

CBZ 80

Z80 BASIC COMPILER

QUICK REFERENCE

grifo[®]
ITALIAN TECHNOLOGY

Via dell' Artigiano, 8/6
40016 San Giorgio di Piano
(Bologna) ITALY
E-mail: grifo@grifo.it



<http://www.grifo.it> <http://www.grifo.com>
Tel. +39 051 892.052 (a. r.) FAX: +39 051 893.661

CBZ-80

Edition 5.02 Rel. 15 June 1999

, GPC[®], grifo[®], are trade marks of grifo[®]

CBZ 80

Z80 BASIC COMPILER

GUIDA RAPIDA

CBZ-80 is a powerful software development tool which allows to create programs for all the Z80 Zilog-based boards. It features a complete support for single and double precision floating point variables up to 54 digits accuracy. The development environment is extremely friendly so that to cut off the development time. The BASIC runs on eeprom and the generated code, by means of **GDOS**[®] features, runs on the on-board EEPROM or parallel EPROM; this way the need to use external hardware (such as in circuit emulator, EPROM programmer, etc.) and the debugging time are drastically reduced. Code productivity and hardware intervention ease make this BASIC compiler an unparalleled professional work tool at all levels.

grifo[®]

ITALIAN TECHNOLOGY

Via dell' Artigiano, 8/6
40016 San Giorgio di Piano
(Bologna) ITALY

E-mail: grifo@grifo.it

<http://www.grifo.it>

<http://www.grifo.com>

Tel. +39 051 892.052 (a. r.) FAX: +39 051 893.661



CBZ-80

Edition 5.02

Rel. 15 June 1999

, GPC[®], grifo[®], are trade marks of grifo[®]

DOCUMENTATION COPYRIGHT BY **grifo**[®], ALL RIGHTS RESERVED

No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, either electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written consent of **grifo**[®].

IMPORTANT

Although all the information contained herein have been carefully verified, **grifo**[®] assumes no responsibility for errors that might appear in this document, or for damage to things or persons resulting from technical errors, omission and improper use of this manual and of the related software and hardware.

grifo[®] reserves the right to change the contents and form of this document, as well as the features and specification of its products at any time, without prior notice, to obtain always the best product.

SYMBOLS DESCRIPTION

In the manual could appear the following symbols:

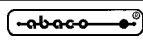


Attention: Generic danger



Attention: High voltage

Trade Marks

 [®], GPC[®], **grifo**[®] : are trade marks of **grifo**[®].

Other Product and Company names listed, are trade marks of their respective companies.

GENERAL INDEX

QUICK REFERENCE TO CBZ-80	1
GENERALITIES	2
CHARACTERISTICS OF CBZ-80	3
CBZ-80 REQUIREMENTS	5
CONVENTIONS	6
DATA TYPES AND THEIR LIMITS	7
OPERATOR LIST	8
STRING HANDLING FUNCTIONS LIST	9
INPUT/OUTPUT INSTRUCTIONS LIST	11
COMMANDS LIST	13
MACHINE SPECIFIC COMMANDS LIST	15
DISK ACCESS COMMANDS LIST	16
NUMERIC FUNCTIONS LIST	22
CURSOR POSITIONING INSTRUCTIONS LIST	24

FIGURE INDEX

FIGURE 1: ON LINE HELP STARTING SCREEN 1
FIGURE 2: CONFIGURATION REQUESTER 3
FIGURE 3: INTERNAL COMMAND LINE EDITOR 5



```

CBZ_80 tm 3.01 BASIC Compiler
(c) 1985, Grifo(r).
November 1st 1985
-----+-----
      (E)dit
      (C)onfigure
      (S)ave
      (P)atch
      (W)arm Start Creator
-----+-----
      CBZ_80 tm Basic Compiler
      Version 3.01 November 1st 1985
-----+-----
help
---- CBZ_80 HELP MENU ----
10. Commands          60. Machine Language
20. Line Editor       70. Screen, Printer, Input
30. String$          80. Statements
40. Graphics          90. Math
50. Disk Files       100. CPM80 2.2 & 3.0

Type HELP and ITEM NUMBER ABOVE
F10 Menu | TERMINAL EMULAT. for GDOS80 - GRIFO® Tel. +39-51-892052 | CAPS NUM

```

FIGURE 1: ON LINE HELP SCREEN

QUICK REFERENCE TO CBZ-80

This quick reference to CBZ-80 programming language lists all its keywords, along with a short description of function and use.

