT$(59)
```

The result is Y$="73"

PI

Format **PI**

Function Constant with the value 3.14159 (single precision)

Example

```
10 Y=2*PI
```

The result is Y=6.28318

RND

Format **RND**

Function Returns a random number between 0 and 0.9999999. **RND** will generate the same random number sequence every time the program is run, unless a **RANDOMIZE** instruction is placed before **RND** in the program.

Examples Example 1

```
10 Y=RND
```

Example 2

```
10 Y=(B-A)*RND+A
```

Y will be assigned a random number between A and B.

SGN

Format **SGN**(argument)

Function The function **SGN**(X) has the value +1 if X is positive, 0 if X is 0 and -1 if X is negative.

Examples Example 1

```
10 Y=SGN(3.42)
```

The result is Y=1

Example 2

```
20 Y=SGN(-42)
```

The result is Y=-1

Example 3

```
10 Y=SGN(23-23)
```

The result is Y=0

SIN

Format **SIN**(argument)

Function The sine of the argument (the argument in radians).

Example

```
10 Y=SIN(PI/2)
```

The result is Y=1

SQR

Format **SQR**(argument)

Function The square root of the argument.

Example

```
10 Y=SQR(121)
```

The result is Y=11

TAN

Format **TAN**(argument)

Function The tangent of the argument (the argument in radians).

Example

```
10 Y=TAN(PI/4)
```

The result is Y=1

10.2 String Functions

Besides the intrinsic mathematical functions (e.g. **SIN** and **LOG**) various functions operating on character strings are provided. These functions allow the program to perform arithmetic operations on numeric strings, concatenate two strings, access part of a string, determine the number of characters in a string, generate the character string which corresponds to a given number or vice versa.

The following string functions are provided:

ADD$(A$,B$,[-]P%)	The arithmetic sum of the strings A$ and B$ with P% decimals, or if P% is preceded by a unary minus with P% digits.
ASCII(A$)	The ASCII value of the first character in A$
CHR$(X%)	The character with the ASCII value X%
COMP%(A$,B$)	True or false, numeric comparison
DIV$(A$,B$,[-]P%)	The quotient A$/B$ with P% decimals or digits (-P%)
INSTR(I%,A$,B$)	The starting position of the substring B$ in A$
LEFT$(A$,I%)	The I% characters furthest to the left in A$
LEN(A$)	The number of characters in A$
MID$(A$,P%,K%)	Gives a substring. Assigns a value to a substring
MUL$(A$,B$,[-]P%)	The numeric product A$*B$ with P% decimals or digits(-P%).
NUM$(V)	Numeric string corresponding to the value V
RIGHT$(A$,I%)	The characters furthest to the right in A$ starting at character position I%
SPACE$(N%)	A string consisting of N% spaces
STRING$(I%,C%)	A string consisting of I% characters with the ASCII value C%
SUB$(A$,B$,[-]P%)	The numeric difference A$ - B$ with P% decimals or digits (-P%).
VAL(A$)	The numeric value of A$
A$+B$	Concatenates two strings

ADD$

Format **ADD$**(A$,B$,[-]P%)

Function Adds the values of the strings A$ and B$ with P% decimals, or if P% is proceded by a unary minus with P% digits.

Example

```
10 A$="123.76"
20 B$=ADD$(A$,"957.63359",3)
```

Note — ASCII arithmetic calculations can operate on up to 125 characters.

ASCII

Format — **ASCII**(A$)

Function — Yields an integer equal to the ASCII value of the first character of A$.

Example

```
10 A%=ASCII("T")
```

The result is A%=84

CHR$

Format — **CHR**$(argument[,argument,...])

Function — A character string which corresponds to the ASCII values of the arguments.

Example

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

The result is an A printed on the screen.

COMP%

Format — **COMP**%(A$,B$)

Function — Yields the value -1, 0 or 1 as a result of a numeric comparison of two numeric strings. The function value is -1 if A$ < B$, 0 if A$=B$ and 1 if A$ > B$.

Example

```
30 A$="123.456" : B$="12.8907"
40 T%=COMP%(A$,B$)
50 PRINT T%
60 PRINT COMP%(B$,A$)
100 END
RUN
1
-1
```

DIV$

Format — **DIV**$(A$,B$,[-]P%)

Function — The quotient A$/B$ rounded off to P% decimals, or if P% is preceded by a unary minus with P% digits.

Example

```
100 LET C$="3.5"
110 V9$=DIV$(C$,"1.7777",3%)
120 PRINT V9$
200 END
RUN
1.969
```

Note — ASCII arithmetic calculations can operate on up to 125 characters.

INSTR

Format **INSTR**(N%,A$,B$)

Function Searches for the string B$ in A$ starting at position N%. If B$ is not present in the part of A$ which is searched, the function value is 0. If B$ is found, the function value equals the position in A$ where B$ begins. The position refers to the beginning of the string. The first character occupies position 1.

Example

```
10 A$="AaBbCcDdEeFf"
20 PRINT INSTR(5%,A$,"eF")
30 END
RUN
10
```

LEFT

Format **LEFT**($)(A$,I%)

Function The first I% characters of the string A$.

Example

```
10 D2$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20 T8$=LEFT$(D2$,6%)
40 PRINT T8$
100 END
RUN
ABCDEF
```

LEN

Format **LEN**(A$)

Function The number of characters of the string A$, i.e. the string length (including spaces).

Example

```
10 S$="August"
20 PRINT LEN(S$)
200 END
RUN
6
```

MID

Format **MID**[$](A$,P%,K%)

Function Assigns new values to the characters no.P% to P%+K%-1 in A$, i.e. exchanges the characters indicated in the string.

Example

```
10 A$="ABCDEFGHI"
20 MID$(A$,6%,2%)="MM"
30 PRINT A$
60 END
RUN
ABCDEMMHI
```

MID

Format **MID**[$](A$,P%,K%)

Function Gives the substring of A$, which starts at position P% and has a length of K% characters, i.e. the characters from no.P% to P%+K%-1.

Example

```
200 W$="carriage return"
210 A2$=MID$(W$,6%,3%)
220 PRINT A2$
500 END
RUN
age
```

MUL$

Format **MUL**$(A$,B$,[-]P%)

Function The product A$*B$ with P% decimals, or if P% is preceded by a unary minus with P% digits.

Example

```
10 LET A$="12345.6789"
20 LET B$="987.54321"
30 Y$=MUL$(A$,B$,6%)
40 PRINT Y$
50 END
RUN
12191891.370535
```

NUM$

Format **NUM**$(argument)

Function The numeric string corresponding to the argument. The string is printed as follows: positive number - the sign position is not indicated; negative number - a minus sign is printed.

Example

```
10 PRINT NUM$(345709702134)
20 END
RUN
3.457097E+11
```

RIGHT

Format **RIGHT**[$](A$,N%)

Function The last characters of A$ starting at position N%.

Example

```
10 M$="abcdefghijkl"
20 H$=RIGHT$(M$,7%)
30 PRINT H$
90 END
RUN
ghijkl
```

SPACE$

Format **SPACE**$(N%)

Function Yields a string consisting of N% spaces.

Example

```
10 Y$=SPACE$(10)
```

The result is a string (Y$) containing 10 spaces.

STRING$

Format **STRING**$(I%,K%)

Function Yields a string of I% ASCII characters. The string has the length I% and consists of equal characters with ASCII value K%.

Example

```
10 G5$=STRING$(4%,33%)
20 PRINT G5$
30 END
RUN
!!!!
```

SUB$

Format **SUB**$(A$,B$,[-]P%)

Function The arithmetic difference A$-B$ of the numeric strings A$ and B$ with P% decimals, or if P% is preceded by a unary minus with P% digits.

Example

```
10 LET H$="9876.54321"
20 PRINT SUB$(H$,"98.76",5%)
30 END
RUN
9777.78321
```

Note ASCII arithmetic calculations can operate on up to 125 characters.

VAL

Format **VAL**(A$)

Function Calculates the numeric value of the numeric string A$. A numeric string may contain digits, +, -, ., and E. The result is a floating point number.

Example

```
330 V4=VAL("14.3E-5")
340 PRINT V4
400 END
RUN
.000143
```

A$+B$

Format A$+B$

Function Concatenates two strings.

Example

```
10 D$="NAME"
20 S$="ADDRESS"
30 A$=D$+" and "+S$
```

The result is A$="NAME_and_ADRESS"

10.3 Other Functions

CALL(A%,[D%])	Calls a machine language routine
CUR(M%,N%)	Positions the cursor on line M%, position N%
CVT%$(X%) **CVT**$%(X$)	Converts the variable from an integer into a string and vice versa
CVTF$(X) **CVT**$F(X$)	Converts the variable from a floating point number into a string and vice versa
ERRCODE	Returns the value of the most recent error code
FN	User-defined function
INP(I%)	Yields the data value from input port number I%
OUT	Transfers data to an output port
PEEK(I%)	Returns the contents of storage address I%
PEEK2(I%)	**PEEK**(I%)+256***PEEK**(I%+1%)
POKE	Writes data at the specified storage address
SWAP%(N%)	The first and second byte of N% change places
SYS(I%)	Returns system status information
TAB(I%)	Tabulates to the I%th position on the line
TIME$	Returns the time and date
VAROOT(X)	Returns the address of the variable X
VARPTR(X)	Returns the address of the value of X

CALL

Format

CALL(A%[,D%])

Function

Calls an ASSEMBLY program starting at adress A% (decimal). Yields a function value from the HL register of the Z80 processor when returning to BASIC. If D% is specified, it is placed in the DE register of the Z80 processor at the call.

CAUTION

This function is machine-oriented, and should only be used for advanced programming. **CALL** can destroy a program execution if used erroneously.

CUR

Format

CUR(L%,N%) ***ABC 806***

where L% (line) is in the interval 0-23 and N% (position) in the interval 0-39/79.

Function

Moves the cursor to line L%, position N%.

Use

When printing or with the graphics

Example

```
10 PRINT CUR(12,20)"Text"
```

CVT

Format

CVT%$(integer variable)
CVT$%(string variable)

CVTF$(floating point variable)
CVT$F(string variable)

Function

Stores numbers as strings or regains the numbers.

Use

The **CVT** function is used to save disk space. Numeric values that are stored on disk require as much space as when they are printed by means of **PRINT**. Integers require up to six characters, floating point numbers in single precision (**SINGLE**) twelve characters, and floating point numbers in double precision (**DOUBLE**) require up to twenty-two characters. Each character is stored in one byte. By means of the **CVT** (from convert) function these data can be stored in 2, 4, and 8 bytes, respectively.

Examples

```
10 PREPARE "NUM.DAT" AS FILE 1
20 I%=15973%
30 PUT £1,CVT%$(I%)
40 CLOSE 1
```

The integer I% is stored on the file NUM.DAT in two bytes. To regain the number, proceed as follows:

```
10 OPEN "NUM.DAT" AS FILE 1
20 GET £1,A$ COUNT 2
30 I%=CVT$%(A$)
40 CLOSE 1
```

The example below shows how to store a floating point number. The number may have either single or double precision:

```
10 DIM A(100)
20 PREPARE "NUM2.DAT" AS FILE 1%
30 FOR I%=1% TO 100%
40 PUT £1%,CVTF$(A(I%))
50 NEXT I%
60 CLOSE 1%
```

The next example shows how to regain the number stored in the example above:

```
10 DIM A(100)
15 L%=LEN(CVTF$(0))
20 OPEN "NUM2.DAT" AS FILE 1%
30 FOR I%=1% TO 100%
40 GET £1%,A$ COUNT L%: A(I%)=CVT$F(A$)
50 NEXT I%
60 CLOSE 1%
```

Note

LEN(CVTF$(0)) is used to find out if the precision is **SINGLE** or **DOUBLE.**

ERRCODE

Format

ERRCODE

Function

Returns the value of the latest generated error code. If no error has been indicated, the function value is 0.

FN

Format

FNidentifier[%/$] [(parameter [,parameter,...])]

where <identifier> is the name of the function.

Function

Calls a user-defined function.

Note

Compare with the instruction **DEF FN.**

INP

Format

INP(I%)

Function

Returns a data value from the input port I%.

CAUTION

This function is machine oriented, and should only be used for advanced programming.

OUT

Format	**OUT** port,data [,port,data, ...] where the port numbers and the data are given as decimal numbers.
Function	Addresses the out ports at data output.
CAUTION	This is a machine-oriented instruction intended for advanced programming. This instruction and the instruction **INP** give the user access to the I/O-functions and the I/O-bus of the ABC 800.
Note	The I/O channel is selected by means of the **OUT** instruction. That channel will remain accessible until a new selection is made by means of **OUT**.

PEEK

Format	**PEEK**(I%)
Function	Gives the contents of storage address I%.
CAUTION	This function is intended for advanced programming.

PEEK2

Format	**PEEK2**(B0%)
Function	Reads the contents of two bytes in the following way: J%=**PEEK**(B0%)+256***PEEK**(B0%+1%)
CAUTION	This function is intended for advanced programming.

POKE

Format	**POKE** address,data [,data, ...] where the <address> is a decimal number. If more than one <data> item is specified, the address will be incremented for each new item.
Function	Loads a value into a storage cell.
Use	**POKE** is mainly used when BASIC cooperates with ASSEMBLER language programs.
CAUTION	This function is intended for advanced programming. If used erroneously, it may destroy the contents of the computer storage.

SWAP%

Format

SWAP%(N%)

Function

The first and the second bytes of N% change places.

CAUTION

This function is machine-oriented, and should only be used for advanced programming.

SYS

Format

SYS(I%)

ABC 802, 806 Function

System status information as follows:

SYS (2) Total storage space
SYS (3) Program size
SYS (4) Remaining storage space
SYS (5) Keyboard flag. Can be cleared by means of **GET, INPUT,** or **INPUT LINE.**
SYS (6) Puts back the last input character into the keyboard buffer.
SYS(11) Starting address of the program
SYS(12) Variable root

Example

```
PRINT SYS(3)
```

The result is the program size.

TAB

Format

TAB(I%)

where I% is in the interval 1 - 40/80.

Function

Tabulates to the I%-th position on the line.

Example

```
10 PRINT TAB(20)"Data"
```

TIME$

Format

TIME$

Function

Returns year-month-day hour.min.sec

Use

The internal clock can be set by means of the following program:

```
10 PRINT CHR$(12%)
20 PRINT "** ABC 800 Set the clock! **"
30 INPUT "Date: YY,MM,DD";Y%,M%,D% !Type date
40 INPUT "Time: HH,MM,SS";H%,M1%,S% !Type time
50 POKE -17,Y%,M%,D%,H%,M1%,S% !Store time,day
60 PRINT CUR(12,12);TIME$ !Display the time
70 GOTO 60
80 END
```

VAROOT

Format **VAROOT** (variable)

Function Gives the address of a table, which contains information about a variable.

CAUTION This function is intended for advanced programming.

VARPTR

Format **VARPTR** (variable)

Function Gives the address of the value of a variable.

CAUTION This function is intended for advanced programming.

BC 802, 806

11 Graphics and Colours

11.1 General Information

ABC 800 C graphics correspond to the Teletext standards. In the graphic mode every output character is interpreted as a graphic character formed by a combination of six graphic points.

When text or graphics are displayed on the screen, the selection of colours etc. is controlled by means of certain arguments in the **PRINT** statement. The statement affects one line at a time. Each argument puts a control character on the screen. Although these characters are invisible, they take up one position each. The control characters can be covered by a background colour, if the control arguments are given in the correct order.

The following colours are available:

Red (**RED**)
Green (**GRN**)
Yellow (**YEL**)
Blue (**BLU**)
Magenta (**MAG**)
Cyanide (**CYA**)
White (**WHT**)

The characthers available in the ABC 800 are listed below. The table gives the ASCII value of each character and its meaning in the character mode and graphic mode. One way of planning a graphical picture is to draw it on a copy of the graphics chart and feed the program the appropriate data.

When you have finished the picture on a copy of the chart you can type the lines one by one. Do not forget to allow space for the control characters, if you vary the control arguments.

Note that the capital letters still remain the same in graphic mode. You can mix capital letters and graphic characters just as you like.

In graphic mode there are 72 graphic lines (0-71), each one with 78 graphic positions (0-77).

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

*) Not generated from keyboard

A	C	G	A	C	G	A	C	G	A	C	G
32	Space	□	56	8	🬖	80	P	P	104	h	🬦
33	!	🬀	57	9	🬗	81	Q	Q	105	i	🬧
34	"	🬁	58	:	🬘	82	R	R	106	j	▐
35	£	🬂	59	;	🬙	83	S	S	107	k	🬨
36	$	🬃	60	<	🬚	84	T	T	108	l	🬩
37	%	🬄	61	=	🬛	85	U	U	109	m	🬪
38	&	🬅	62	>	🬜	86	V	V	110	n	🬫
39	'	🬆	63	?	🬝	87	W	W	111	o	🬬
40	(	🬇	64	@	@	88	X	X	112	p	🬭
41	)	🬈	65	A	A	89	Y	Y	113	q	🬮
42	*	🬉	66	B	B	90	Z	Z	114	r	🬯
43	+	🬊	67	C	C	91	←	←	115	s	🬰
44	,	🬋	68	D	D	92	1/2	1/2	116	t	🬱
45	-	🬌	69	E	E	93	→	→	117	u	🬲
46	.	🬍	70	F	F	94	↑	↑	118	v	🬳
47	/	🬎	71	G	G	95	#	#	119	w	🬴
48	0	🬏	72	H	H	96	—	🬞	120	x	🬵
49	1	🬐	73	I	I	97	a	🬟	121	y	🬶
50	2	🬑	74	J	J	98	b	🬠	122	z	🬷
51	3	🬒	75	K	K	99	c	🬡	123	1/4	🬸
52	4	🬓	76	L	L	100	d	🬢	124	‖	🬹
53	5	▌	77	M	M	101	e	🬣	125	3/4	🬺
54	6	🬔	78	N	N	102	f	🬤	126	÷*	🬻
55	7	🬕	79	O	O	103	g	🬥	127	■	█

*) Not generated from keyboard

ASCII codes (A) for character mode (C) and graphic mode (G).

ABC800® VIDEO GRAPHICS CHART PROGRAM............ ....

0	71 70 69	▨																																						
1	68 67 66	▨																																						
2	65 64 63	▨																																						
3	62 61 60	▨																																						
4	59 58 57	▨																																						
5	56 55 54	▨																																						
6	53 52 51	▨																																						
7	50 49 48	▨																																						
8	47 46 45	▨																																						
9	44 43 42	▨																																						
10	41 40 39	▨																																						
11	38 37 36	▨																																						
12	35 34 33	▨																																						
13	32 31 30	▨																																						
14	29 28 27	▨																																						
15	26 25 24	▨																																						
16	23 22 21	▨																																						
17	20 19 18	▨																																						
18	17 16 15	▨																																						
19	14 13 12	▨																																						
20	11 10 9	▨																																						
21	8 7 6	▨																																						
22	5 4 3	▨																																						
23	2 1 0	▨																																						
	DOT	▨	0 1	2 3	4 5	6 7	8 9	10 11	12 13	14 15	16 17	18 19	20 21	22 23	24 25	26 27	28 29	30 31	32 33	34 35	36 37	38 39	40 41	42 43	44 45	46 47	48 49	50 51	52 53	54 55	56 57	58 59	60 61	62 63	64 65	66 67	68 69	70 71	72 73	74 75	76 77
CHR		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39

▨ - reserved for graphic control characters

11.2 Instructions

ABC 806

PRINT

ABC 802, 806

Format

PRINT [**CUR**(L,N)]argument [;argument;...]''text''

Function

Used for printing text and graphics. The arguments control the colour selection etc. A G at the beginning of the colour selection argument (e.g. **GRED**) sets the line to the graphic mode so that all characters within quotes are interpreted as being graphics (see the ASCII table). If **CUR**(L,N) is specified, the picture is drawn from the starting point at line L (0-23), position N (0-39/79).

The following arguments are available:

RED,GRN,YEL,BLU, **MAG,CYA,WHT**	Alphanumeric colour characters
GRED,GGRN,GYEL,GBLU,GMAG, **GCYA,GWTH**	Colour graphics
FLSH,STDY	Flashing, steady
NRML,DBLE	Normal, double height
GCON,SEP	Continuous, separated graphics
NWBG,BLBG	New background, black background
GHOL,GREL	Hold, release graphics
HIDE	Concealed text/graphics

The control arguments should be given in the following order:

PRINT <background colour argument> <argument for new background colour> <text colour argument> ''Text'' <argument for black background>

Example