CBZ-80 is a powerful software development tool which allows high-level programming in BASIC on all the **grifo®** boards based on Z80 microprocessor family. The code compiled by **CBZ-80** needs to use the functions and features of **GDOS 80**, a rom-based Operating System. The development environment is extremely friendly and achieves to reduce the development time, being anyway compliant to the operational feeling of all the BASICs. Unexperienced programmers will be able to take advantage of its numerous commands and functions, becoming productive in few hours of work, while experienced programmers won't need any training. However the great code performance and the rapidity of hardware intervenes make the **CBZ-80** an unreplaceable work instrument for all the applications.

The compiler supports mathematic functions, control applications, data base management, interfacing to generic consoles, Operating System calls and many other features designed to solve industrial automation problems. The programmer can choose between to develop structured code and non-structured code, obtaining a level of efficiency and flexibility hard to see in other development tools of equal price.

GENERALITIES

CBZ-80 is a composite programming and development environment made by a set of independent items which the programmer has the liberty to use or not use without any limitation. Wishing to make comparisons amongst **CBZ-80** and other well-known BASIC programming tools, we detect that **CBZ-80** has an environment similar to GWBASIC's one and instruction set comparable to the QUICKBASIC's one.

CBZ-80 enables to take the greatest advantage of the hardware resources from the boards you are using, because you may use them directly by the high-level instructions, without no need to develop specific firmware. For example, **CBZ-80** has the capacity to manage hardware resources like serial lines, printers, mass storage devices, operator interfaces, etc.

CBZ-80 software package is made up by a set of disks, a rich reference manual and a great number of examples (both source and compiled code) showing how to employ the control board's hardware resources.

```

Double Precision accuracy 6-54 000E 00014 ?
Single Precision accuracy 2-52 0006 00006 ?
Scientific Precision. 2 to DBL 0006 00006 ?
Maximum File Buffers Open 0-99 0002 00002 ?
Array Base 0 or 1 0000 00000 ?
Rounding number 00 to 99 0031 00049 ?
Default Variable Type:
<I>nTEGER,<S>INGLE or <D>OUBLE I ?
Test Array Bounds <Y/N>. N ?
Convert to Upper case <Y/N>. N ?

CPM80 tm 2.2 and 3.0 SPECIAL Configuration See Appendix

Default CLEAR nnnn Memory Size 03E8 01000 ?
Clear Screen String (Hex Code) 000C 00012 ?
Clear End of Line. (Hex Code) 4B1B 19227 ?
Clear End of Page. (Hex Code) 6B1B 27419 ?
Cursor (off) string (Hex Code) 501B 20507 ?
Cursor (on) String (Hex Code) 4D1B 19739 ?
List Previous Line <KEY>. 0005 00005
List Next Line <KEY>. 0018 00024
List 1st line <KEY>. 0013 00019
List Last Line <KEY>. 0004 00004

F10 Menu | TERMINAL EMULAT. for GDOS80 - GRIFO® Tel. +39-51-892052 | NUM

```

FIGURE 2: CONFIGURATION REQUESTS

CBZ-80 FEATURES

Fundamentally **CBZ-80** has two main working modalities: configuration mode and source development mode. The main characteristics of these two modalities are described as follows:

- Configuration mode: in this situation becomes possible to set many internal parameters of the compiler which affect directly or indirectly the code generation.
 - Precision of floating point variables (from 2 to 54 digits)
 - Maximum number of open files at the same time (from 0 to 99)
 - Method of enumeration of the indexes for arrays and matrices
 - Approximation threshold for real variables
 - Type of undeclared variables
 - Array bounds check
 - Automatic uppercase conversion
 - Dimensions of indexed memory
 - Shortcuts to the most frequently used editor functions
 - Console control sequences. By default **CBZ-80** is configured for supporting the ADDS VIEWPOINT standards, which is used by **GET 80** and all the **QTP xxx** operator interfaces.
 - Memory area dedicated to chains
- Source development mode: this is the situation normally used by the end user and it includes the editor, the compiler and the debugging environment. Use of this mode is common to all languages, it allows to:

- 1) write and correct the source of the application (this phase can be performed by the integrated editor or an external ASCII editor, like the **GET 80**'s one).
- 2) upload the source to the board using the features of **GDOS 80** file system.