```
10 PRINT RED NWBG GYEL "I,6 BOW WOW"
```

The result is a yellow ''dog'' on red background

TXPOINT

ABC 802, 806

Format

TXPOINT X,Y[,1/0]

Function

Turns on (1) or off (0) a graphical point in position X, Y, where X=0-77 and Y=0-71.

Example

```
10 PRINT CHR$(12)
20 FOR I=0 TO 23
30 PRINT CUR(I,0) GGRN;
40 NEXT I
50 FOR I=0 TO 77
60 TXPOINT I,32+SIN(I/5)*30
70 NEXT I
80 PRINT CUR(0,15) RED FLSH DBLE "SINE"
90 END
```

The lines 10 - 40 clear the screen and set it to the graphic mode (green). The lines 50 - 70 draw a sine curve. Line 80 displays SINE in red, flashing and with double height.

TXPOINT can be used as a function, too, to check if a point is turned on (-1) or off (0). **TXPOINT**(X,Y).

Note — The origin is in the lower, left-hand corner.

ABC 802

SET DOT

Format — **SET DOT** L%,N%

Function — Turns on the graphic point in position L%, N%, where L%=0-71 and N%=2-79.

Note — The origin is in the upper left-hand corner.

ABC 802

CLR DOT

Format — **CLR DOT** L%,N%

Function — Turns off the graphic point in position L%, N%, where L%=0-71 and N%=2-79.

Note — The origin is in the upper, left-hand corner.

ABC 802

DOT

Format — **DOT**(L%,N%)

Function — Will be -1 (true) if the point is lit, else 0 (false). L%=line (0-71), N%=position (2-79)

Note — The origin is at the upper, left-hand corner.

12 High Resolution Graphics

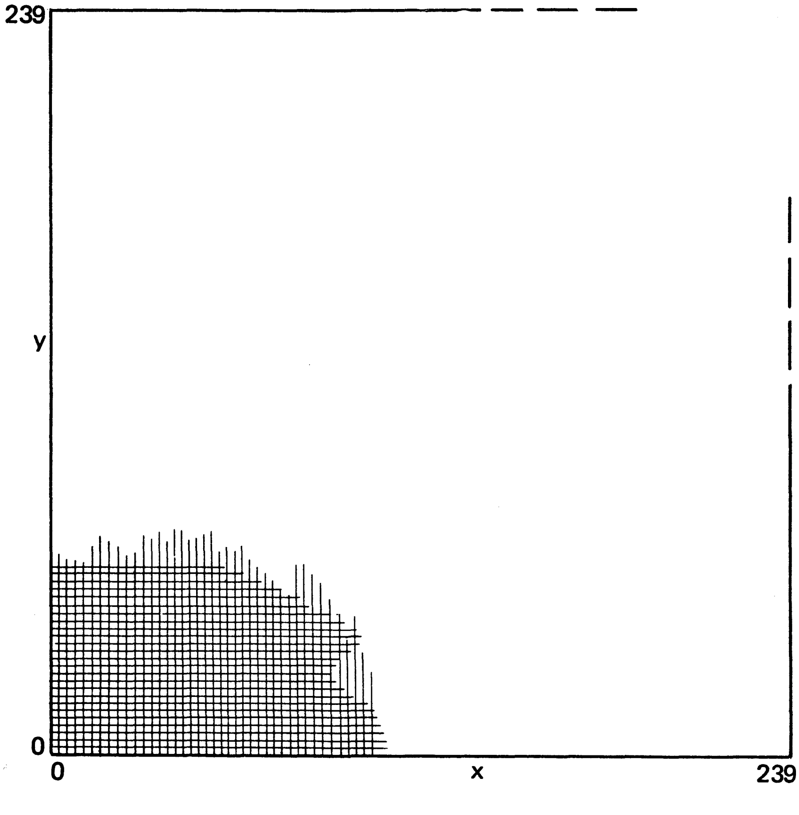
ABC 802, 806

12.1 General Information

ABC 806

High resolution graphics, which is an option, can be used with the ABC 800 C as well as with the ABC 800 M.

The screen is subdivided into 240 x 240 picture elements (pixels). Each pixel can be addressed directly and is independent of the others. Two data bits correspond to each pixel. The data bits are used to select one of four colours. High resolution graphics can be shown together with the usual text or graphics display. The origin of the picture is in the lower left-hand corner and the positions are numbered from 0 to 239.



The screen is adjusted to obtain the following relations between height and width:

	Height (mm)	Width (mm)	W/H
ABC 810	185 +- 2	225 +- 2	1.2
ABC 815	166 +- 2	250 +- 2	1.5

12.2 Instructions

- The colour number is a digit from 0 to 3, where 0 indicates the background colour. The meaning of the digit is listed in the colour selection table (chapter 12.4)

- The colour number is optional. If no colour number is given, the previous colour number will be used.

- The starting position of the picture is selected by means of **OUT** 6,line number, where line number is in the interval 0 - 255.

FGCTL

Format **FGCTL** colour selection command

Where the <colour selection command> is in accordance with the colour selection table (chapter 12.4)

Function Selects the colour combination to be used. Each combination consists of four of the colours available. Black and white are treated as colours.

FGFILL

Format **FGFILL** x,y[,colour number]

Function Fills a rectangle from the previous position to the position indicated by the coordinates (x,y).

FGLINE

Format **FGLINE** x,y[,colour number]

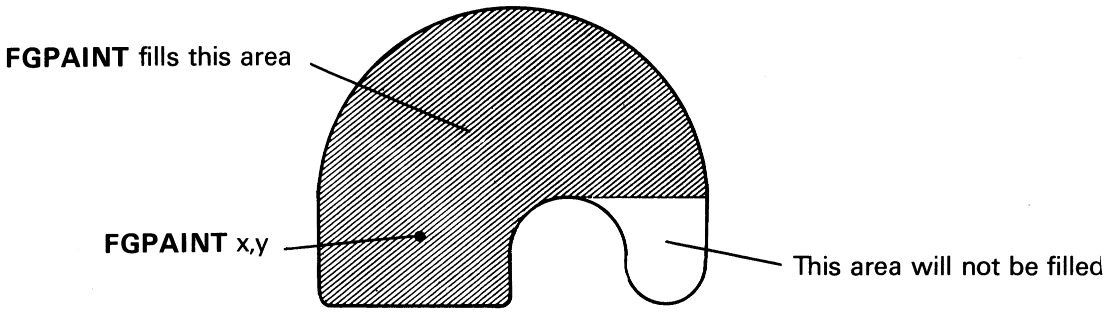
Function Draws a line from the previous position to the position indicated by the coordinates (x,y).

FGPAINT

Format **FGPAINT** x,y[,colour number]

Function Fills a closed area.

Note The function has certain limitations.



FGPOINT

Format **FGPOINT** x,y[, colour number]

Function Will turn on a pixel in position x,y, x=0-239, y=0-239

FGPOINT

Format **FGPOINT** (x,y)

Function Returns the colour number of pixel x,y.

12.3 Animation Mode

ABC 806

Two colours are used in the animation mode.
The following procedure can be used:

1. Pick out a colour selection group (72-127, 200-255). The colour selection groups are used two and two together e.g. 72-73, 74-75 .
2. Draw a picture with colour number 1 or 2. Select the same colour as the one the picture is drawn on. The picture cannot be seen.
3. Change the colour selection group so that the picture that was drawn in point 2 above becomes visible.
4. Draw a new picture according to point 2 above.
5. Change the colour selection group so that the picture that was drawn in point 2 disappears and the one drawn in point 4 will show.
6. Erase the picture drawn in point 2 and draw a new one.
7. Change the colour selection group so that the picture drawn in point 6 becomes visible.

Repeat the procedures under points 6 and 7.

To protect the current picture until a new picture is to be shown use the following method:

```
100 FGLINE 100,100,256*2+1
```

This instruction will cause a line to be drawn from the previous position to the point 100,100 with colour number 1. Colour number 2 is protected and will not be changed.

ABC 806

12.4 Colour Selection Table

The colour selection command (according to the table below) is in the interval 0-255. Values less than 128 mean that the ordinary text and graphics are displayed on top of the high resolution graphics. From 128 upwards the high resolution graphics memory is displayed. The values from 72 to 127 and 200 to 255 are used in animation mode (see above).

B=blue, C=cyanide, Y=yellow, GR=green, M=magenta, R=red, BK=black, W=white

Selection Command Graphics + text		Colour			Selection Command Graphics only
	0	1	2	3	
0	BK	BK	BK	BK	128
1	BK	W	W	W	129
2	BK	R	GR	Y	130
3	BK	R	GR	B	131
4	BK	R	GR	M	132
5	BK	R	GR	C	133
6	BK	R	GR	W	134
7	BK	R	Y	B	135
8	BK	R	Y	M	136
9	BK	R	Y	C	137
10	BK	R	Y	W	138
11	BK	R	B	M	139
12	BK	R	B	C	140
13	BK	R	B	W	141
14	BK	R	M	C	142
15	BK	R	M	W	143
16	BK	R	C	W	144
17	BK	GR	Y	B	145
18	BK	GR	Y	M	146
19	BK	GR	Y	C	147
20	BK	GR	Y	W	148
21	BK	GR	B	M	149
22	BK	GR	B	C	150
23	BK	GR	B	W	151
24	BK	GR	M	C	152
25	BK	GR	M	W	153
26	BK	GR	BK	W	154
27	BK	Y	B	M	155
28	BK	Y	B	C	156
29	BK	Y	B	W	157
30	BK	Y	M	C	158
31	BK	Y	M	W	159
32	BK	Y	C	W	160
33	BK	B	M	C	161

Selection command Graphics + text		Colour			Selection command Graphics only
	0	1	2	3	
34	BK	B	M	W	162
35	BK	B	C	W	163
36	BK	M	C	W	164
37	R	GR	Y	B	165
38	R	GR	Y	M	166
39	R	GR	Y	C	167
40	R	GR	Y	W	168
41	R	GR	B	M	169
42	R	GR	B	C	170
43	R	GR	B	W	171
44	R	GR	M	C	172
45	R	GR	M	W	173
46	R	GR	C	W	174
47	R	Y	B	M	175
48	R	Y	B	C	176
49	R	Y	B	W	177
50	R	Y	M	C	178
51	R	Y	M	W	179
52	R	Y	C	W	180
53	R	B	M	C	181
54	R	B	M	W	182
55	R	B	C	W	183
56	R	M	C	W	184
57	GR	Y	B	M	185
58	GR	Y	B	C	186
59	GR	Y	B	W	187
60	GR	Y	M	C	188
61	GR	Y	M	W	189
62	GR	Y	C	W	190
63	GR	B	M	C	191
64	GR	B	M	W	192
65	GR	B	C	W	193
66	GR	M	C	W	194
67	Y	B	M	C	195
68	Y	B	M	W	196
69	Y	B	C	W	197
70	Y	M	C	W	198
71	B	M	C	W	199
72	BK	R	BK	R	200
73	BK	BK	R	R	201
74	BK	GR	BK	GR	202
75	BK	BK	GR	GR	203
76	BK	Y	BK	Y	204
77	BK	BK	Y	Y	205
78	BK	B	BK	B	206
79	BK	BK	B	B	207
80	BK	M	BK	M	208
81	BK	BK	M	M	209
82	BK	C	BK	C	210
83	BK	BK	C	C	211
84	BK	W	BK	W	212

Selection command Graphics + text		Colour			Selection command Graphics only
	0	1	2	3	
85	BK	BK	W	W	213
86	R	GR	R	GR	214
87	R	R	GR	GR	215
88	R	Y	R	Y	216
89	R	R	Y	Y	217
90	R	B	R	B	218
91	R	R	B	B	219
92	R	M	R	M	220
93	R	R	M	M	221
94	R	C	R	C	222
95	R	R	C	C	223
96	R	W	R	W	224
97	R	R	W	W	225
98	GR	Y	GR	Y	226
99	GR	GR	Y	Y	227
100	GR	B	GR	B	228
101	GR	GR	B	B	229
102	GR	M	GR	M	230
103	GR	GR	M	M	231
104	GR	C	GR	C	232
105	GR	GR	C	C	233
106	GR	W	GR	W	234
107	GR	GR	W	W	235
108	Y	B	Y	B	236
109	Y	Y	B	B	237
110	Y	M	Y	M	238
111	Y	Y	M	M	239
112	Y	C	Y	C	240
113	Y	Y	C	C	241
114	Y	W	Y	W	242
115	Y	Y	W	W	243
116	B	M	B	M	244
117	B	B	M	M	245
118	B	C	B	C	246
119	B	B	C	C	247
120	B	W	B	W	248
121	B	B	W	W	249
122	M	C	M	C	250
123	M	M	C	C	251
124	M	W	M	W	252
125	M	M	W	W	253
126	C	W	C	W	254
127	C	C	W	W	255

12.5 Examples

Example 1

```
 10 FGPOINT 0,0,0 :REM Sets pixel 0,0 to colour no. 0
 15 PRINT CHR$(12) :REM Clears the display storage
 20 FGFILL 239,239 :REM Clears the high resolution storage
 30 FGCTL 3 :REM Selects colours BK, R, GR, B + text
 35 REM Draw a square
 40 FGPOINT 20,20,2 :REM Sets pixel 20,20 to colour 2 (GR)
 50 FGLINE 220,20 :REM Draws a line to 220,20 in colour 2
 60 FGLINE 220,220,3 :REM Draws a line to 220,220, in colour 3 (B)
 70 FGL!NE 20,220,2 :REM Draws a line to 20,220 in colour 2 (GR)
 80 FGLINE 20,20 :REM Draws a line to 20,20 in colour 2
 90 PRINT CUR(12,15);CYA DBLE "SQUARE";
100 END
```

Example 2

```
 10 !Draw a circle
 20 EXTEND
 30 C=1.2 !Width/height correction factor
 40 ;CHR$(12) !Clears the screen
 50 FGPOINT 0,0,0 !Sets pixel 0,0 to colour 0
 60 FGFILL 239,239 !Fills the screen with colour 0
 70 FGCTL 7 !Selects a colour combination
 80 Origin=119
 90 Radius=95
100 Colour=3
110 FGPOINT Origin+Radius,Origin,Colour
120 WHILE Xposition<=2*PI
130 FGLINE COS(Xposition)*Radius+Origin, SIN(Xposition)*Radius*C+Origin
140 Xposition=Xposition+1/18
150 WEND
160 FGPAINT Origin,Origin
170 END
```

ABC 802, 806

13 Function Keys

The computer has eight function keys that are situated between the alphanumeric and the numeric keys. The function keys are labelled PF1, PF2, ..., PF8.

A programmer can assign various functions to the function keys, e.g. cursor movements or a jump to a program module.

The function keys can produce 32 different ASCII values as shown in the following table:

		SHIFT	CTRL	SHIFT+CTRL
PF1	192	208	224	240
PF2	193	209	225	241
PF3	194	210	226	242
PF4	195	211	227	243
PF5	196	212	228	244
PF6	197	213	229	245
PF7	198	214	230	246
PF8	199	215	231	247

When a function key is pressed, a subroutine can be called as shown below:

Example:

```
10 ON ERROR GOTO 100
20 INPUT "Number ",P(I)
30 I=I+1
40 GOTO 20
I
I
100 IF ERRCODE <> 53 THEN RESUME
110 A=SYS(6)
120 GET X$
130 ON ASC(X$)-191 RESUME 400, 500, 600
```

When a function key is pressed at **INPUT** or **INPUT LINE**, an error is generated. The **ERRCODE** is 53. The program should contain a routine which handles error 53. To find out which one of the function keys that was pressed, use the function **SYS**(6) and read the character by means of **GET**.

14 Differences in BASIC between ABC 800 and ABC 80

ABC 806

The changes, which have been made in relation to the ABC 80 BASIC, are adjustments to the ANSI standards. The memory mapping and the internal code have also been changed.

1. When an integer variable is assigned a floating point value, the value will be rounded off.

   Example:
   ```
   A%=3.567
   ```
   The variable A% is assigned the value:
   ABC 80: 3
   ABC 800: 4

2. When **TAB** is used for printouts, the printing position is specified starting at **TAB**(1).

   Example:
   ```
   PRINT TAB(5)"B"
   ```
   ABC 80: B is printed at position 6 (i.e. the positions are 0-39)
   ABC 800: B is printed at position 5 (i.e. the positions are 1-40/80)

3. When the value of a variable is printed using **NUM$**, the position which was meant for the + sign is no longer used.

   Example:
   ```
   20 I=1234
   30 PRINT NUM$(I)
   ```
   The result is:
   ABC 80 : 1234
   ABC 800:1234

4. When numeric variables that are to be printed are separated by a semicolon, an extra space will be printed between them.

   Example:
   ```
   PRINT X;Y
   ```
   results in the following printout:
   ABC 80:0 0
   ABC 800:0   0

5. The **CALL** instructions for file access are replaced by **POSIT**, **GET** - **COUNT**, and **PUT**.

   Example: Reading
   ABC 80:
   Z=**CALL**(28666,file number)+**CALL**(28668,sector number)
   ABC 800:
   POSIT £file number,sector number*253 : **GET**£file number, Q0$ **COUNT** 253
   Writing
   ABC 80:
   Z=**CALL**(28666,file number) : Q0$=A$ : Z=**CALL**(28670,sector number)
   ABC 800:
   POSIT £file number,sector number*253: **PUT** £file number,A$

6. The **CHAIN** "" instruction is removed or changed to **END** in ABC 800 programs. *ABC 802*
7. The **END** instruction should be the only instruction on the line. **END** closes all files but does not clear the variables.
8. The instruction **ON ERROR GOTO** replaces **ON ERROR GOTO** 0.
9. An ABC 80 program can be tranformed to ABC 800 if it is stored in text format (.BAS) i.e. by means of the **LIST** <file name> command. The lines which are incompatible will cause error messages. A question mark following the line number will indicate such a line.

15 Error Messages

Error no. 19–68: I/O errors
Error no. 120–129: ISAM errors
Error no. 130–176: Errors during program execution
Error no. 180–191: Logical errors
Error no. 200–211: General errors
Error no. 220–234: Formal BASIC errors

Error	Message	Comment
19	Cannot open more files	Seven files are open
20	Line overflow (> 160 characters)	A line may contain a maximum of 160 characters
21	File not found	The file is not present or has been called by a wrong name
32	File not opened	
34	End of file	Attempt to read after EOF (end of file)
35	CRC or AM error during read	The disk or the cassette tape is damaged
36	CRC or AM error during write	The disk is damaged
37	Incorrect sector format	Disk or cassette error
38	Sector number outside the file	Attempt to read further than the file allows
39	File write-protected	
40	File delete-protected	
41	Disk space full	The disk is too full to accomodate the file
42	Disk not ready	No disk present or the flap is open
43	Disk write-protected	
44	Logical file not opened	
45	Illegal logical file number	
46	Illegal unit number	
47	Illegal trap number	
48	Failure in system data	
49	Incorrect physical file number	Disk error
51	Unit busy	
52	Illegal device operation	
53	Function key	Function key has been pressed in **INPUT** or **INPUT LINE** statement
54	IEC, both talker and listener	IEC option
55	IEC, listener not active	IEC option
56	IEC, talker not active	IEC option
57	Character from keyboard too late	
58	Invalid character loaded	
64	Incorrect "**NAME**"	The new file name already exists
68	Incorrect time specification	
120	The key does not exist	ISAM option
121	Double key	ISAM option
122	Wrong key	ISAM option
123	Error at check reading	ISAM option
124	Index does not exist	ISAM option

ABC 802, 80

Error	Message	Comment
125	Wrong post length	ISAM option
126	Wrong ISAM file version	ISAM option
127	Reserved code	ISAM option
128	End of memory in the central	ISAM option
129	Reserved code	ISAM option
130	Floating point overflow	
131	Index outside array	Attempt to use an index greater than the DIM allows
132	Integer overflow	
133	Error in ASCII arithmetic expression	
134	Index outside string	Index is too great or negative
135	Negative "**SPACE$**", "**STRING$**", or "**TAB**" < 1	
136	String too long	The dimensions of the receiving string are too small
137	Extending "**DIM**" not permitted	A vector cannot be extended beyond its original length
138	Incorrect value in "**ON**" expression	
139	"**RETURN**" without "**GOSUB**"	A **RETURN** statement is encountered when no **GOSUB** has been executed
140	Incorrect "**RETURN**" variable	
141	End of data	The data list is exhausted and a **READ** statement wants more data
142	Incorrect argument in function	
143	Incorrect "**SYS**" function	
144	Invalid line	
145	"**FNEND**" not preceded by "**RETURN**"	
146	"**PRINT USING**" error	Wrong format in **PRINT USING** statement
147	Wrong data	
148	Too little input data	Too few data items typed at **INPUT**
149	"**RESTORE**" not on a "**DATA**" line	
150	Too much input data	Too many data items typed at **INPUT**
151	"**RESUME**" without error	
176	Graphic dot outside screen	
180	Cannot find this line number	Reference to a non-existant linenumber
181	Incorrect jump into function	
182	"**NEXT**" or "**WEND**" missing	
183	"**FOR**" or "**WHILE**" missing	
184	Wrong variable after "**NEXT**"	
185	Mixed "**FOR**" loops with same variable	
186	"**FOR**" loop with local variable not permitted	Applies to multiple line functions
187	Function not defined	Call for undefined function
188	More than one function with same name	
189	Incorrect function	Mixing of several **DEF** instructions is not allowed

Error	Message	Comment
190	Wrong number of indexes	The number of indexes is not in accordance with the **DIM** statement
191	Not allocable (in a function)	The argument of the function cannot be assigned within the function
200	Unit not connected	
201	End of memory	Not enough space for program and data in the main storage
202	"**LIST**"-protected program	
203	Incorrect program format	The program is saved under an incompatible BASIC version
204	"**MERGE**" cannot be used on "BAC" file	
205	"**COMMON**" error	
206	Use "**RUN**" command	
207	Cannot continue	Applies to **GOTO** line number and **CON**
208	Invalid as a command	The instruction cannot be used as a command
209	Wrong data with command	Wrong argument to the command e.g. **LIST** ££
210	Incorrect number	The number contains other characters than digits
211	Precision must not be changed	Change of precision after assignment not allowed
220	Spelling error	Formal BASIC error
221	Illegal character after statement	Formal BASIC error. The computer expects Return, colon (:) or exclamation point (!)
222	Must be first on a line	
223	Wrong number or types or arguments	
224	Illegal mixture of numbers and strings	
225	Not single variable	Indexed variable not allowed e.g. in a **FOR** loop
226	Wrong statement after "**ON**"	Formal BASIC error
227	"," missing	Formal BASIC error
228	"=" missing	
229	")" missing	Formal BASIC error
230	"**AS FILE**" missing	In **OPEN** and **PREPARE** instructions
231	"**AS**" missing	Error in **NAME** ... **AS** ...
232	"**TO**" missing	In **FOR** loops
233	Line number missing	
234	Wrong variable	

16 Summary of Commands and Instructions

ABS (function) Page 63
Format **ABS**(argument)
Function The absolute value of the argument.

ADD$ (function) Page 67
Format **ADD$(A$,B$,[-]P%)**
Function Adds the values of the strings A$ and B$ to P% decimals, or digits (-P%).

ASCII (function) Page 68
Format **ASCII**(A$)
Function The ASCII value of the first character of A$.

ATN (function) Page 63
Format **ATN**(argument)
Function The arctangent (in radians) of the argument.

ABC 806

AUTO (command) Page 25
Format **AUTO** [argument 1[, argument 2]]
where <argument 1> specifies the first line number to be written and <argument 2> specifies the line interval.
Function Automatic line numbering.

$BAS (command under DOS) Page 26
Format **$BAS**
Function Transfers control to BASIC.

BYE (command) Page 26
Format **BYE**
Function Transfers control to the disk operating system (DOS).

CALL (function) Page 73
Format **CALL(A%[,D%])**
Function Calls an assembler program.
CAUTION **CALL** can destroy a program execution if used erroneously.

CHAIN (instruction) Page 35
Format **CHAIN** "file name.extension"/string variable
Function Loads and executes a program.

CHR$ (function) Page 68
Format **CHR$**(argument[,argument,...])
Function A character string which corresponds to the ASCII values of the arguments.

CLEAR (command) Page 26
Format **CLEAR**
Function Clears all variables and closes all open files.

CLOSE (instruction) Page 35
Format **CLOSE** [file number, ...]
Function Closes the file(s).

ABC 806 **CLR DOT** (instruction - graphics) Page 82
Format **CLR DOT** L%,N%
Function Turns off the graphic point on line L% (0-71) and position N% (2-79).
Note The origin is in the upper, left-hand corner.

COMMON (instruction) Page 35
Format **COMMON** variable[(n,...)] [,variable, ...]
COMMON string variable[(n,...)]=length [,string variable=length, ...]
Function A declaration of the variables, whose values are to be transferred to another program.

COMP% (function) Page 68
Format **COMP%**(A$,B$)
Function Comparison of two numeric strings.

CON (command) Page 26
Format **CON** (or **CONTINUE**)
Function Will continue program execution.

COS (function) Page 63
Format **COS**(argument)
Function The cosine of the argument (the argument in radians).

CUR (function) Page 73
Format **CUR**(L%,N%)
Function Moves the cursor to line L%(0-23), position N%(0-39/79).

CVT (function) Page 73
Format **CVT**%$(integer variable)
CVT$%(string variable)
CVTF$(floating point variable)
CVT$F(string variable)
Function Stores numbers as strings or recovers the numbers.

DATA (instruction) Page 36
Format **DATA** value [,value, ...]
Function Assigns values to variables (used with **READ**).

DEF (instruction) Page 36
Format Single line function:
DEF FNidentifier[(argument)]=function
Multiple line function:
DEF FNidentifier[%/$] [(argument)] [**LOCAL** variable [, variable, ...]]
Function Defines single line and multiple line functions.

DIGITS (instruction) Page 38
Format **DIGITS** number of digits
Function Gives the number of digits to be printed.

DIM (instruction) Page 39
Format **DIM** variable(n)[, variable(n,...), ...]
DIM string variable[(n,...)] [=length]
Function Allocates space for strings and vectors.

DIV$ (function) Page 68
Format **DIV$**(A$,B$,[-]P%)
Function The quotient A$/B$ rounded off to P% decimals/digits.

DOT (instruction - graphics) Page 82 ***ABC 806***
Format **DOT**(L%,N%)
Function Will be -1 (true) if the point is lit, else 0 (false).
L%=line (0-71), N%=position (2-79)
Note The origin is at the upper, left-hand corner.

DOUBLE (instruction) Page 40
Format **DOUBLE**
Function Floating point numbers are double precision (16 digits).

ED (command) Page 27
Format **ED** [line number]
Function Starts program editing.

END (instruction) Page 40
Format **END**
Function Terminates the program.

ERASE (command) Page 28
Format **ERASE** line number I [- line number II]
ERASE line number -
ERASE - line number
Function Erases one or more program lines.

ERRCODE (function) Page 74
Format **ERRCODE**
Function The value of the latest generated error code.

EXP (function) Page 63
Format **EXP**(argument)
Function Gives e** argument.

EXTEND (instruction) Page 40
Format **EXTEND**
Function Allows long variable names.

FGCTL (instruction - high res. graphics) Page 84 ***ABC 802***
Format **FGCTL** colour selection command
Function Selects the colour combination.

ABC 802 **FGFILL** (instruction - high res. graphics) Page 84
Format **FGFILL** x,y[,colour number]
Function Fills a rectangle from the previous position to the position indicated by the coordinates (x,y).
Note The origin is at the lower left-hand corner.

ABC 802 **FGLINE** (instruction - high res. graphics) Page 84
Format **FGLINE** x,y[,colour number]
Function Draws a line from the previous position to the position indicated by the coordinates (x,y).
Note The origin is at the lower left-hand corner.

ABC 802 **FGPAINT** (instruction - high res. graphics) Page 84
Format **FGPAINT** x,y[,colour number]
Function Fills a closed area.
Note The origin is at the lower left-hand corner.

ABC 806
ABC 802 **FGPOINT** (instruction - high res. graphics) Page 85
Format **FGPOINT** x,y[, colour number]
Function Will turn on a pixel in position x,y. x=0-239, y=0-239.
Note The origin is at the lower left-hand corner.

FIX (function) Page 63
Format **FIX**(argument)
Function Gives the truncated value of the argument.
Note Compare with **INT**(X).

FLOAT (instruction) Page 40
Format **FLOAT**
Function All variables are interpreted as floating point.

FN (function) Page 74
Format **FN**identifier[%/$] [(parameter [,parameter,...])]
Function Calls a user-defined function.
Note Compare with **DEF FN.**

FNEND (instruction) Page 38
Format **FNEND**
Function Terminates a multiple line function.

FOR-TO-STEP (instruction) Page 41
Format **FOR** variable=expression **TO** expression [**STEP** interval]
Function Starts a program loop.

GET (instruction) Page 42
Format **GET** string variable
Function Reads one character from the keyboard.

GET £ (instruction) Page 43
Format **GET** £ file number, string variable
[**COUNT** number of characters]
Function Reads from a file.

GOSUB (instruction) Page 43
Format **GOSUB** line number
Function Unconditional jump to a subroutine.

GOTO (instruction) Page 44
Format **GOTO** line number
Function Unconditional jump to the given line number.

HEX$ (function) Page 64
Format **HEX**$(argument)
Function Converts a decimal number into a hexadecimal string.

IF-THEN-ELSE (instruction) Page 44
Format **IF** condition **THEN** statement(s)/line number
ELSE statement(s)/line number
Function Conditional control of the order of execution of the program lines.

INP (function) Page 74
Format **INP**(I%)
Function Gives the data value from the input port I%.
CAUTION This function is machine oriented, and should only be used for advanced programming.

INPUT (instruction) Page 45
Format **INPUT** [£file number/"prompt text"] variable [,variable, ...]
Function Fetches data for the current program.

INPUT LINE (instruction) Page 46
Format **INPUT LINE** [£file number,]string variable
Function Accepts a line of characters.

INSTR (function) Page 69
Format **INSTR**(N%,A$,B$)
Function Searches for the string B$ in A$ starting at position N%.

INT (function) Page 64
Format **INT**(argument)
Function The value of the greatest integer less than or equal to the argument. Compare with **FIX**.

INTEGER (instruction) Page 46
Format **INTEGER**
Function All variables are supposed to be integer variables, unless otherwise declared.

KILL (instruction) Page 47
Format **KILL** "[device:]file name.extension"
Function The file in question is erased from the external storage.

LEFT (function) Page 69
Format **LEFT**[$](A$,I%)
Function The first I% characters of the string A$.

LEN (function) Page 69
Format **LEN(A$)**
Function The number of characters of the string A$, i.e. the string length (including spaces).

LET (instruction) Page 47
Format [**LET**] variable=expression
Function Assigns a value to a variable.

LIST (command) Page 28
Format
LIST [device:]file name[.extension]
LIST [line number [-line number]]
LIST line number -
LIST - line number
LIST device: [,line number - line number]
Function Lists the whole program or part of it.

LOAD (command) Page 29
Format **LOAD** [device:]file name[.extension]
Function Loads a BASIC program into the computer.

LOG (function) Page 64
Format **LOG**(argument)
Function The natural logarithm of the argument.

LOG10 (function) Page 65
Format **LOG**10(argument)
Function The common logarithm of the argument.

MERGE (command) Page 29
Format **MERGE** [device:]file name[.extension]
Function Merges program files.

MID (instruction) Page 69
Format **MID**[$](A$,P%,K%)
Function Assigns new values to the characters no.P% to P%+K%-1 in A$, i.e. exchanges the characters indicated in the string.

MID (function) Page 70
Format **MID**[$](A$,P%,K%)
Function Gives the substring of A$, which starts in position P% and has a length of K% characters, i.e. the characters from no.P% to P%+K%-1.

MOD (function) Page 65
Format **MOD**(argument 1, argument 2)
Function The remainder of an integer division of the arguments.

MUL$ (function) Page 70
Format **MUL$**(A$,B$,[-]P%)
Function The product A$*B$ with P% decimals or digits (-P%).

NAME (instruction) Page 48
Format **NAME** "[device:]file name1.extension" **AS** "file name2.extension"
Function Changes the name of a file.

NEW (command) Page 30
Format **NEW**
Function Clears the storage.
Note The command **SCR** (scratch) can be used, as well.

NEXT (instruction) Page 42
Format **NEXT** variable
Function **NEXT** terminates a program loop, which begins with a **FOR** statement.

NO EXTEND (instruction) Page 49
Format **NO EXTEND**
Function Terminates work in **EXTEND** mode.

NO TRACE (instruction) Page 48
Format **NO TRACE**
Function Terminates the printout of line numbers, which was started by the instruction **TRACE.**

NUM$ (function) Page 70
Format **NUM$**(argument)
Function The numeric string corresponding to the argument.

OCT$ (function) Page 65
Format **OCT$**(argument)
Function Converts a decimal number into an octal string.

ON ERROR GOTO (instruction) Page 49
Format **ON ERROR GOTO** [line number]

ON - GOSUB (instruction) Page 49
Format **ON** expression **GOSUB** line number[,line number,...]
Function Conditional jump to one of several subroutines or to one of several entry points in a subroutine.

ON - GOTO (instruction) Page 50
Format **ON** expression **GOTO** line number[,line number,...]
Function Jump to one of several line numbers, depending on the value of the expression.

ON - RESTORE (instruction) Page 50
Format **ON** expression **RESTORE** line number[,line number,...]
Function Sets the **DATA** pointer by the same selection routine as **ON - GOTO**.

ON - RESUME (instruction) Page 51

Format **ON** expression **RESUME** line number[,line number,...]

Function Jump to one of several line numbers, depending on the value of the expression. Error handling is resumed. Used with **ON ERROR GOTO**.

OPEN (instruction) Page 51

Format **OPEN** "[device:][file name[.extension]]" **AS FILE** file number

Function Opens a file.

OPTION BASE (instruction) Page 52

Format **OPTION BASE** n

Function Denotes the lowest vector index value (n=0 or 1).

OUT (instruction) Page 75

Format **OUT** port,data [,port,data, ...]

Function Addresses the out ports at data output.

CAUTION This is a machine oriented instruction meant for advanced programming. This instruction and the instruction **INP** give the user access to the I/O-functions and the I/O-bus of the ABC 800.

PEEK (function) Page 75

Format **PEEK**(I%)

Function Gives the contents of storage address I%.

CAUTION This function is meant for advanced programming.

PEEK2 (function) Page 75

Format **PEEK2**(B%)

Function Reads the contents of two bytes.

CAUTION This function is meant for advanced programming.

PI (function) Page 65

Format **PI**

Function Constant with the value 3.14159 (single precision)

POKE (function) Page 75

Format **POKE** address,data [,data, ...]

Function Loads a value into a storage cell.

CAUTION This function is meant for advanced programming. If the instruction is used erroneously it may destroy the contents of the computer storage.

POSIT (instruction) Page 52

Format **POSIT** £file number, position

Function Positions the file pointer.

PREPARE (instruction) Page 53

Format **PREPARE** "[device:][file name.extension]" **AS FILE** file number

Function Creates and opens a new file.

PRINT (instruction) Page 53

Format **PRINT** [£file number,] "data"/variable [,"data"/variable, ...]

Function Prints data on an ASCII format.

PRINT (instruction - colour selection etc.) Page 81

Format **PRINT** [**CUR**(L,N)]argument [;argument;...]"text"

Function Used for printing text and graphics. The arguments control the colour selection etc. The starting point is at line L (0-23), position N (0-39/79).

Note The origin is at the upper, left-hand corner.

PRINT USING (instruction) Page 54

Format **PRINT** [£file number] **USING** "formatstring";"data"/variable [,"data"/variable, ...]

Function Prints numbers and strings in the format specified.

PUT (instruction) Page 57

Format **PUT** £file number, string va able

Function Writes a string variable.

RANDOMIZE (instruction) Page 57

Format **RANDOMIZE**

Function Sets a random starting value for the **RND** function (the random number generator).

CAUTION Should only be used once in a program.

READ (instruction) Page 57

Format **READ** variable[, variable, ...]

Function Used together with **DATA** statements as a way of assigning values to variables.

REM (instruction) Page 58

Format **REM** text

Function Inserts a comment in a program.

Note **REM** can be exchanged for !

RENUMBER (command) Page 30
REN

Format **REN[UMBER]** [line number[,interval[,from line -to line]]]

Function Changes the line numbering of the current program.

RESTORE (instruction) Page 58

Format **RESTORE** [line number]

Function Makes possible renewed use of the contents of DATA statements.

RESUME (instruction) Page 59

Format **RESUME** [line number]

Function Return from error handler.

RETURN (instruction) Page 59
Format **RETURN** [variable]
Function Return from subroutine or multiple line function.

RIGHT (function) Page 70
Format **RIGHT**[$](A$,N%)
Function The last characters of A$ starting at position N%.

RND (function) Page 65
Format **RND**
Function A random number between 0 and 0.9999999.

RUN (command) Page 31
Format **RUN** [device:]file name[.extension]
Function Loads and executes a BASIC program or executes the current program.

SAVE (command) Page 32
Format **SAVE** [device:]file name[.extension]
Function Creates a program file on an external storage and stores the current program on that file.

ABC 802, 806 **SCR** (command) Page 25
Format **SCR**
Function Clears the storage.
Note The command **NEW** can be used, as well. It works just like **SCR**.

ABC 806 **SET DOT** (instruction - graphics) Page 82
Format **SET DOT** L%,N%
Function Turns on a graphical point in position L% (0-71), N% (0-77).
Note The origin is in the upper left-hand corner.

SGN (function) Page 66
Format **SGN**(argument)
Function The function **SGN**(X) has the value +1 if X is positive, 0 if X is 0 and -1 if X is negative.

SIN (function) Page 66
Format **SIN**(argument)
Function The sine of the argument (the argument in radians).

SINGLE (instruction) Page 59
Format **SINGLE**
Function Changes all variables and expressions, which are floating point numbers, to single precision (7 digits).
Note **SINGLE** and **DOUBLE** cannot be mixed in the same program.

SPACE$ (function) Page 71
Format **SPACE**$(N%)
Function Yields a string consisting of N% spaces.

SQR (function) Page 66
Format **SQR**(argument)
Function The square root of the argument.

STOP (instruction) Page 60

Format **STOP**
Function Stops the program execution.
Note The program execution can be continued by one of the commands **CON** or **GOTO.**

STRING$ (function) Page 71

Format **STRING**$(I%,K%)
Function A string of I% ASCII characters.

SUB$ (function) Page 71

Format **SUB**$(A$,B$,[-]P%)
Function The arithmetic difference A$-B$ of the numeric strings A$ and B$ with P% decimals or digits (-P%).

SWAP% (function) Page 76

Format **SWAP**%(N%)
Function The first and the second bytes of N% change places.

SYS (function) Page 76 ***ABC 802, 806***

Format **SYS**(I%)
Function The system status as follows:
SYS(2) Total storage space
SYS(3) Program size
SYS(4) Remaining storage space
SYS(5) Keyboard flag
SYS(6) Puts back the last input character into the keyboard buffer.
SYS(11) Starting adress of the program
SYS(12) Variable root

TAB (function) Page 76

Format **TAB**(I%)
Function Tabulates to the I%-th position on the line.

TAN (function) Page 66

Format **TAN**(argument)
Function The tangent of the argument (the argument in radians).

TIME$ (function) Page 76

Format **TIME**$
Function Returns year-month-day hour.min.sec

TRACE (instruction) Page 60

Format **TRACE** [£file number]
Function Prints the line numbers of the executed program lines.

TXPOINT (instruction - graphics) Page 81 ***ABC 806***

Format **TXPOINT** x,y(,1/0)
Function Turns on (1) or off (0) a graphical point in position x, y, where x=0-77 and y=0-71.
Note The origin is in the lower left-hand corner.

UNSAVE (command) Page 32
Format UNSAVE [device:]file name[.extension]
Function Erases a file from a disk.

VAL (function) Page 71
Format **VAL**(A$)
Function Calculates the numeric value of the numeric string A$.

VAROOT (function) Page 77
Format **VAROOT** (variable)
Function Gives the address of a table which contains information about a variable.
CAUTION This function is meant for advanced programming.

VARPTR (function) Page 77
Format **VARPTR** (variable)
Function Gives the address of the value of a variable.
CAUTION This function is meant for advanced programming.

WEND (instruction) Page 60
Format **WEND**
Function **WEND** terminates a loop that begins with **WHILE.**

WHILE (instruction) Page 61
Format **WHILE** expression
Function Specifies the condition for branching out of a program loop.

ABC 802, 806 **A$+B$** (function) Page 72
Format A$+B$
Function Concatenates two strings.

17 Literature References

- "The ABC of Microcomputers" by Gunnar Markesjö.
  Explains how the ABC 800 works.

- "ABC's of BASIC" by Anders Andersson, Arne Kullbjer, Jan Lundgren and Sören Thornell.
  An introduction to ABC 80 BASIC.

- "Controlling and Measuring with the ABC 80" by Åke Westh.

- "ABC's of Programming and Documentation" by Jan Lundgren and Bengt Lundin.

- "Programming the ABC 80" by Lennart Råde.

- "Instruction Manual for PASCAL" by Anders Haraldsson.

- "BASIC Computer Games" by David H. Ahl.

- "More BASIC Computer Games" by David H. Ahl.

- "Z80 Technical Manual" published by Zilog.

- "Z80 Programming Manual" published by Zilog.

- "Data Processing Glossary". Swedish Standard SIS Handbook 142.

Books about the ABC 800, BASIC II and programming will be issued from time to time during 1981.

18 Appendices

ABC 802, 806

Appendix 1: BASIC II Errata

If variables are dimensioned (**DIM**) or assigned values before a **COMMON** declaration, ABC 800 will "get lost".

ABC 806

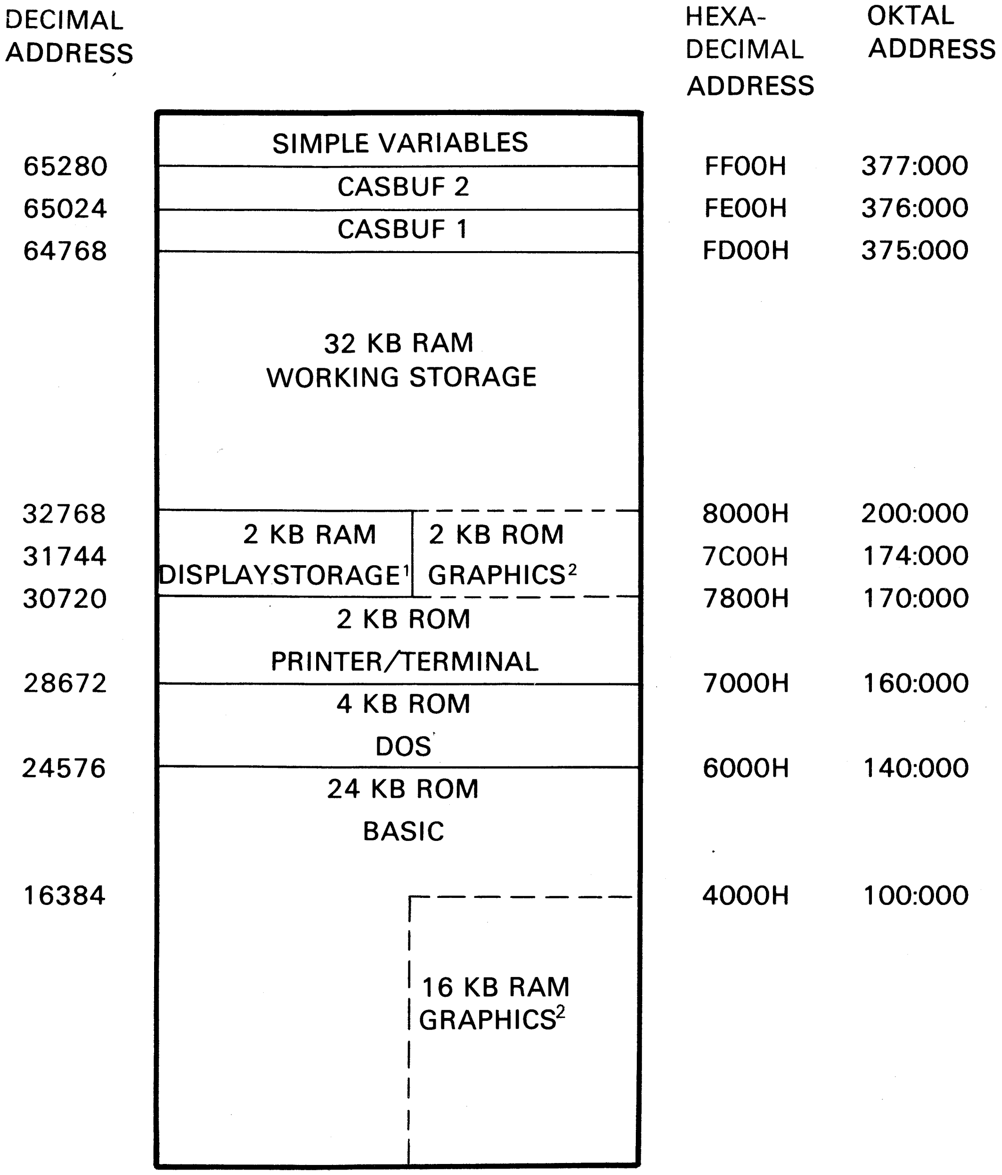
Appendix 2: The I/O Ports of ABC 800

Port	Address bit 7 bit 0		
CTC	011XXX00	channel 0	
	011XXX01	channel 1	
	011XXX10	channel 2	
	011XXX11	channel 3	
SIO/2	010XXX00	V24 data	channel B
	010XXX01	V24 control	channel B
	010XXX10	cassette data	
	010XXX11	cassette control	
DART	0010XX00	printer data	channel A
	0010XX01	printer control	channel A
	0010XX10	keyboard data	
	0010XX11	keyboard control	

X = don't care

Appendix 3: Storage Disposition

ABC 800 M/C HR Memory Map without Disk Drives.

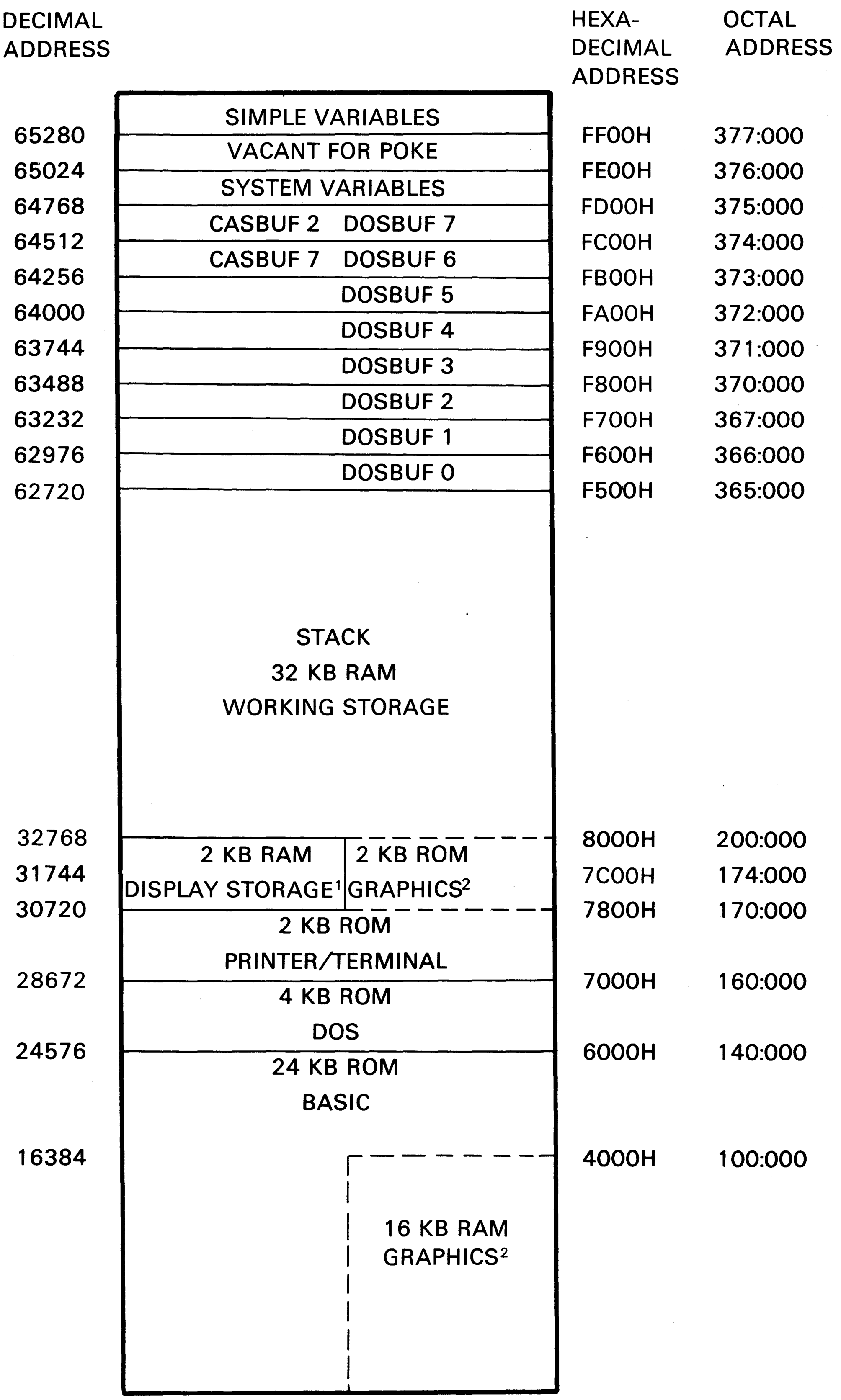


1. ABC 800 C uses only 1 kB CRT text display storage (31744 -32786).

2. The CRT text display storage (2 kB) on the VU board is parallel with the graphics system program (2 kB) on the PU board. Likewise, the CRT graphics display storage (16 kB) is parallel with the system program for BASIC. The different areas of the memory do not interact. ABC 800 runs in a special mode when the graphics storage is addressed. If storage space for machine language routines is to be allocated, the following addresses are changed:

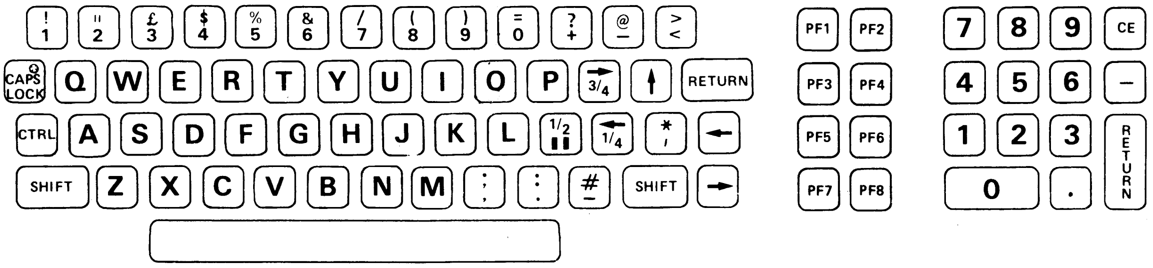
- ● The pointer for the lowest memory address of a BASIC program (BOTTOM): 65292
- ● The pointer for the highest memory address of a BASIC program (TOP): 65294

ABC 800 Memory Map with Disk Drives.

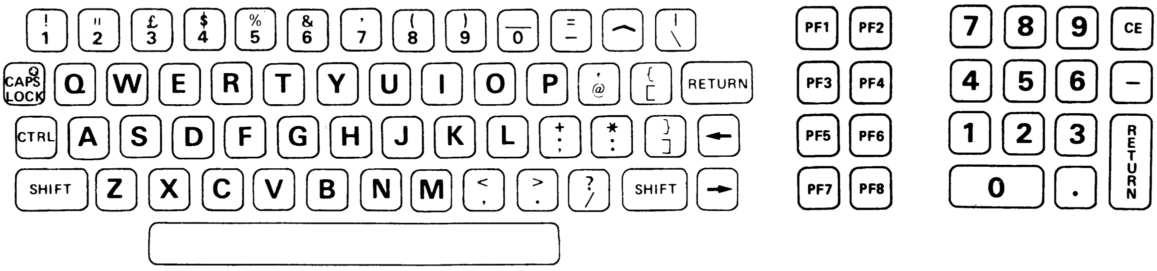


DECIMAL ADDRESS			HEXA-DECIMAL ADDRESS	OCTAL ADDRESS
	SIMPLE VARIABLES			
65280			FF00H	377:000
	VACANT FOR POKE			
65024			FE00H	376:000
	SYSTEM VARIABLES			
64768			FD00H	375:000
	CASBUF 2	DOSBUF 7		
64512			FC00H	374:000
	CASBUF 7	DOSBUF 6		
64256			FB00H	373:000
		DOSBUF 5		
64000			FA00H	372:000
		DOSBUF 4		
63744			F900H	371:000
		DOSBUF 3		
63488			F800H	370:000
		DOSBUF 2		
63232			F700H	367:000
		DOSBUF 1		
62976			F600H	366:000
		DOSBUF 0		
62720			F500H	365:000
	STACK 32 KB RAM WORKING STORAGE			
32768			8000H	200:000
	2 KB RAM	2 KB ROM		
31744			7C00H	174:000
	DISPLAY STORAGE[1]	GRAPHICS[2]		
30720			7800H	170:000
	2 KB ROM PRINTER/TERMINAL			
28672			7000H	160:000
	4 KB ROM DOS			
24576			6000H	140:000
	24 KB ROM BASIC			
16384			4000H	100:000
		16 KB RAM GRAPHICS[2]		

Appendix 4: Keyboard Layout, ASCII Codes


! 1 " 2 £ 3 $ 4 % 5 & 6 / 7 (8) 9 = 0 ? + @ – > <
CAPS LOCK Q W E R T Y U I O P 3/4 ↑ RETURN
CTRL A S D F G H J K L 1/2 1/4 * ←
SHIFT Z X C V B N M ; : # SHIFT →
PF1 PF2 PF3 PF4 PF5 PF6 PF7 PF8
7 8 9 CE 4 5 6 – 1 2 3 RETURN 0 .

Viewdata keyboard


! 1 " 2 £ 3 $ 4 % 5 & 6 ' 7 (8) 9 0 = – \
CAPS LOCK Q W E R T Y U I O P @ [RETURN
CTRL A S D F G H J K L + ; * :] ←
SHIFT Z X C V B N M < , > . ? / SHIFT →
PF1 PF2 PF3 PF4 PF5 PF6 PF7 PF8
7 8 9 CE 4 5 6 – 1 2 3 RETURN 0 .

Typewriter keyboard

Codes obtained from the keyboard

			Key			
ASCII code	CTRL	Shift	ABC 800 C	ABC 800 M	ASCII name	Function
0	X		@	@	NUL	Time filler character
1	X		A	A	SOH	–
2	X		B	B	STX	–
3	X		C	C	ETX	Stops execution
4	X		D	D	EOT	–
5	X		E	E	ENQ	–
6	X		F	F	ACK	–
7	X		G	G	BEL	"Beep" issued by loudspeaker
8	X		H	H	BS	*) "←" key
9	X		I	I	HT	*) "→" key
10	X		J	J	LF	Line feed
11	X		K	K	VT	–
12	X		L	L	FF	*) Erases screen
13	X		M	M	CR	*) "RETURN" key
14	X		N	N	SO	–
15	X		O	O	SI	–
16	X		P	P	DLE	–
17	X		Q	Q	DC1	–
18	X		R	R	DC2	–
19	X		S	S	DC3	Steps one program instruction
20	X		T	T	DC4	–
21	X		U	U	NAK	–
22	X		V	V	SYN	–
23	X		W	W	ETB	–
24	X		X	X	CAN	*) Deletes entered line
25	X		Y	Y	EM	–
26	X		Z	Z	SUB	–
27	X		←	[	ESC	–
28	X		1/2	\	FS	–
29	X		→	]	GS	–
30	X		↑	↑	RS	–
31	X	X	O	O	US	–
127	X		<	<	DEL	–

*) These characters affect the screen directly.

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

A	C	G	A	C	G	A	C	G	A	C	G
32	Space	□	56	8	🬖	80	P	P	104	h	🬦
33	!	🬀	57	9	🬗	81	Q	Q	105	i	🬧
34	"	🬁	58	:	🬘	82	R	R	106	j	▐
35	£	🬂	59	;	🬙	83	S	S	107	k	🬨
36	$	🬃	60	<	🬚	84	T	T	108	l	🬩
37	%	🬄	61	=	🬛	85	U	U	109	m	🬪
38	&	🬅	62	>	🬜	86	V	V	110	n	🬫
39	'	🬆	63	?	🬝	87	W	W	111	o	🬬
40	(	🬇	64	@	@	88	X	X	112	p	🬭
41	)	🬈	65	A	A	89	Y	Y	113	q	🬮
42	•	🬉	66	B	B	90	Z	Z	114	r	🬯
43	+	🬊	67	C	C	91	←	←	115	s	🬰
44	,	🬋	68	D	D	92	1/2	1/2	116	t	🬱
45	-	🬌	69	E	E	93	→	→	117	u	🬲
46	.	🬍	70	F	F	94	↑	↑	118	v	🬳
47	/	🬎	71	G	G	95	#	#	119	w	🬴
48	0	🬏	72	H	H	96	_	🬞	120	x	🬵
49	1	🬐	73	I	I	97	a	🬟	121	y	🬶
50	2	🬑	74	J	J	98	b	🬠	122	z	🬷
51	3	🬒	75	K	K	99	c	🬡	123	1/4	🬸
52	4	🬓	76	L	L	100	d	🬢	124	||	🬹
53	5	▌	77	M	M	101	e	🬣	125	3/4	🬺
54	6	🬔	78	N	N	102	f	🬤	126	—*	🬻
55	7	🬕	79	O	O	103	g	🬥	127	■	█

ASCII codes (A) for character mode (C) and graphic mode (G).

Decimal codes obtained from function keys.

		SHIFT	CTRL	SHIFT + CTRL
PF1	192	208	224	240
PF2	193	209	225	241
PF3	194	210	226	242
PF4	195	211	227	243
PF5	196	212	228	244
PF6	197	213	229	245
PF7	198	214	230	246
PF8	199	215	231	247

The following commands are used as control functions and are typed at the keyboard:

RETURN	the DO IT command
CTRL/C	stops program execution (NOTE! CTRL/C terminates execution twice)
CTRL/H or ←	erases one character
CTRL/I or →	used for editing
CTRL/L	clears the screen
CTRL/S	single step execution
CTRL/X or CE	erases the last entered line

Appendix 5: Differences between ABC 800 and ABC 802

This appendix contains the differencies in the BASIC program for ABC 802 as compared with the program for ABC 800. The paragraphs affected are indicated with ABC 802 in the text margin.

6.4 Guide to the Statements

(Page 21)

The paragraph "To define and ... **INPUTLINE.**" is omitted.

Miscellaneous statements:

COMMON and **DIM** sets the size of variables.
STOP, TRACE, and **NOTRACE** facilitate the debugging of a program.
WIDTH chooses the number of characters per line (40 or 80).

8 Commands

(Page 24)

- the devices are addressed as DR0:, DR1:, CAS:, PR:, CON:, or MEM:.
- the primary default device is disk drive 0 (DR0:) and the secondary one is disk drive 1 (DR1:). If both a disk drive and a cassette recorder are connected, the device CAS: must be given if a command is to act on the cassette recorder. Correspondingly, the device MEM: must be given if a command is to act on the external memory.

LIST (page 28)

Note

A long program is listed on the screen until it is filled. The next line will be displayed when you press the space bar. A listing can be stopped by CTRL/C, RETURN or any BASIC command.

When the device MEM: is given, file name.extension shall be replaced by a number (No.). This number is for identifying the file and for calculating the address on which the file is stored. The address is to be calculated in the following way:

Address = number × 253

LOAD (page 29)

Note

If no extension is given, the computer will first search for .BAC and then .BAS. The entire file is read, until EOF (end of file) and not only to the **END**.

When the device MEM: is given, file name.extension shall be replaced by a number (No.). This number is for identifying the file and for calculating the address on which the file is stored. The address is to be calculated in the following way:

Address = number × 253

RUN (page 31)

Examples

Example 1