- 3) compile the uploaded program, to get the compiled code. In case of error, please return back to point 1
- 4) execute the compiled code directly on the control board . If during the functional test of the program problems are detected, you must go back to point 1
- 5) recompile the code in the final **GDOS 80** executable form (for example ready for EPROM, or FLASH EPROM burning).

Amongst the many characteristics of this development environment, we remind:

- Numbered or unnumbered BASIC source code; when line numbers are not used entry points are indicated by labels.
- Standard syntax; it allows to reuse code written and already tested on other BASIC programming environments.
- Four different data types: integer, single and double floating point, string.
- Wide range of operators including mathematical, relational, logical and shift operators.
- Complete set of mathematic functions including trigonometric and transcendental functions.
- Support for the most commonly used numeration bases (binary, hexadecimal, octal and decimal).
- Instruction set dedicated to the use of an operator interface (cursor positioning, partial or total screen clear, check for key pressed, data input, etc.). By means of these functions you may control the complete **QTP xxx** terminals serie.
- Wide range of **GDOS 80** file system management instructions set. There is no more need for low-level memory and data area management. **GDOS 80** takes care of this by manipulating RAM data files, which can be created, deleted, renamed, copied, downloaded etc.
- Interesting string manipulation instructions set (concatenation, fragmentation, search, conversion etc.).
- Indexed management of a memory area, which can be addressed using pointers.
- Powerful control flow instructions set, which allows to perform iterations, single or multiple tests, define functions and procedures, run other programs etc.
- Basic low-level hardware resources management instructions set, like I/O instructions, direct memory access, machine language routines, absolute calls to external procedures etc.
- High level devices management instructions set, which, by means of **GDOS 80** features, allows easy use of peripherals like printers and serial lines.
- Different compilation modalities which permit to optimize compilation times and compiled code.
- Complete management of chain technique, which makes possible to run any number of programs sequentially with data forward communication. Using this powerful feature, the problems of automaton involving great amounts of data and code can be easily solved.
- On line help, easy to use and capable to give a whole description of any part of **CBZ-80**, makes training faster.
- No license fee or overcharge, developers are free to create programs without even informing **grifo®**.

```
new
CBZ_80 Ready
DIR C:
CBZ_80.G80      CBZ_80.HLP      DEMO.ZBA      DEMO.BAK
CBZ_80 Ready
LOAD C:DEMO.ZBA
CBZ_80 Ready
LIST
00001 REM Demo program for CBZ 80
00002 DIM i%
00003
00004 PRINT "Demo program"
00005 FOR i%=0 TO 50
00006   PRINT USING"###",i%;
00007 NEXT i%
00008 STOP
CBZ_80 Ready
RUN
Demo program
 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 2
6 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
Break in 00008
CBZ_80 Ready
F10 Menu | TERMINAL EMULAT. for GDOS80 - GRIFO® Tel. +39-51-892052 | CAPS NUM
```

FIGURE 3: INTERNAL COMMAND LINE EDITOR

CBZ-80 REQUIREMENTS

Only three elements are required to be immediatly up and running:

- A Z80 based control board like:

GPC® 80F
GPC® 81F
GPC® 011
GPC® 15A
GPC® 15R
GPC® 153
GPC® 154
GPC® 150

- A GDOS 80 operating system for the desired control board.

- A personal computer, connected to the control board through a serial line.

CONVENTIONS

This quick reference uses the following typographic conventions:

KEYWORDS (boldface uppercase)

Keywords of CBZ-80.

<*placeholders*> (enclosed in angular brackets)

Variables, expressions, constants or other informations needed on each particular situation.

[*optional values*] (enclosed in square brackets)

Items that are not required.

item repeating . . . (followed by ellipsis (three dots))

You may add items with the same form.

DATA TYPES AND THEIR LIMITS

Here follows a list of the data types provided by **CBZ-80**, their bound values and their memory occupation in bytes.