```
10 READ A,B
20 LET A=A+B
30 PRINT A
40 DATA 2,3
50 END
RUN
5
```

Example 2
If the same program had been stored on an external storage under the name AADDB, the screen would look like this:

```
RUN AADDB
5
```

When the device MEM: is given, file name.extension shall be replaced by a number (No.). This number is for identifying the file and for calculating the address on which the file is stored. The address is to be calculated in the following way:

Address = number × 253

SAVE (page 32)

Note

If the file exists already on the disk, the old file will be destroyed and replaced by the new program, unless the file or the disk is write protected.

When the device MEM: is given, file name.extension shall be replaced by a number (No.). This number is for identifying the file and for calculating the address on which the file is stored. The address is to be calculated in the following way:

Address = number × 253

UNSAVE (page 32)

Note

When the extension is omitted, the computer will look for .BAC first and then .BAC.
The command **UNSAVE** cannot be used on an erase protected file or on MEM:.

WIDTH (page 34)

Selects a number of characters per line

CHAIN (page 35)

Note

Variables can be passed on to a **CHAIN**ed program by means of the **COMMON** instruction.

When the device MEM: is given, file name.extension shall be replaced by a number (No.). This number is for identifying the file and for calculating the address on which the file is stored. The address is to be calculated in the following way:

Address = number × 253

KILL (page 47)

Example

When the file XYZ.TXT on the disk is no longer needed, the file can be erased from the disk by means of the following statement:

```
460 KILL "XYZ.TXT"
```

Note that the instruction KILL does not function together with MEM:.

OPEN (page 51)

Format

OPEN "[device:][file name[.extension]]" **AS FILE** file number

Where	<device> may be for instance
DR0:	Disk drive 0
DR1:	Disk drive 1
PR:	Printer
CAS:	Cassette recorder
CON:	Keyboard and screen
MEM:	32 Kbyte internal RAM (RAM-floppy)

The expression following **AS FILE** should be an integer value between 0 and 255.

Note

When data is to be read from an existing file, the file should be opened by the **OPEN** instruction. Up to seven files may be open at the same time.

When the device MEM: is given, file name.extension shall be replaced by a number (No.). This number is for identifying the file and for calculating the address on which the file is stored. The address is to be calculated in the following way:

Address = number × 253

PREPARE (page 53)

Example

```
10 PREPARE "DATA.TXT" AS FILE 2
20 PRINT #2,A
30 PRINT #2,B
40 PRINT #2,C$
```

The values of the variables A, B, and C$ are written on file 2 (DATA.TXT).

When the device MEM: is given, file name.extension shall be replaced by a number (No.). This number is for identifying the file and for calculating the address on which the file is stored. The address is to be calculated in the following way:

Address = number × 253

WIDTH (page 61)

Format

WIDTH [#file number,] number of characters

Function

Denotes number of characters per line for the current file. When # file number is omitted the screen is intended (40 alt. 80 characters).

Use

Is used for conversion of line length.

Example

```
10 WIDTH 40
```

Converts ABC 802 to 40-character mode.

```
10 WIDTH # 1.64
```

Converts file 1 to 64 characters per line.

10.3 Other Functions

SYS (page 76)

Function

System information as follows:

SYS (2)	Total storage space
SYS (3)	Program size
SYS (4)	Remaining storage space
SYS (5)	Keyboard flag. Can be cleared by means of **GET, INPUT,** or **INPUT LINE.**
SYS (6)	Puts back the last input character into the keyboard buffer
SYS (8)	Is −1 when a key is pressed
SYS(11)	Starting adress of the program
SYS(12)	Variable root

10.4 Inverted video (Page 77)

Inverted video can be displayed. This is done by inserting (1) bit 8 (128) into the ASCII-value of the character in question. The following programming example can be used for presentation of inverted video.

```
  10 !******************************************
  20 !*         Test example of inverted video         *
  30 !******************************************
  40 EXTEND
  50 PRINT ”Ordinary text”
  60 PRINT FNInv$(”Inverted text”)
  70 PRINT ”Mixed”;FNInv$(”text:”);”usual/”;
     FNInv$(”inverted”)
1000 !******************************************
1010 !*              function for inverted text             *
1020 !*       input parameter=text for inv. printing      *
1030 !******************************************
1040 DEF FNInv$(Text$) LOCAL Text$=160
1050 FOR I=1 TO LEN (Text$)
1060 PUT (CHR$(ASCII(MID$ (Text$,I,1)) OR 128))
1070 NEXT I
1080 RETURN ””
1090 FNEND
```

11 Graphics and Colours (Page 78)

11.1 General Information

ABC 802 graphics correspond to the Teletext standards. In the graphic mode every output character is interpreted as a graphic character formed by a combination of six graphic points.

When text or graphics are displayed on the screen, the selection of colours etc. is controlled by means of certain arguments in the **PRINT** statement. The statement affects one line at a time. Each argument puts a control character on the screen. Although these characters are invisible, they take up one position each. The control characters can be covered by a background colour, if the control arguments are given in the correct order.

N.B.
Programs for colours can be written on ABC 802 for later execution on ABC 800 C and ABC 806. ABC 802 cannot be used for presentation of colours.

The following colours are available:

Red **(RED)**
Green **(GRN)**
Yellow **(YEL)**
Blue **BLU)**
Magenta **(MAG)**
Cyanide **(CYA)**
White **(WHT)**

The characters available in the ABC 802 are listed below. The table gives the ASCII value of each character and its meaning in the character mode and graphic mode. One way of planning a graphical picture is to draw it on a copy of the graphics chart and feed the program the appropriate data.

When you have finished the picture on a copy of the chart you can type the lines one by one. Do not forget to allow space for the control characters, if you vary the control arguments.

Note that the capital letters still remain the same in graphic mode. You can mix capital letters and graphic characters just as you like.

In graphic mode there are 72 graphic lines (0-71), each one with 78/158 graphic positions (0-77/157).

The number of graphical positions/line is dependent upon which character mode has been selected, 40 or 80 characters.

A	C	G	A	C	G	A	C	G	A	C	G
32	Blank		56	8	🬖	80	P	P	104	h	🬦
33	!	🬀	57	9	🬗	81	Q	Q	105	i	🬧
34	"	🬁	58	:	🬘	82	R	R	106	j	▐
35	#	🬂	59	;	🬙	83	S	S	107	k	🬨
36	$	🬃	60	<	🬚	84	T	T	108	l	🬩
37	%	🬄	61	=	🬛	85	U	U	109	m	🬪
38	&	🬅	62	>	🬜	86	V	V	110	n	🬫
39	'	🬆	63	?	🬝	87	W	W	111	o	🬬
40	(	🬇	64	@	@	88	X	X	112	p	🬭
41	)	🬈	65	A	A	89	Y	Y	113	q	🬮
42	*	🬉	66	B	B	90	Z	Z	114	r	🬯
43	+	🬊	67	C	C	91	[	[	115	s	🬰
44	,	🬋	68	D	D	92	\	\	116	t	🬱
45	-	🬌	69	E	E	93	]	]	117	u	🬲
46	.	🬍	70	F	F	94	↑	↑	118	v	🬳
47	/	🬎	71	G	G	95	-	-	119	w	🬴
48	0	🬏	72	H	H	96	´	🬞	120	x	🬵
49	1	🬐	73	I	I	97	a	🬟	121	y	🬶
50	2	🬑	74	J	J	98	b	🬠	122	z	🬷
51	3	🬒	75	K	K	99	c	🬡	123	{	🬸
52	4	🬓	76	L	L	100	d	🬢	124	,	🬹
53	5	▌	77	M	M	101	e	🬣	125	}	🬺
54	6	🬔	78	N	N	102	f	🬤	126	—*	🬻
55	7	🬕	79	O	O	103	g	🬥	127	■	█

ASCII code interpreted into character mode (C) and graphic mode (G).

Argument	CHR$( )	Argument	CHR$( )
RED	129	**GYEL**	147
GRN	130	**GBLU**	148
YEL	131	**GMAG**	149
BLU	132	**GCYA**	150
MAG	133	**GWHT**	151
CYA	134	**HIDE**	152
WHT	135	**GCON**	153
FLSH	136	**GSEP**	154
STDY	137	**BLBG**	156
NRML	140	**NWBG**	157
DBLE	141	**GHOL**	158
GRED	145	**GREL**	159
GGRN	146		

ABC800®   VIDEO GRAPHICS CHART   PROGRAM............   ....

0	71 70 69	▨																																							
1	68 67 66	▨																																							
2	65 64 63	▨																																							
3	62 61 60	▨																																							
4	59 58 57	▨																																							
5	56 55 54	▨																																							
6	53 52 51	▨																																							
7	50 49 48	▨																																							
8	47 46 45	▨																																							
9	44 43 42	▨																																							
10	41 40 39	▨																																							
11	38 37 36	▨																																							
12	35 34 33	▨																																							
13	32 31 30	▨																																							
14	29 28 27	▨																																							
15	26 25 24	▨																																							
16	23 22 21	▨																																							
17	20 19 18	▨																																							
18	17 16 15	▨																																							
19	14 13 12	▨																																							
20	11 10 9	▨																																							
21	8 7 6	▨																																							
22	5 4 3	▨																																							
23	2 1 0	▨																																							
	DOT	▨	0 1	2 3	4 5	6 7	8 9	10 11	12 13	14 15	16 17	18 19	20 21	22 23	24 25	26 27	28 29	30 31	32 33	34 35	36 37	38 39	40 41	42 43	44 45	46 47	48 49	50 51	52 53	54 55	56 57	58 59	60 61	62 63	64 65	66 67	68 69	70 71	72 73	74 75	76 77
CHR		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39

▨ - reserved for graphic control characters

11.2 Instructions (page 81)

PRINT

Format **PRINT** [**CUR**(L,N)]argument [;argument;...]"text"

Function Used for printing text and graphics. The arguments control the colour selection etc. A G at the beginning of the colour selection argument (e.g. **GRED**) sets the line to the graphic mode so that all characters within quotes are interpreted as being graphics (see the ASCII table). If **CUR** (L,N) is specified, the picture is drawn from the starting point at line L (0-23), position N (0-39).

The following arguments are available:

RED, GRN, YEL, BLU, MAG, CYA, WHT	Alphanumeric colour characters
GRED, GGRN, GYEL, GBLU, GMAG, GCYA, GWTH	Colour graphics
FLSH, STDY	Flashing, steady
NRML, DBLE	Normal, double height N.B. DBLE cannot be generated on ABC 802
GCON, SEP	Continuous, separated graphics N.B. Cannot be generated on ABC 802
NWBG, BLBG	New background, black background
GHOL, GREL	Hold, release graphics N.B. Cannot be generated on ABC 802
HIDE	Concealed text/graphics

The arguments can also be given with **CHR$**.
The control arguments should be given in the following order:

PRINT <background colour argument> <argument for new background colour> <text colour argument> "Text" <argument for black background>

Example

```
10 PRINT GYEL "1,6 BOW WOW"
```

The programs can thus be written for colour, for execution on, for example, ABC 800 C.

TXPOINT (page 81)

Format

TXPOINT X,Y [,1/0]
where X=0-77/157 and Y=0-71.

The upper X limit varies according to 40/80-character mode.

Function

Turns on (1 can be omitted) or turns off (0) a graphic point in position X, Y.

Example

```
10 PRINT CHR$(12)
20 FOR I=0 TO 23
30 PRINT CUR(1,0) GGRN;
40 NEXT I
50 FOR I=2 TO 157
60 TXPOINT 1,32+SIN(I/5)*15
70 NEXT I
80 PRINT CUR(3,32) RED FLSH "SINE"
90 END
```

The lines 10–40 clear the screen and set it to the graphic mode (green). The lines 50–70 draw a sine curve. Line 80 displays SINE in red, flashing text. **TXPOINT** can also be used as a function, to check if a point is turned on (−1) or off (0). **TXPOINT**(X,Y).

Note

The origin is in the lower, left-hand corner.

SET DOT (page 82)

Format

SET DOT R%,K%

where R%=0-71 and K%=2-79/159, according to 40- or 80-character mode.

Function

Turns on a graphic point (the origin is in the upper left-hand corner).

CLR DOT (page 82)

Format

CLR DOT R%,K%

where R%=0-71 and K%=2-79/159 (40 alt 80 characters).

Function

Turns off a graphic point (the origin is in the upper left-hand corner).

DOT (page 82)

Format

DOT(R%,K%)

where R%=0-71 and K%=2-79/159 (40 alt 80 characters).

Function

Will be −1 (true) if the point is lit, else 0 (false).

12 High Resolution Graphics

(page 83)

This section is not valid for ABC 802.

13 Function Keys

(page 90)

ABC 802 is provided with special function codes. These are generated by **CTRL, SHIFT** and certain alphanumeric keys, see table 1.

These function codes correspond completely with the codes which are generated by ABC 800 function keys (PF1–PF8).

A programmer can assign various functions to the function keys, e.g. cursor movements, jump to a program module etc. For assigning the same functions to ABC 802 as those included in ABC 800, the following combinations are used.

The function keys can produce 32 different ASCII values as shown in the following table:

Table 1

	UNSHIFT	SHIFT	CTRL	SHIFT+CTRL
PF1	CTRL/1	SHIFT/CTRL 1	CTRL/SHIFT C	CTRL/SHIFT A
PF2	CTRL/2	SHIFT/CTRL 2	CTRL/SHIFT W	CTRL/SHIFT S
PF3	CTRL/3	SHIFT/CTRL 3	CTRL/SHIFT E	CTRL/SHIFT D
PF4	CTRL/4	SHIFT/CTRL 4	CTRL/SHIFT R	CTRL/SHIFT F
PF5	CTRL/5	SHIFT/CTRL 5	CTRL/SHIFT T	CTRL/SHIFT G
PF6	CTRL/6	SHIFT/CTRL 6	CTRL/SHIFT Y	CTRL/SHIFT H
PF7	CTRL/7	SHIFT/CTRL 7	CTRL/SHIFT U	CTRL/SHIFT J
PF8	CTRL/8	SHIFT/CTRL 8	CTRL/SHIFT I	CTRL/SHIFT K

Table 2

	UNSHIFT	SHIFT	CTRL	SHIFT+CTRL
PF1	192	208	224	240
PF2	193	209	225	241
PF3	194	210	226	242
PF4	195	211	227	243
PF5	196	212	228	244
PF6	197	213	229	245
PF7	198	214	230	246
PF8	199	215	231	247

When a function key is pressed, a subroutine can be called as shown below:

Example

```
 10 ON ERROR GOTO 100
 20 INPUT "Number",P(I)
 30 I=I+1
 40 GOTO 20
  I
  I
100 IF ERRCODE < > 53 THEN RESUME
110 A=SYS(6)
120 GET X$
130 ON ASC(X$)−191 RESUME 400, 500, 600
```

When a function key is pressed at **INPUT** or **INPUT LINE**, an error is generated. The **ERRCODE** is 53. The program should contain a routine which handles error 53. To find out which one of the function keys that was pressed, use the function **SYS**(6) and read the character by means of **GET**.

14 Differences in BASIC between ABC 800 and ABC 80 (Page 91)

6. The instruction **CHAIN""** is changed to **CHAIN"NUL:"**.

16 Summary of Commands and Instructions (Pages 97, 98, and 104)

FGCTL, FGFILL, FGLINE, FGPAINT, FGPOINT and **SCR** are omitted.

SYS (page 105) (function) Page 76

Format **SYS**(I%)

Function The system status as follows:

SYS(2)	Total storage space
SYS(3)	Program size
SYS(4)	Remaining storage space
SYS(5)	Keyboard flag
SYS(6)	Puts back the last input character into the keyboard buffer
SYS(8)	Is −1 when a key is pressed
SYS(11)	Starting adress of the program
SYS(12)	Variable root

WIDTH (page 106) (instruction) Page 61

Format **WIDTH** [#file number] number

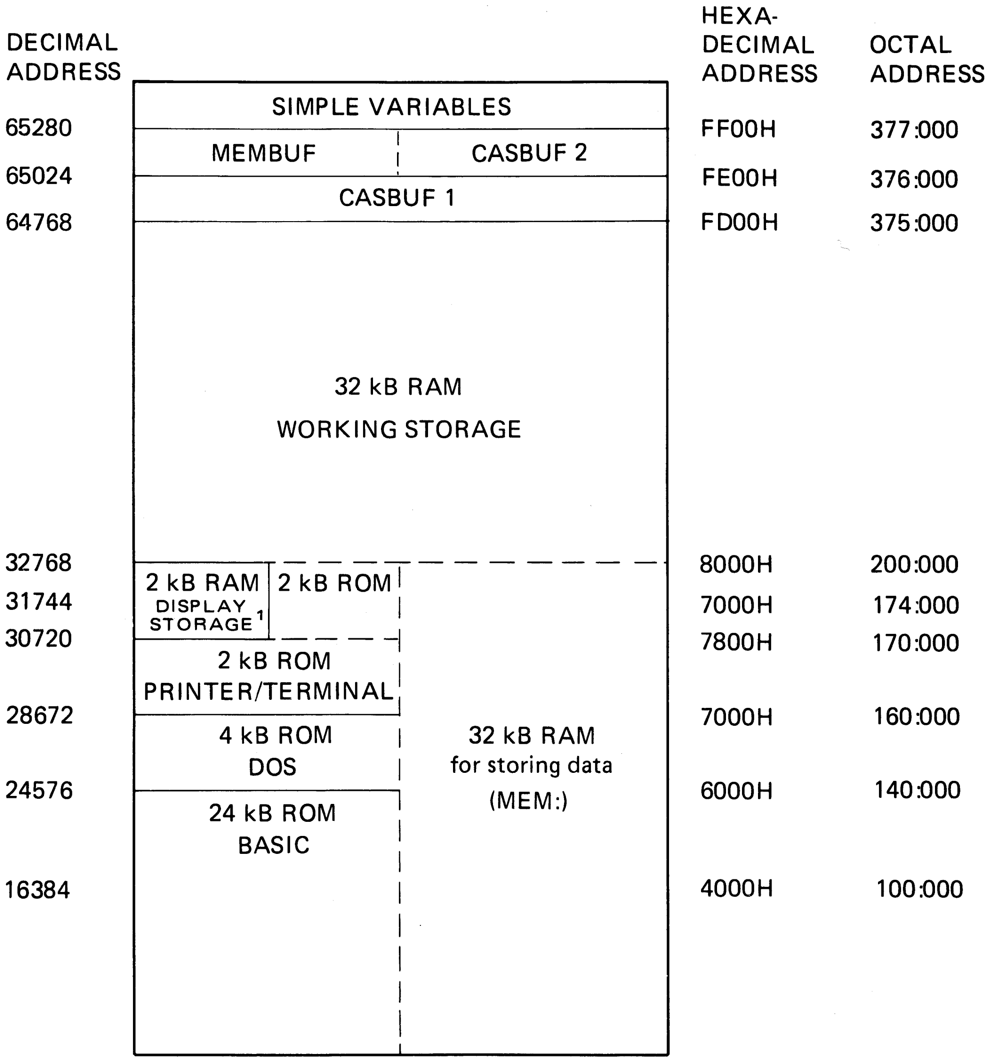
Function Denotes number of characters per line.

Appendix 1: BASIC II Errata (page 108)

Appendix 1 is omitted.

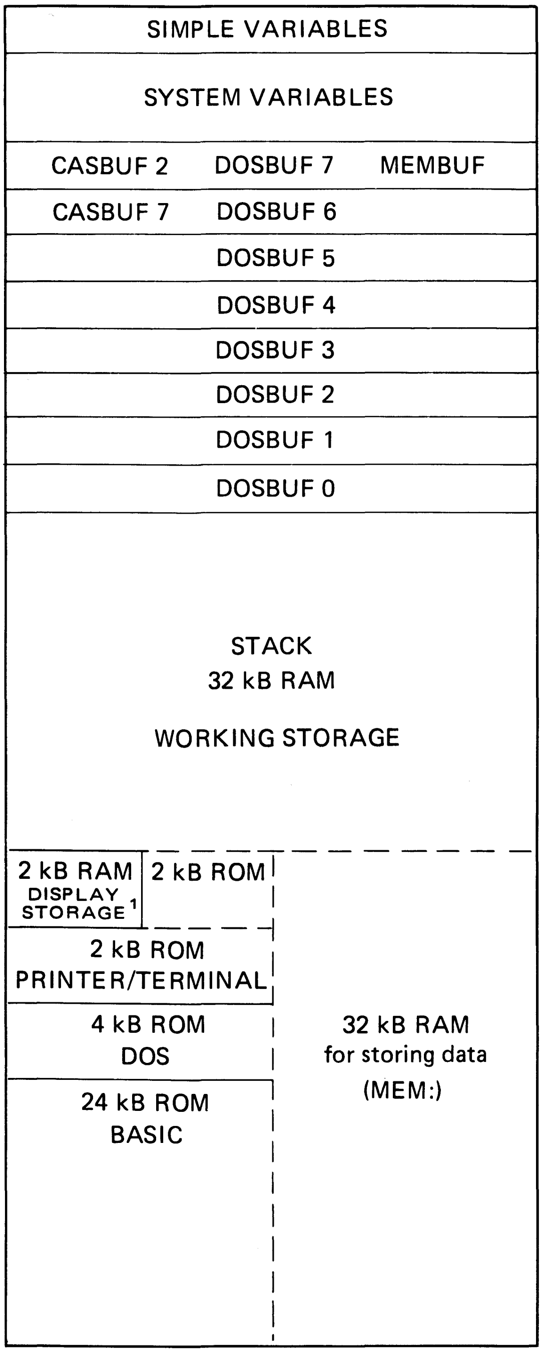
Appendix 3: Storage Disposition (page 109)

ABC 802 without Disk Drives



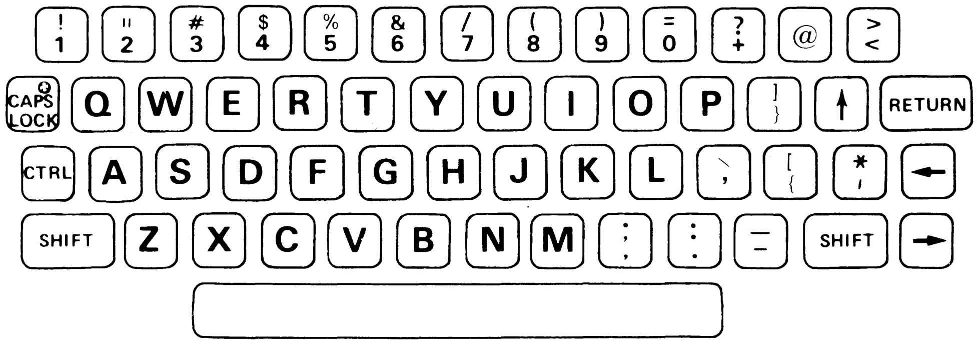
1. The display storage (2 kB) is parallel with the system program in PROM. Likewise is the MEM: storage (32 kB) parallel with the system program for BASIC. The different areas of the memory do not interact. ABC 802 runs in a special mode when the graphics storage is addressed. If storage space for machine language routines is to be allocated, the following addresses are changed:
   - The pointer for the lowest memory address of a BASIC program (BOTTOM): 65292
   - The pointer for the highest memory address of a BASIC program (TOP): 65294.

ABC 802 Memory Map with Disk Drives



DECIMAL ADDRESS			HEXA-DECIMAL ADDRESS	OCTAL ADDRESS
	SIMPLE VARIABLES			
65280			FF00H	377:000
	SYSTEM VARIABLES			
64768			FD00H	375:000
	CASBUF 2 DOSBUF 7 MEMBUF			
64512			FC00H	374:000
	CASBUF 7 DOSBUF 6			
64256			FB00H	373:000
	DOSBUF 5			
64000			FA00H	372:000
	DOSBUF 4			
63744			F900H	371:000
	DOSBUF 3			
63488			F800H	370:000
	DOSBUF 2			
63232			F700H	367:000
	DOSBUF 1			
62976			F600H	366:000
	DOSBUF 0			
62720			F500H	365:000
	STACK 32 kB RAM WORKING STORAGE			
32768			8000H	200:000
	2 kB RAM DISPLAY STORAGE[1] | 2 kB ROM	32 kB RAM for storing data (MEM:)		
31744			7000H	174:000
30720			7800H	170:000
	2 kB ROM PRINTER/TERMINAL			
28672			7000H	160:000
	4 kB ROM DOS			
24576			6000H	140:000
	24 kB ROM BASIC			
16384			4000H	100:000

Appendix 4: Keyboard Layout, ASCII Codes (page 111)



Codes obtained from the keyboard

ASCII code	CTRL	SHIFT	Key	ASCII name	Function
0	X		@	NUL	Time filler character
1	X		A	SOH	–
2	X		B	STX	–
3	X		C	ETX	Stops execution
4	X		D	EOT	–
5	X		E	ENQ	–
6	X		F	ACK	–
7	X		G	BEL	–
8	X		H	BS	*) "←" key
9	X		I	HT	*) "→" key
10	X		J	LF	Line feed
11	X		K	VT	–
12	X		L	FF	*) Erases screen
13	X		M	CR	*) "RETURN" key
14	X		N	SO	–
15	X		O	SI	–
16	X		P	DLE	–
17	X		Q	DC1	–
18	X		R	DC2	–
19	X		S	DC3	Steps one program instruction
20	X		T	DC4	–
21	X		U	NAK	–
22	X		V	SYN	–
23	X		W	ETB	–
24	X		X	CAN	*) Deletes entered line
25	X		Y	EM	–
26	X		Z	SUB	–
27	X		[	ESC	–
28	X		\	FS	–
29	X		]	GS	–
30	X		↑	RS	–
31	X	X	O	US	–
127	X		<	DEL	Generates a filled square (■)

*) These characters affect the screen directly.

Argument	CHR$( )
RED	129
GRN	130
YEL	131
BLU	132
MAG	133
CYA	134
WHT	135
FLSH	136
STDY	137
NRML	140
DBLE	141
GRED	145
GGRN	146
GYEL	147
GBLU	148
GMAG	149
GCYA	150
GWHT	151
HIDE	152
GCON	153
GSEP	154
BLBG	156
NWBG	157
GHOL	158
GREL	159

A	C	G	A	C	G	A	C	G	A	C	G
32	Blank	□	56	8	🬖	80	P	P	104	h	🬦
33	!	🬀	57	9	🬗	81	Q	Q	105	i	🬧
34	"	🬁	58	:	🬘	82	R	R	106	j	▐
35	#	🬂	59	;	🬙	83	S	S	107	k	🬨
36	$	🬃	60	<	🬚	84	T	T	108	l	🬩
37	%	🬄	61	=	🬛	85	U	U	109	m	🬪
38	&	🬅	62	>	🬜	86	V	V	110	n	🬫
39	'	🬆	63	?	🬝	87	W	W	111	o	🬬
40	(	🬇	64	@	@	88	X	X	112	p	🬭
41	)	🬈	65	A	A	89	Y	Y	113	q	🬮
42	*	🬉	66	B	B	90	Z	Z	114	r	🬯
43	+	🬊	67	C	C	91	[	[	115	s	🬰
44	,	🬋	68	D	D	92	\	\	116	t	🬱
45	-	🬌	69	E	E	93	]	]	117	u	🬲
46	.	🬍	70	F	F	94	↑	↑	118	v	🬳
47	/	🬎	71	G	G	95	-	-	119	w	🬴
48	0	🬏	72	H	H	96	´	🬞	120	x	🬵
49	1	🬐	73	I	I	97	a	🬟	121	y	🬶
50	2	🬑	74	J	J	98	b	🬠	122	z	🬷
51	3	🬒	75	K	K	99	c	🬡	123	{	🬸
52	4	🬓	76	L	L	100	d	🬢	124	,	🬹
53	5	▌	77	M	M	101	e	🬣	125	}	🬺
54	6	🬔	78	N	N	102	f	🬤	126	—*	🬻
55	7	🬕	79	O	O	103	g	🬥	127	■	█

ASCII codes (A) for character mode (C) and graphic mode (G).

Decimal codes obtained from function keys and their correspondence in ABC 800.

	UNSHIFT	SHIFT	CTRL	SHIFT+CTRL
PF1	CTRL/1 192	CTRL/SHIFT/1 208	CTRL/SHIFT Q 224	CTRL/SHIFT A 240
PF2	CTRL/2 193	CTRL/SHIFT/2 209	CTRL/SHIFT W 225	CTRL/SHIFT S 241
PF3	CTRL/3 194	CTRL/SHIFT/3 210	CTRL/SHIFT E 226	CTRL/SHIFT D 242
PF4	CTRL/4 195	CTRL/SHIFT/4 211	CTRL/SHIFT R 227	CTRL/SHIFT F 243
PF5	CTRL/5 196	CTRL/SHIFT/5 212	CTRL/SHIFT T 228	CTRL/SHIFT G 244
PF6	CTRL/6 197	CTRL/SHIFT/6 213	CTRL/SHIFT Y 229	CTRL/SHIFT H 245
PF7	CTRL/7 198	CTRL/SHIFT/7 214	CTRL/SHIFT U 230	CTRL/SHIFT J 246
PF8	CTRL/8 199	CTRL/SHIFT/8 215	CTRL/SHIFT I 231	CTRL/SHIFT K 247

The following commands are used as control functions and are typed at the keyboard:

RETURN — the DO IT command
CTRL/C — stops program execution
(NOTE! CTRL/C terminates execution twice)
CTRL/H or ← — erases one character
CTRL/I or → — used for editing
CTRL/L — clears the screen
CTRL/S — single step execution
CTRL/X or CE — erases the last entered line

Appendix 6: Differences between ABC 800 and ABC 806

This appendix contains the differencies in the BASIC program for ABC 806 as compared with the program for ABC 800. The paragraphs affected are indicated with ABC 806 in the text margin.

6.4 Guide to the Statements

(Page 21)

The paragraph ”To define and ... **INPUTLINE.**” is omitted.

Miscellaneous statements:

COMMON and **DIM** sets the size of variables.
STOP, TRACE, and **NOTRACE** facilitate the debugging of a program.
WIDTH chooses the number of characters per line (40 or 80).

8 Commands

(Page 24)

- the devices are addressed as DR0:, DR1:, PR:, V24:, or CON:.
- the primary default device is disk drive 0 (DR0:) and the secondary one is disk drive 1 (DR1:).

WIDTH (page 34) Selects a number of characters per line

OPEN (page 51)

Format **OPEN** ”[device:][file name[.extension]]” **AS FILE** file number

Where	<device> may be for instance
DR0:	Disk drive 0
DR1:	Disk drive 1
PR:	Printer
CON:	Keyboard and screen

The expression following **AS FILE** should be an integer value between 0 and 255.

WIDTH (page 61)

Format

WIDTH [#file number,] number of characters

Function

Denotes number of characters per line for the current file. When # file number is omitted the screen is intended (40 alt. 80 characters).
The number of characters is a number between 0 and 255.

Use

Is used for conversion of line length.

Example

```
10 WIDTH 40
```

Converts ABC 806 to 40-character mode.