Type	Minimum value	Maximum value	Memory occupation (in bytes)
INTEGER	-32768	+32767	2
REAL (BCD)	-9.999E+63	+9.999E+63 (From 6 to 54 digits)	From 2 to 28
HEXADECIMAL	&H0000	&HFFFF	2
OCTAL	&O000000	&O377777	2
BINARY	&Xbbbbbbbbbbbbbbbb		2
STRING	From 0 to 256 characters (see DEF LEN)		From 0 to 256

Overflow errors are raised only for BCD type.

To implicitly declare the type of a variable you should write its name followed by an opportune character, e.g.: **DIM B%** declares B% as an integer.

Type	Character
INTEGER	%
REAL (single precision)	!
REAL (double precision)	#
STRING	\$

OPERATORS LIST

C = command mode

R = run mode

\wedge or [R, C

Power raising, only for real numbers.

+ - * / \ R, C

Sum, subtraction, multiplication, division and real numbers division.

< = > <= or =< >= or => <> R, C

Conditional operators. Zero means false, one means true.

<numeric expression1> **AND** <numeric expression2> R, C

If the two expressions are true (non-zero), result is true, where true=1 and false=0.

Also used to compare bits with binary digits.

<expr1> **MOD** <expr2> R, C

Returns the remainder of the integer division between <espr1> and <espr2> with sign of <espr1> leading.

NOT <expr> R, C

Returns the opposite of <expr>.

<expr1> **OR** <expr2> R, C

Performs logical or bitwise **OR**.

<expr1> **XOR** <expr2> R, C

Performs logical or bitwise **XOR**.

<expr1> >> <expr2> R, C

Performs the logical right shift of <expr1>'s bits by <expr2> positions.

<espr1> << <espr2> R, C

Performs the logical left shift of <expr1>'s bits by <expr2> positions.

STRING HANDLING FUNCTIONS LIST

ASC(<constant, variable string, or substring reference>) R, C
Return the decimal value of argument string's first char, according to the ASCII code translation.
e. g.: ASC("B")=66, ASC("CLUNK")=67 (note: ASC is the inverse of CHR\$ function).

BIN\$(<numeric expression>) R, C
Calculates binary value of <numeric expression>. e. g.: BIN\$(255)=0000000011111111
Inversely &X0000000011111111=255

CHR\$(<numeric expression>) R, C
Return the ASCII char corresponding to the value of <numeric expression>. Argument must be an integer in the range 0 e 255.
e. g.: CHR\$(65)="A", CHR\$(97)="a", CHR\$(32)=" " (blank)

DATE\$ R, C
Returns system date in the format month/day/year.

HEX\$(<expr>) R, C
Converts the value of <expr> in hexadecimal format.
e. g.: HEX\$(255)=FF , HEX\$(170)=AA ,HEX\$(85)=55

INDEX\$(<string>) R
INDEX\$(<string>, <expr>)
Returns the index number associated to <string>. If <expr> is present then the search for <string> will start from position <expr>.
e. g.: INDEX\$(0) = "CIAO", INDEX\$(1) = "CIAO1" ,INDEX\$(2) = "CIAO2" INDEXF ("CIAO")=0, INDEXF ("CIAO1",2)=-1

INSTR (<expr>, <string1>, <string2>) R
Returns the position of <string2> in <string1>. <expr> specifies the starting position of search.
e. g.: A\$="Prova ciao" B\$="ciao" INSTR (1, A\$, B\$)=7

LEN (<string>) R, C
Returns the length of <string>.
e. g.: A\$="Prova ciao" PRINT LEN (A\$)= 10

LEFT\$(<string>, <expr>) R
Selects a substring of <expr> characters from <string>.
e. g.: A\$="Prova ciao" PRINT LEFT\$ (A\$,3)="Pro" PRINT LEFT\$ (A\$, 5)="Prova"

MID\$(<string>, <expr>, <expr>) R, C
Return a substring of <string> starting at the <expr1>th character of <string> and <expr2> characters long.

MKB\$(<expr>) R, C
Returns a string which contains the compressed floating point value of <expr>.

MKI\$ (<expr>)	R, C
Returns a two-characters string specified by the two-bytes integer <expr>.	
OCT\$ (<expr>)	R, C
Returns a string containing the octal (base 8) value of <espr>.	
PSTR\$ (<var>)	R
Returns a string whose address is the value of variable <var>.	
RIGHT\$ (<string>, <expr>)	R, C
Returns the last< expr> characters of <string>.	
SPACE\$ (<expr>)	R, C
Returns a string containing<expr> blanks.	
STR\$ (<expr>)	R, C
Returns a string containing the numeric value of <expr>.	
STRING\$ (<expr1>, <string> or <expr2>)	R, C
Returns a string containing <expr1> times the first chracter of <string> or the ASCII char corresponding to the code <expr2>.	
TIME\$	
Return system time in the format hour:minute:seconds.	
UCASE\$ (<string>)	R, C
Converts <string> in uppercase.	
UNS\$ (<expr>)	R, C
Returns a string containing the unsigned decimal value of <expr>.	

INPUT/OUTPUT INSTRUCTIONS LIST

CLS R, C

Clears the screen and positions the cursor in the upper left corner.

CLS <numeric expression >: Fills the screen with characters whose ASCII code is specified by <numeric expression >.

CLS LINE : Clear the line where the cursor finds.

CLS PAGE : Clears from the cursor position to the bottom of the screen.

DEF TAB =<number> R

Defines the number of characters (blanks) printed by a tabulation in the commands **PRINT** , **PRINT#** or **LPRINT** , ranging from 1 to 255 (default 16).

INPUT [@ (expr^x,expr^y)] [;] [!] [&expr] ["string";] <variable> [, <var.>] R

@ (expr^x,expr^y) Optional, allows to place the cursor at x, y coordinates.

; Optional, suppresses the trailing carriage return/line feed.

! Optional, enables automatic carriage return after a maximum number of input characters.

&<expr> Optional, enables automatic carriage return after <expr> input characters.

"string" Optional, prints "string" as a prompt.

<variabile> Any variable.

INPUT is used to input numeric or string values from keyboard into variables.

INKEY\$ R

Returns a string containing the key pressed. If no key has been pressed then returns an empty string.

e. g.: Type "A" ,A\$=INKEY\$, A\$="A"

LINEINPUT [@ (expr.x,expr.y)] [;] [!] [&expr] ["string";] <variable string> R

LINEINPUT is analog to **INPUT** except for it can input only into string variables. **LINEINPUT** accepts any ASCII value.

LPRINT [variables, constants, ...] R, C

Prints the value of its arguments to the printer. Precede it by a colon character <::> to use it in the Standard Line Editor (:LPRINT).

PAGE R, C

Returns the current line position of the printer.

PRINT [{ @ or % } (<expr1>, <expr2>)] **USING** <format string> ; <expr> ; [**USING** ...]
[<list of values to print>] R, C

Writes informations to the current device, normally the screen.

@ and % with <expr1> and <expr2> allow to place the cursor anywhere on the screen, **USING** <format string> allows to format the text.

SPC (<expr>) R, C

Prints <expr> spaces.

TAB (<expr>)

R

Moves cursor to the position <expr>.

WIDTH [**LPRINT**] [=] <expr>

Sets the print width for **PRINT** or **LPRINT**.

COMMANDS LIST

APPEND <line number> <file name>

Inserts part of a program or an ASCII subroutine (without lin numbers) starting from the specified line. e. g.: APPEND 1000 PROGRAM.ASC

APPEND* <line number> <file name>

Deletes REM's and blankc from an ASCII saved file, saving space in memory.

AUTO

Starts auto line number enumeration.

AUTO <number> Starts auto enumeration from <number> line number.

AUTO <number>,<increment> Starts auto enumeration from <number> line number and <increment> as increment step.

AUTO ,<increment> Starts auto enumeration with <increment> as increment step.

DELETE <line number1> - <line number2>

Every line with number ranging from <line number1> to <line number2> is deleted. <line number2> must be greater than <line number1>.

DIR <drive>

Lists the directory of the specified drive.

EDIT <line>

E <line>

Used to correct a specified line. Its sub-commands make the editing easy and fast.

I-Enters in Insert mode, exits pressing <ESC>.