```
10 WIDTH #1.64
```

Converts file 1 to 64 characters per line.

10.3 Other Functions

CUR (page 73)

Format

CUR(L%,N%)

where L% (line) is in the interval 0-24 (physical 1-25) and N% (position) in the interval 0-39/79. Line 24 is a status line.

SYS (page 76)

Function

System information as follows:

SYS (2)	Total storage space
SYS (3)	Program size
SYS (4)	Remaining storage space
SYS (5)	Keyboard flag. Can be cleared by means of **GET, INPUT,** or **INPUT LINE.**
SYS (6)	Puts back the last input character into the keyboard buffer
SYS (8)	Is −1 when a key is pressed
SYS(11)	Starting adress of the program
SYS(12)	Variable root

11 Graphics including attribute handling (page 78)

ABC 806 can use graphics with a resolution of 78/158 x 72 pixels (picture elements). If the 40 character mode (78 pixels) is used, the graphics are compatible with the VIDEOTEX standards.

A separate attribute in the computer can be used for storing control characters for text and graphics. However, in the VIDEOTEX mode the control characters are saved in the character storage. The instruction **ATTRIBUTE** 1 enables you to determine which version to use.

High resolution graphics (see chapter 12) can be used in conjunction with ordinary text and graphics, since the high resolution graphics are saved in a separate storage. By defining a priority of your own choice, you determine what will be displayed first.

11.1 VIDEOTEX graphics

If no attribute or **ATTRIBUTE** 0 has been programmed, all characters will be stored in the ordinary character memory. When the memory content is displayed on the screen, each control character will occupy one position.

ABC 806 graphics correspond to the VIDEOTEX standards. In the graphic mode every output character is interpreted as a graphic character formed by a combination of six graphic points.

When text or graphics are displayed on the screen, the selection of colours etc. is controlled by means of certain arguments in the **PRINT** statement. The statement affects one line at a time. Each argument puts a control character on the screen. Although these characters are invisible, they take up one position each. The control characters can be covered by a background colour, if the control arguments are given in the correct order.

The following colours are available:

Black **(BLK)**
Red **(RED)**
Green **(GRN)**
Yellow **(YEL)**
Blue **(BLU)**
Magenta **(MAG)**
Cyanide **(CYA)**
White **(WHT)**

The characters available in the ABC 806 are listed below. The table gives the ASCII value of each character and its meaning in the character mode and graphic mode. One way of planning a graphical picture is to draw it on a copy of the graphics chart and feed the program the appropriate data.

When you have finished the picture on a copy of the chart you can type the lines one by one. Do not forget to allow space for the control characters, if you work with **ATTRIBUTE** 0 or if you have not indicated any attribute.

Note that the capital letters still remain the same in graphic mode. You can mix capital letters and graphic characters just as you like.

The number of graphical positions/line is dependent upon which character mode has been selected, 40 or 80 characters. In VIDEOTEX standards there are only 40 characters/line.

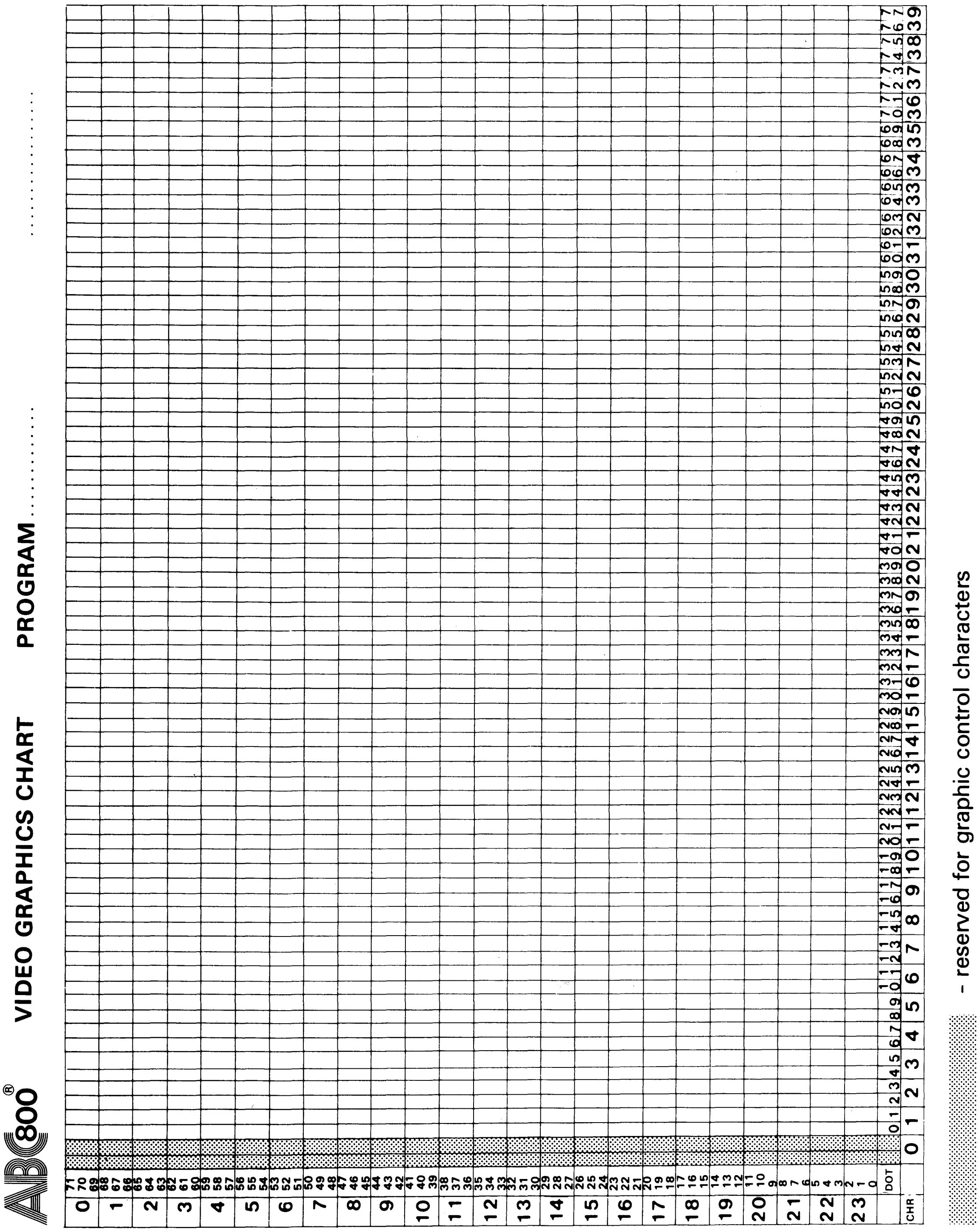
11.1.1 ASCII codes for characters, graphics and arguments

A	C	G	A	C	G	A	C	G	A	C	G
32	Blank	□	56	8	🬖	80	P	P	104	h	🬦
33	!	🬀	57	9	🬗	81	Q	Q	105	i	🬧
34	''	🬁	58	:	🬘	82	R	R	106	j	▐
35	#	🬂	59	;	🬙	83	S	S	107	k	🬨
36	$	🬃	60	<	🬚	84	T	T	108	l	🬩
37	%	🬄	61	=	🬛	85	U	U	109	m	🬪
38	&	🬅	62	>	🬜	86	V	V	110	n	🬫
39	'	🬆	63	?	🬝	87	W	W	111	o	🬬
40	(	🬇	64	@	@	88	X	X	112	p	🬭
41	)	🬈	65	A	A	89	Y	Y	113	q	🬮
42	*	🬉	66	B	B	90	Z	Z	114	r	🬯
43	+	🬊	67	C	C	91	[	[	115	s	🬰
44	,	🬋	68	D	D	92	\	\	116	t	🬱
45	-	🬌	69	E	E	93	]	]	117	u	🬲
46	.	🬍	70	F	F	94	↑	↑	118	v	🬳
47	/	🬎	71	G	G	95	–	–	119	w	🬴
48	0	🬏	72	H	H	96	´	🬞	120	x	🬵
49	1	🬐	73	I	I	97	a	🬟	121	y	🬶
50	2	🬑	74	J	J	98	b	🬠	122	z	🬷
51	3	🬒	75	K	K	99	c	🬡	123	{	🬸
52	4	🬓	76	L	L	100	d	🬢	124	,	🬹
53	5	▌	77	M	M	101	e	🬣	125	}	🬺
54	6	🬔	78	N	N	102	f	🬤	126	—*	🬻
55	7	🬕	79	O	O	103	g	🬥	127	■	█

ASCII codes (A) for character mode (C) and graphic mode (G).

Argument	CHR$( )	Argument	CHR$( )
BLK	128	**GBLK**	144
RED	129	**GRED**	145
GRN	130	**GGRN**	146
YEL	131	**GYEL**	147
BLU	132	**GBLU**	148
MAG	133	**GMAG**	149
CYA	134	**GCYA**	150
WHT	135	**GWHT**	151
FLSH	136	**HIDE**	152
STDY	137	**GCON**	153
ULN	138	**GSEP**	154
NULN	139	**BLBG**	156
NRML	140	**NWBG**	157
DBLE	141	**GHOL**	158
EL	142	**GREL**	159

11.1.2 Map of graphics for 40 characters



11.2 Handling of attributes

When operating with ordinary text and graphics, the computer handles attributes without having to display control characters (for graphics and the like) on the screen. Instead, all information of this kind is saved in the attribute storage, which can contain information about

- Colour of the character
- Background colour
- Steady character
- Flashing character
- Double character height
- Double character width
- Underlining
- Concealed text
- VIDEOTEX graphics
- Soft-scroll; to be generated from disk

If **ATTRIBUTE** 1 has been specified, all control characters will be stored in a separate attribute memory. Then colours etc. can be changed one by one without affecting the content of the character memory. Since the control character and the actual character are in parallel in the memory, they will be displayed in the same position on the screen.

11.3 Programming examples

The difference between **ATTRIBUTE** 0 and **ATTRIBUTE** 1 is evident from the two examples below. Each control character takes up one position on the screen when **ATTRIBUTE** 0 is used.

```
100 !
100 ! Example of using
120 ! the instruction ATTRIBUTE 0
130 !
140 ATTRIBUTE 0
150 PRINT CHR$(12)
160 PRINT ”Using ATTRIBUTE 0”
170 PRINT
180 PRINT CYA NWBG BLK ”Control” EL MAG ”characters” FLSH RED ULN
    ”occupy” NULN STDY NRML BLK ”character positions” BLBG WHT ”on
    the display.”
```

```
100 !
110 ! Example of using
120 ! the instruction ATTRIBUTE 1
130 !
140 ATTRIBUTE 1
150 PRINT CHR$(12)
160 PRINT ”ATTRIBUTE 1”
170 PRINT
180 PRINT CYA NWBG BLK ”Control” EL MAG ”characters” FLSH RED ULN
    ”do NOT occupy” NULN STDY NRML BLK ”character positions” BLBG
    WHT ”on the display.”
```

11.4 Instructions (page 81)

ATTRIBUTE

Format **ATTRIBUTE**n

Function Specifies whether the attributes of text and graphics shall be stored in the character memory (compatible with ABC 800 if no attribute is specified) or in the attribute memory. The instruction also controls the function soft-scroll.

ATTRIBUTE 0 Mode compatible with ABC 800. The attributes of text and graphics are stored in the character memory.

ATTRIBUTE 1 The attributes of text and graphics are stored in the attribute memory.

ATTRIBUTE 2 Activates soft-scroll.

Different attributes can be combined, e.g. 3, which will engage both the soft-scroll function and storing in the attribute memory.

The drive routine **SOFTOPT.REL** that handles attribute 2 is generated from a disk.

PRINT

Format **PRINT** [**CUR**(L,N)]argument [;argument;...]”text”

Function Used for printing text and graphics. The arguments control the colour selection etc. A G at the beginning of the colour selection argument (e.g. **GRED**) sets the line to the graphic mode so that all characters within quotes are interpreted as being graphics (see the ASCII table). If **CUR** (L,N) is specified, the picture is drawn from the starting point at line L (0-24), position N (0-39/79).

The following arguments are available:

RED, GRN, YEL, **BLU, MAG, CYA,** **WHT**	Alphanumeric colour characters
GRED, GGRN, **GYEL, GBLU,** **GMAG, GCYA,** **GWTH, GBLK**	Colour graphics and capital letters
FLSH, STDY	Flashing, steady
EL, DBLE	Double width and height
NRML	Normal width and height

GCON, GSEP	Continuous, separated graphics N.B. Cannot be generated on ABC 806
NWBG, BLBG	New background, black background
GHOL, GREL	Hold, release graphics N.B. Cannot be generated on ABC 806
HIDE	Concealed text/graphics
ULN, NULN	Underline, not underline

The arguments can also be given with **CHR$**.
The control arguments should be given in the following order:

PRINT <background colour argument> <argument for new background colour> <text colour argument> "Text" <argument for black background>

Example

```
10 PRINT RED NWBG GYEL "1,6 BOW WOW"
```

The results will be a yellow dog with red background colour.

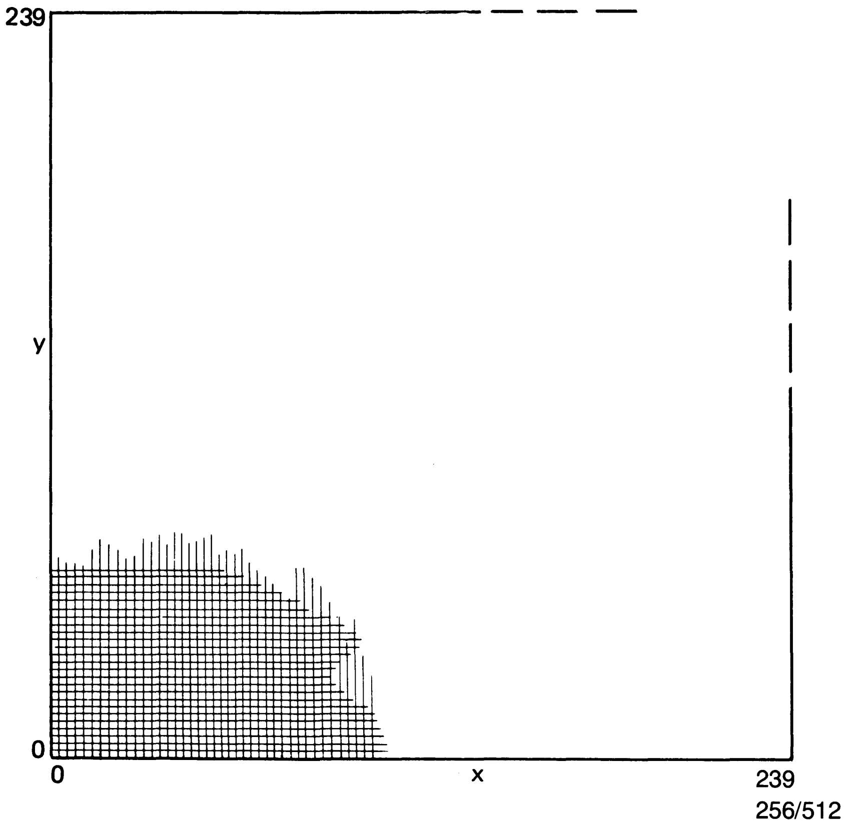
The instructions **TXPOINT, SET DOT, CLR DOT**, and **DOT** in an ABC 800 program cause no action on the screen of ABC 806. Instead you will get an error message, see chapter 15.

12 High Resolution Graphics (page 83)

High resolution graphics can be displayed in three different ways.

- Graphics according to ABC 800, where colours are selected by means of the table in section 12.4. Resolution is 240×240 picture elements (pixels).
- Graphics freely selected among four colours. Text and graphics can be mixed on the screen and they can be assigned different priorities. Resolution is 512×240 pixels.
- Graphics freely selected among 2×8 colours. Text and graphics can be mixed on the screen and they can be assigned different priorities. Resolution is 256×240 pixels.

Each pixel can be addressed independent of the others by specifying its X and Y coordinates. The origin of the picture is in the lower, left-hand corner and the positions are numbered from 0 to 239, alt. 256/512 and 0 to 239.



The screen is adjusted to obtain the following relations between height and width:

	Height (mm)	Width (mm)	W/H
ABC 812/ABC 815	187 ±2	225 ±2	1.2

Which resolution is obtained (256 or 512 pixels) is dependent on the number of colours selected. If you select no more than four colours, the resolution will be 512×240 pixels.

Example **FGCTL GRED+ RED+ BLU+ GRN**
0 1 2 3

After selection, the colours are assigned the serial numbers 0-3, which are used for the instructions **FGFILL, FGLINE** and **FGPAINT**.

The letter G before the name of the colour means that this is the foregound colour. As described above, the instruction **FGFILL** X, Y, 0 will assign priority 1 to the graphics, i.e. the graphics will be in front of any text displayed. On the other hand, if **FGFILL** X, Y, 1 is specified, the graphics will be behind the text.

If more than four colours are selected, the resolution will be 256×240 pixels and colours in the range 0 to 15 can be used.

Example **FGCTL RED+ GRN+ GBLU... GWHT**
0 1 2 15

You can use the instruction **FGPICTURE** to select a certain picture to be displayed while the computer is generating another picture.

Example **FGPICTURE** 0, 1, 2

In this example picture 1 will be displayed while picture 0 is generated. The last digit denotes the number of pictures.

However, a picture is not displayed automatically. You must give the instruction **FGCTL** to display the picture on the screen. If **FGPICTURE** is not specified, picture 0 will be displayed.

The high resolution graphics can be used together with text and graphics (according to chapter 11), since the information is stored in separate memories.

If the high resolution graphics is not used, the whole graphics memory space (128 Kbyte) can be used as data storage or RAM-floppy.

12.1 High resolution graphics on ABC 815

To display high resolution graphics on ABC 815, you must state covering graphics in the colour selection command, e.g. **FGCTL** RED + GBLU. Note that even the colour selection command **GBLK** will result in visible graphics on a monochrome screen.

12.2 Instructions (Page 84)

The general instructions below are applicable.

- The colour number is a number from 0 to 3 alt. 0 to 15. When using a mode compatible with ABC 800 the meaning of the number can be seen in section 12.4.
- The colour number is optional. If no colour number is given, the previous colour number will be used.

FGCTL

Format

FGCTL colour selection command (compatible mode)
FGCTL colour + colour..........

Function

Selects the colour combination to be used. In compatible mode the colour selection command is controlled by a number between 0-255, according to 12.4. The choise between 256/512 pixels/line is defined by the selection of colours.

The following colours are available

Background colour	Foreground colour	
BLK	**GBLK**	Black
RED	**GRED**	Red
GRN	**GGRN**	Green
YEL	**GYEL**	Yellow
BLU	**GBLU**	Blue
MAG	**GMAG**	Magenta
CYA	**GCYA**	Cyanide
WHT	**GWHT**	White

Example

FGCTL 132

Compatible mode with colour combination 132.

FGCTL BLK+RED+GCYA+GYEL

512 pixels per line with four colour parameters.

FGCTL BLK+RED+BLU+WHT+GRED...

256 pixels per line with 16 colour parameters.

FGFILL

Format

FGFILL x,y[,colour number]

Function

Fills a rectangle from the previous position to the position indicated by the coordinates (x,y)

FGLINE

Format

FGLINE x,y[,colour number]

Function

Draws a line from the previous position to the position indicated by the coordinates (x,y).

FGPAINT

Format

FGPAINT x,y[,colour number]

Function

Fills a closed area.

FGPICTURE

Format

FGPICTURE No. 1, No. 2, number of pictures.

Function

Controls which picture shall be displayed and which picture shall be generated. This enables you to display one picture while updating another one. The picture designated No. 2 is displayed whereas No. 1 is the picture to be updated. Pictures are numbered from 0 to 3. The number of pictures is designated by figures 1 through 4.

Example

FGPICTURE 2, 3, 4

Picture 3 is displayed while picture 2 is updated.
Four pictures are used.

CAUTION!
The graphics memory will be erased if you shift to high resolution graphics by using a colour selection command that is compatible with ABC 800.

FGPOINT

Format **FGPOINT** x,y[,colour number]

Function Will turn on a pixel in position x,y

Example **FGPOINT** 100, 100, 2

Note **FGPOINT** (x,y) assigns the colour number of pixel (x,y).

FGPOINT, function

Format **FGPOINT** (x,y)

Function Assign the colour number of pixel (x,y)

Example **FGPOINT** (100,100)

According to the example above it will be 2.

12.3 Examples

Example 1

```
100 PRINT CHR$(12)
110 PRINT CUR (3,7) 'This program show priority and not priority in high resolu-
    tion graphics.'
120 PRINT CUR (10,0);
130 FOR I=32 TO 127
140 PRINT CHR$(I);
150 NEXT I
160 ! Empty the high resolution graphics memory
170 FGCTL+BLK+GRED+BLU+GGRN
180 FGPOINT 0,0,0
190 FGFILL 511,239
200 ! Draw two rectangles
210 FGPOINT 0,125,1
220 FGFILL 50,239
230 FGPOINT 511,239,2
240 FGFILL 450,0
250 ! Draw a triangle
260 FGPOINT 200,100,2
270 FGLINE 250,105
280 FGLINE 230,200
290 FGLINE 200,100
300 ! Paint a triangle
310 FGPAINT 205,101,3
```

Example 2