X-Go to the end of the line and enter insert mode.

<n>D-Deletes <n> characters from the current cursor position.

<n>C<key>-Replaces from the character under the cursor with <key> character <n> times.

H-Deletes everything from the current cursor position to the end of the screen and enters insert mode.

<n>S<key>-Searches the <n>th occurrence of <key> .

L-Lists line being edited, then performs a carriage return/line feed.

A-Aborts changes, restores the line like it was before the editing.

<n>K<key>-Deletes text from cursor to the <n>th occurrence of <key> .

<n> <SP>-Moves cursor <n> characters to the right.

<n> <BS>-Moves cursor <n> characters to the left.

<ESC>-Exits insert mode and waits for a new command.

<ENTER>-Exits EDIT and lists the changed line.

<BREAK>-Abort key, exits EDIT session, no changes made.

FIND <string>

FIND #<string>

Locates text in a program, very useful with EDIT command, to find the next string type “;” or FIND <ENTER>.

Character “#” allows to find a line by its number.

e. g.: FIND #1234508000 GOTO 12345

HELP**HELP** <number>

Activates **HELP** menu, press the space key to continue, to exit you must reach the end of the **HELP** sequence.

If a number is specified then only that **HELP** item will be showed.

L[IST] [+][*] <line number> or <label>

Used from the Standard Line Editor, allows to list the content of the current program to the screen.

LLIST

Same syntax as **LIST**, sends its output to the printer.

LOAD [*] [“] <file name> [“]

Used from the Standard Line Editor, allows to load into memory an ASCII-saved or a compiled program. If the program doesn't have line numbers they will be added with one as increment .

LOAD * deletes comments and unnecessary blanks to save space in memory.

e. g.: **LOAD** PROVA.ZBA

MEM[ORY]

In command mode, returns informations about the use of the memory.

MERGE [“] <file name> [“]

Merges a line-numbered program on disk with the program in memory.

NEW

Deletes the program in memory.

QUIT

Exits from CBZ-80.

RENUM [new] [, [old]][, increment]

Recalculates the line numbers.

RUN [[+ or *][“] <file name> [“]]

Compiles and runs a program.

SAVE [[+ or *][“] <file name> [“]]

Saves the program in memory.

MACHINE SPECIFIC COMMANDS LIST

CALL <number> R, C

Executes a machine language subroutine starting at the specified address.

CALL <line number> or <label> R

Executes a machine language subroutine located at the specified <line number> or <label> e. g.:

80 CALL LINE 100

90 CALL LINE "LABEL"

100 MACHLG 34, 21, x%, 255, 9:RETURN

110 "LABEL" : MACHLG 34, 21, x%, 255, 9:RETURN

DEF USR <digit>=<expr> R, C

Sets the starting address of the user defined machine language subroutines (USR0 to USR9).

INP (<expr>) R

Allows to read the data present at the specified I/O port.

e. g.: X=INP(65535), X=0-255

LINE <line number> or <label> R, C

This function calculates the byte size in memory of a compiled line.

If it's used with CALL, it calls a machine language program line.

OUT <port>, <data> R

Outputs <data> to the specified <port>.

PEEK [WORD] or **LONG** (<expr>) R, C

Returns the content of the memory location addressed by <expr>.

POKE [WORD] or **LONG** <expr1>, <expr2> R, C

Writes the value of <expr2> into the memory location addressed by <expr1>.

MACHLG [bytes] - or - **MACHLG** [words] - or - **MACHLG** [variables] R

Inserts bytes directly into the compiled program.

USR <number> (<word expr>) R

User defined functions.

DISK ACCESS COMMANDS LIST

ERRMSG\$ <expr> R, C

Returns the disk error message string corresponding to the value of <expr>.

ERROR = <expr>

Sets the disk error number.

ERROR R

Returns the disk error number, zero means no errors found, this function is used in conjunction with **ON ERROR**. After a disk error occurred, **ERROR** must be set to zero.

CLOSE <#numeric expression1>, <#numeric expression2>, R

CLOSE closes the file(s) opened with **OPEN** whose number(s) is specified as arguments. If no number is given all files will be closed.

INPUT # <expr> , <variable> , <variable1> R

It reads data from a disk file (previously opened by **OPEN**) or other devices specified by the value of <expr> until a carriage return, “;” or an END OF FILE is encountered, or 255 characters are read.

e. g.: OPEN 0, 1, “NOME FILE” INPUT #, A\$

KILL <string> R, C

Deletes the disk file whose name is specified by the value of <string>.

e. g.: KILL “PROVA.COM”

LINEINPUT # <expr> , < string variable> R

This command works like **INPUT**, the advantage of **LINEINPUT** is its capacity to recognize more characters, like comma “;”, EOF. In addition it sets ERROR = End Of File.

LOC (<expr>) R

Returns the position pointer into the current record of the file specified by the value of <expr> .

e. g.: OPEN “R”, 1, “FILE” RECORD#1, 6, 8 PRINT LOC(1) prints 8.

LOF (<expr>) R

Returns the last valid record number for the file specified by the value of <expr>.

ON ERROR [GOSUB] <line number> or <label> or 65535 R

Disk errors management.

OPEN “I” or “O” or “R”, [#] <file number>, <file name> [, <record length>] R

Opens a file.

PRINT # <expr>, <list of values to print> R

Writes the list of values to a file or another device specified by <expr>.

- READ #** <file number >, <var> or < string>;<length> [, ...] R
Reads strings or numbers saved in compressed format by **WRITE#**.
- RECORD [#]** <file number>, <record number> [, <position in record>] R
Moves the file position pointer anywhere in a record.
- REC (<file number>)** R
Returns the file position pointer relative to a file whose number is specified by the value of <file number>.
- RENAME** <name1> **TO** <name2> R, C
Renames the disk file <name1> to the new name<name2>.
- ROUTE [#]** <expr> R
Redirects output to a device specified by the value of <expr>.
- RUN [<file number>]** R
Runs the program in memory. If <file number> is present it runs the program stored in the open file <file number>.
- WRITE#** <expr1>, <var> or <string>;<length> [, ..] R
Writes the content of variables and strings to a file.

INSTRUCTIONS LIST

CLEAR R, C

Sets all variables to zero or to a null value.

CLEAR <number> Sets aside <number> bytes for INDEX\$.

CLEAR END Clears all variables not being used.

CLEAR INDEX\$ Sets all elements of INDEX\$ array to empty strings.

DATA <list of constants>, ,..... R

It stores string and/or numeric comma-separated constants that can be stored into variables by the **READ** instruction. **DATA** instructions may appear anywhere in the program, the constants are read in the order they are written in the program lines.

DEF R

DEFINT <numeric variable>, , :Defines as % integer the listed variable(s).

DEFSNG <variabile numerica>, , :Defines as ! single precision the listed variable(s).

DEFDBL <variabile numerica>, , :Defines as # double precision the listed variable(s).

DEFSTR <variabile numerica>, , :Defines as \$ string the listed variable(s).

DEF FN <variable> = <expression>

Sets a function, specified by <expression>, to be executed by **FN** <variable> .

DEF LEN <number> R

Sets the default length of strings, ranging from 1 to 255 .

DELAY <expr> R, C

Produces a delay of <expr> milliseconds, the precision of timing depends on the CPU clock.

DIM R

Preserves an area of memory to store declared strings and arrays, it guarantees enough space in memory to satisfy the demand of room in every variable declaration. **DIM** automatically initializes the variables declared in it.

e. g.: DIM A\$(30),Q(100),Z(5,2), DIM X7(X,Y),X8(X,X,X), DIM C\$(100*3)

DO UNTIL R

DO instruction determines the entry point of a loop that will be executed until the expression after **UNTIL** is TRUE.

ELSE <line number> or <label> R, C

ELSE <instruction>

Used in conjunction with the **IF** instructions,executes the code following it whether the condition after **IF** is FALSE.

END R

Stops the program and returns to the operating system.

END FN = <expr> R

Used to close the **LONG FN** instruction. <expr> can be any numeric type expression if long expression is numeric and **MUST** be a string if type of long expression is string.

END IF R

See **LONG IF**.