```
100 EXTEND
110 FLOAT
120 PRINT 'Wait!!'
130 DEF FNDelay(K) LOCAL J
140  J=0
150  WHILE J<K
160   J=J+1
170  WEND
180  RETURN 0
190 FNEND
200 ! Draw a circle
210 DEF FNCircle (X,Y) LOCAL Fi,Epsylon,R
220  Fi=0
230  R=80
240  Epsylon=(2*PI/36)
250  FGPOINT X+2*R,Y
260  WHILE Fi<2*PI
270   Fi=Fi+Epsylon
280   FGLINE X+2*(R*COS(Fi)),Y + 1.2*(R*SIN(Fi)),1
290  WEND
300  FGPAINT X,Y,2
310  RETURN 0
320 FNEND
330 ! Clean 4 sides and draw a circle on each side
340 FGCTL BLK+BLK+BLK+BLK
350 I=0
360 WHILE I<=3
370  FGPICTURE I,I,I+1
380  FGPOINT 511,239,0
390  FGFILL 0,0
400  FGFILL 511,10,3
410  Z=FNCircle(255,SQR(I)*3+107)
420  I=I+1
430 WEND
440 FGCTL GBLK+GRED+GBLU+GGRN
450 ! Bounce until you press CTRL-C
460 WHILE-1
470  I=3
480  ! The ball falls
490  WHILE I>=0
500   FGPICTURE 0,1,4
510   I=I-1
520   Z=FNDelay(I*13)
530  WEND
540 I=0
550  ! The ball bounces up
560  WHILE I<=3
570   FGPICTURE 0,1,4
580   I=I+1
590   Z=FNDelay(1*13)
600  WEND
610 WEND
```

12.3 Animation mode (Page 85)

This section is not valid for ABC 806.

12.4 Colour Selection Table (Page 86)

The colour selection command (according to the table below) is in the interval 0-255.

For colour selection commands 1–71 the colour 0 is displayed behind the text, whereas colours 1–3 are displayed in front of the text.

In the interval 72–127 the colour 0 and the colour being the same as colour 0 are displayed behind the text, whereas the other two colours are displayed in front of the text.

From 128 upwards all high resolution graphics are displayed in front of the text.

See table on pages 86–88.

12.5 Examples (page 89)

The examples are not applicable to ABC 806.

13 Function Keys (Page 90)

The computer enables the use of special function codes. These are generated in different ways dependent on which keyboard is used. The table below shows which keys are to be operated on the different keyboards in order to generate the function codes. A total of 32 different codes can be used to accomplish different, fixed functions when you are programming. Examples of such functions are cursor movements, change of pages or jump to a program module.

A programmer can assign various functions to the function keys, e.g. cursor movements, jump to a program module etc. For assigning the same functions to ABC 802 as those included in ABC 800, the following combinations are used.

ABC 77 and ABC 22	Value	ABC 55
PF1	192	CTRL+1
PF2	193	CTRL+2
PF3	194	CTRL+3
PF4	195	CTRL+4
PF5	196	CTRL+5
PF6	197	CTRL+6
PF7	198	CTRL+7
PF8	199	CTRL+8

ABC 77 and ABC 22	Value	ABC 55
SHIFT+PF1	208	CTRL+SHIFT+1
SHIFT+PF2	209	CTRL+SHIFT+2
SHIFT+PF3	210	CTRL+SHIFT+3
SHIFT+PF4	211	CTRL+SHIFT+4
SHIFT+PF5	212	CTRL+SHIFT+5
SHIFT+PF6	213	CTRL+SHIFT+6
SHIFT+PF7	214	CTRL+SHIFT+7
SHIFT+PF8	215	CTRL+SHIFT+8
SHIFT+PF1	224	CTRL+SHIFT+Q
SHIFT+PF2	225	CTRL+SHIFT+W
SHIFT+PF3	226	CTRL+SHIFT+E
SHIFT+PF4	227	CTRL+SHIFT+R
SHIFT+PF5	228	CTRL+SHIFT+T
SHIFT+PF6	229	CTRL+SHIFT+Y
SHIFT+PF7	230	CTRL+SHIFT+U
SHIFT+PF8	231	CTRL+SHIFT+I
SHIFT+CTRL+PF1	240	CTRL+SHIFT+A
SHIFT+CTRL+PF2	241	CTRL+SHIFT+S
SHIFT+CTRL+PF3	242	CTRL+SHIFT+D
SHIFT+CTRL+PF4	243	CTRL+SHIFT+F
SHIFT+CTRL+PF5	244	CTRL+SHIFT+G
SHIFT+CTRL+PF6	245	CTRL+SHIFT+H
SHIFT+CTRL+PF7	246	CTRL+SHIFT+J
SHIFT+CTRL+PF8	247	CTRL+SHIFT+K

Example (ABC 55):
Functions PF1 to PF8 are obtained by simultaneous pressing of CTRL+1....8
SHIFT+PF1–PF8 is generated by pressing CTRL+SHIFT+1....8
CTRL+PF1–PF8 is generated by pressing CTRL+SHIFT+Q....I
SHIFT+CTRL+PF1–PF8 is generated by pressing CTRL+SHIFT+A....K.

When a function key is pressed, a subroutine can be called as shown below:

Example

```
 10 ON ERROR GOTO 100
 20 INPUT ”Number”,P(I)
 30 I=I+1
 40 GOTO 20
  I
  I
100 IF ERRCODE <> 53 THEN RESUME
110 A=SYS(6)
120 GET X$
130 ON ASC(X$)–191 RESUME 400, 500, 600
```

When a function key is pressed at **INPUT** or **INPUT LINE**, an error is generated. The **ERRCODE** is 53. The program should contain a routine which handles error 53. To find out which one of the function keys that was pressed, use the function **SYS**(6) and read the character by means of **GET**.

14 Differences in BASIC between ABC 800 and ABC 80 (Page 91)

This section is not valid for ABC 806.

14 Differences in BASIC between ABC 806 and ABC 800 (Page 91)

A program that has been written on ABC 806 and that includes any of the new instructions, e.g. **ATTRIBUTE**, cannot be loaded into ABC 800.

The new instructions are:
ATTRIBUTE
FGCTL colour+colour
FGPICTURE
WIDTH

The instructions **TXPOINT, SET DOT, CLR DOT** and **DOT** in a program for ABC 800 cause no action on the screen of ABC 806. However, the error message 200 is generated. See chapter 15.

16 Summary of Commands and Instructions (Pages 96, 97, 104, and 105)

CLR DOT, DOT, SCR, SET DOT and **TXPOINT** are omitted.

ATTRIBUTE (page 95) (instruction) Page 135

Format **ATTRIBUTE**n

Function Specifies whether the attributes of text and graphics shall be stored in the attribute memory or in the character memory.
N.B.
Is not **ATTRIBUTE** specified the storing will be in the character memory.

FGPICTURE (page 98) Page 139

Format **FGPICTURE** no. 1, no. 2, number of pictures

Function Controls which picture shall be displayed and which picture shall be generated.

SYS (page 105) (function) Page 76

Format **SYS**(I%)

Function The system status as follows:

SYS(2)	Total storage space
SYS(3)	Program size
SYS(4)	Remaining storage space
SYS(5)	Keyboard flag
SYS(6)	Puts back the last input character into the keyboard buffer
SYS(8)	Is −1 when a key is pressed
SYS(11)	Starting adress of the program
SYS(12)	Variable root

WIDTH (page 106) (instruction) Page 61

Format **WIDTH** [#file number] number

Function Denotes number of characters per line.

Appendix 1: BASIC II Errata (Page 108)

Appendix 1 is omitted.

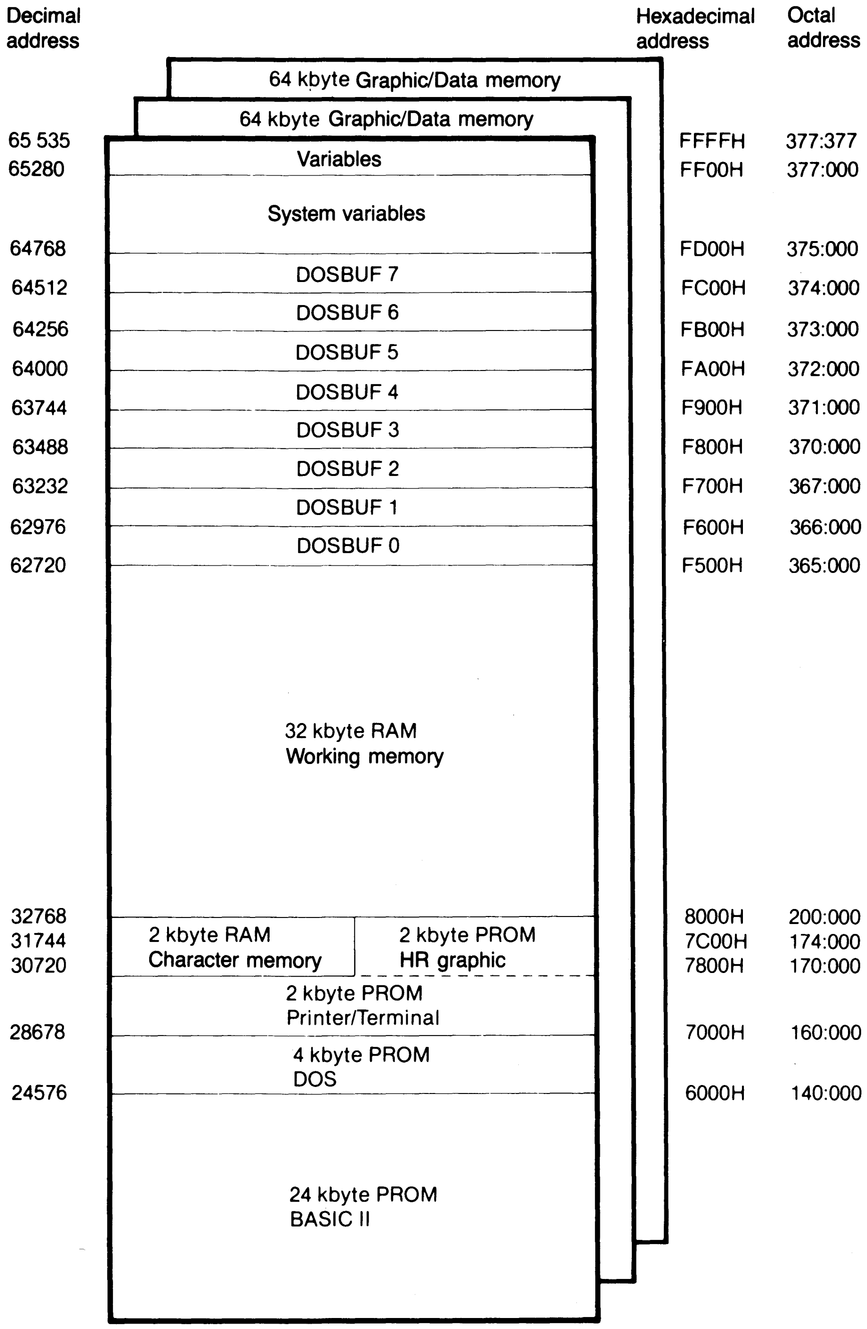
Appendix 2: The I/O ports of ABC 806 (Page 108)

Port	Address bit	Function	Output	Input
	7 0		Decimal	
ABC bus	000XX000	Input port 0		0
	000XX001	Input port 1		1
	000XX010	Input port 2		2
	000XX111	I/O RESET		7
	000XX000	Output port 0	0	
000XX001	Output port 1	1		
	000XX010	Output port 2	2	
	000XX011	Output port 3	3	
	000XX100	Output port 4	4	
	000XX101	Output port 5	5	
HR graphics	000XX110	HRC	6	
	000XX111	HRS	7	
ABC bus		XOUTSTB, XINSTB	0–31	
DART	0010XX00	Printer data CH.A	32	32
	0010XX01	Printer control CH.A	33	33
	0010XX10	Keyboard data	34	34
	0010XX11	Keyboard control	35	35
CRTC	00110XX1	Read register		49
VIDEO	00110100	Table of memory blocks	52	52
	00110101	Attribute	53	53
	00110111	Sync signal delay	55	
	00110111	FGCTL PROM+CLOCK		55
CRTC	00111XX0	Write register address	56	
	00111XX1	Write register	57	
SIO/2	010XXX00	V24 data CH.B	64	64
	010XXX01	V24 control CH.B	65	65
CTC	011XXX00	Channel 0	96	96
	011XXX01	Channel 1	97	97
	011XXX10	Channel 2	98	98
	011XXX11	Channel 3	99	99
ABC bus	1XXXXXXX	XOUTSTB, XINSTB	128–255	

Addresses not presented are not used
X = Don't care

Appendix 3: Storage Disposition (Page 109)

ABC 806 Memory Map with Disk Drives



Decimal address		Hexadecimal address	Octal address
	64 kbyte Graphic/Data memory		
	64 kbyte Graphic/Data memory		
65 535		FFFFH	377:377
	Variables		
65280		FF00H	377:000
	System variables		
64768		FD00H	375:000
	DOSBUF 7		
64512		FC00H	374:000
	DOSBUF 6		
64256		FB00H	373:000
	DOSBUF 5		
64000		FA00H	372:000
	DOSBUF 4		
63744		F900H	371:000
	DOSBUF 3		
63488		F800H	370:000
	DOSBUF 2		
63232		F700H	367:000
	DOSBUF 1		
62976		F600H	366:000
	DOSBUF 0		
62720		F500H	365:000
	32 kbyte RAM Working memory		
32768		8000H	200:000
31744	2 kbyte RAM Character memory / 2 kbyte PROM HR graphic	7C00H	174:000
30720		7800H	170:000
	2 kbyte PROM Printer/Terminal		
28678		7000H	160:000
	4 kbyte PROM DOS		
24576		6000H	140:000
	24 kbyte PROM BASIC II		

The display storage (2 kbytes) is parallel with the system program for high resolution graphics. The two areas of the memory do not interact. ABC 806 runs in a special mode when the graphics storage is addressed. When the operating system CP/M is loaded parts of the graphics storage is used.

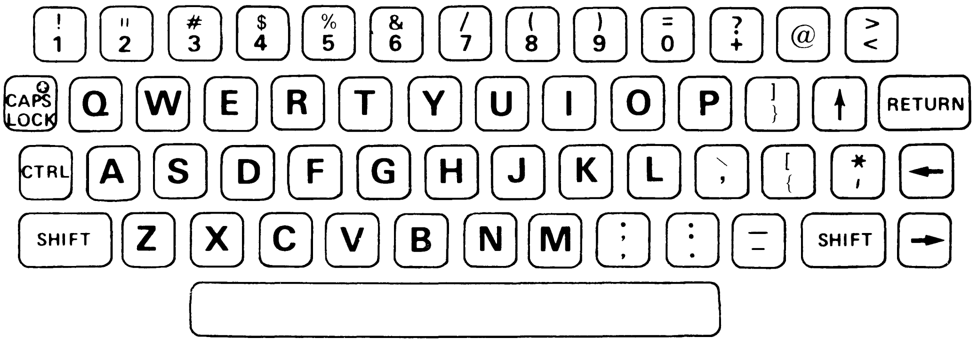
If storage space for machine language routines is to be allocated, the following addresses are changed:

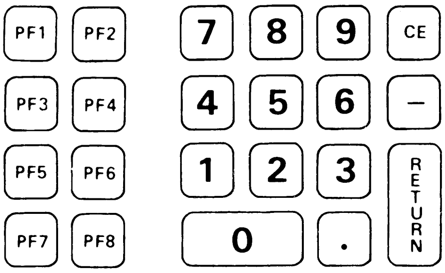
- The pointer for the lowest memory address of a BASIC program (BOTTOM): 65292
- The pointer for the highest memory address of a BASIC program (TOP): 65294.

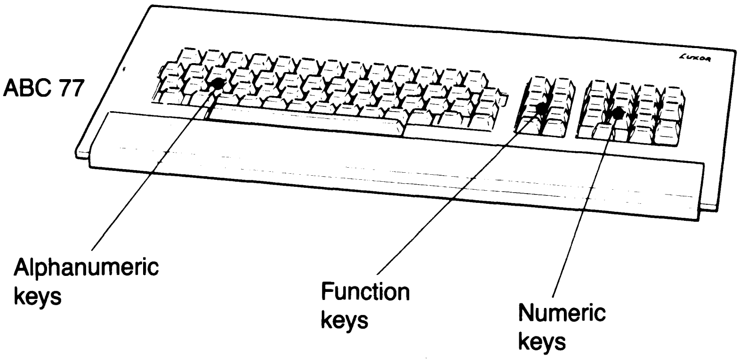
The graphics/data memory cannot be directly addressed in BASIC.

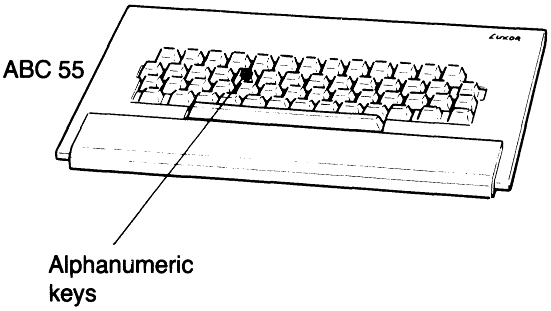
Appendix 4: Keyboard Layout, ASCII Codes

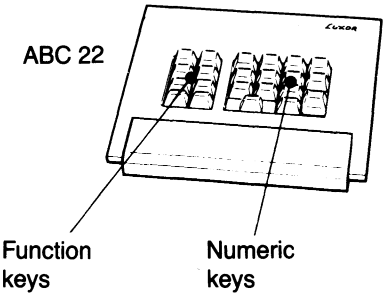
(Page 111)


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









Codes obtained from the keyboard

ASCII code	CTRL	SHIFT	Key	ASCII name	Function
0	X		@	NUL	Time filler character
1	X		A	SOH	–
2	X		B	STX	–
3	X		C	ETX	Stops execution
4	X		D	EOT	–
5	X		E	ENQ	–
6	X		F	ACK	–
7	X		G	BEL	”Beep” in the loudspeaker
8	X		H	BS	*) ”←” key
9	X		I	HT	*) ”→” key
10	X		J	LF	Line feed
11	X		K	VT	–
12	X		L	FF	*) Erases screen
13	X		M	CR	*) ”RETURN” key
14	X		N	SO	–
15	X		O	SI	–
16	X		P	DLE	–
17	X		Q	DC1	–
18	X		R	DC2	–
19	X		S	DC3	Steps one program instruction
20	X		T	DC4	–
21	X		U	NAK	–
22	X		V	SYN	–
23	X		W	ETB	–
24	X		X	CAN	*) Deletes entered line
25	X		Y	EM	–
26	X		Z	SUB	–
27	X		[	ESC	–
28	X		\	FS	–
29	X		↑	GS	–
30	X		≈	RS	–
31	X	X	O	US	–
127	X		<	DEL	Generates a filled square (■)

*) These characters affect the screen directly.

The following commands are used as control functions and are typed at the keyboard:

RETURN	the DO IT command
CTRL/C	stops program execution (NOTE! CTRL/C twice, terminates execution)
CTRL/H or ←	erases one character
CTRL/I or →	used for editing
CTRL/L	clears the screen
CTRL/S	single step execution
CTRL/X or CE	erases the last entered line

19 Index

LUXOR
Computers

Art. nr. 66 78088-92

TELUB TECHNICAL DOCUMENTATION Växjö, Sweden 1984