FN <name> (<expr1>, <expr2>,) R

Like **FN** instruction, in addition this syntax may pass variables to the subroutines defined using **DEF FN** or **LONG FN**.

e. g.: **LONG FN** Prova\$ (x\$)**HELP** command

END FN = x\$ => **FN** Prova\$ (A\$)

FOR <variable> = <expr1> **TO** <expr2> **STEP** <expr3>...**NEXT** <variable> R

First <variable> is initialized to <expr1>, then starts a loop that repeats up to when the value of <variable> is <expr2>, **STEP** determines the increment of <variable>'s value, if omitted increment is 1. The loop is not executed if <expr1> is greater than <expr2> and <expr3> is negative.

GOSUB <line number> or <label> R

Jumps to the specified line, returns to the calling point when a **RETURN** instruction is encountered.

GOTO <line number> or <label> R

Jumps to the specified line.

IF <logical expression> **THEN** <instruction> R, C

IF <expression > **THEN** <instruction> **ELSE** <instruction>

When <logical expression> is TRUE <instruction> after **THEN** is executed, otherwise if an **ELSE** clause is specified <instruction> after it is executed. If no **ELSE** is present and <logical expression> is FALSE the next instruction after **IF** is executed (sequential order).

Writing a line number or a label after **THEN** or **ELSE** will cause a jump to that location if proper conditions occur.

INDEX\$ (<expr1>) = <string1>

INDEX\$I (<expr2>) = <string2>

INDEX\$D (<expr3>)

Replaces <string1> into the existing string specified by <expr1>, inserts <string2> in the position specified by <expr2>; deletes the string specified by <expr3>.

LET <variable> = <expression> R

Assigns the value of <expression> to <variable>. Keyword **LET** is always optional.

e. g.: **LET** A=100 => A=100 **LET** A\$="Prova ciao" => A\$="Prova ciao"

LONG FN <function name > [(var [, var [, ...]])] . . **END FN** [= expr] R

Like **DEF FN**, allows to define functions across more than one line.

LONG IF . . [**XELSE**] . . **ENDIF** R

Allows to structure the IF - THEN - ELSE construct across more than one line.

NEXT R

See **FOR**.

ON <expr> GOSUB <line number> [, <line number> [, ...]] Calls one of the subroutines listed according to the value of <expr>.	R
ON <expr> GOTO <line number> [, <line number> [, ...]] Jumps to the specified position according to the value of <expr>.	R
PSTR\$ (<var>) = <constant string> Store the address of <constant string> in the variable <var>.	R
RANDOM[IZE] [<expr>] Regenerates the random numbers seed.	R
READ [<var> or PSTR\$ (<var>) [, ...]] Reads constants strings and/or numeric values listed in DATA instructions.	R
REM [<string>] Allows to insert a comment.	R, C
RESTORE [<expr>] Positions the data item pointer to the <expr> position or to the top of the list if <expr> omitted.	R
RETURN [<line number>] Continues execution after the last GOSUB or ON GOSUB instruction.	R
STEP See FOR .	R
STOP Stops execution and prints the number of the last executed line.	R
SWAP var1, var2 Swaps the values of var1 and var2.	R
TROFF Disables instruction tracing.	R, C
TRON B or S or X Enables instruction tracing.	R, C
UNTIL See DO .	R
USR <number> (<expr>) See DEF USR .	R
WEND See WHILE .	R

WHILE <expr> . . **WEND**

Loop repeated until <expr> is TRUE.

R

XELSE

See **LONG IF**.

R

NUMERIC FUNCTIONS LIST

- ABS**(<numeric expression>) R, C
 Returns the absolute value of <numeric expression>.
 e. g.: ABS(3)=3, ABS(-3)=3, ABS(0)=0
- ASC**(<string constant, string variable, substring>) R, C
 Returns the ASCII code of the first character of argument string.
 e. g.: ASC("B")=66, ASC("CLUNK")=67 (note: **ASC** is the inverse function of **CHR\$**).
- ATN**(<numeric expression >) R, C
 Returns an approximation of arctangent of <numeric expression> expressed in radians.
 e. g.: ATN(5)=1.3734007, ATN(1.7)=1.0390722
- COS**(<numeric expression >) R, C
 Return the cosin of <numeric expression > expressed in radians.
 e. g.: COS(0)=1, COS(3.1415926/2)=0
- CVB** <string> R, C
 Returns the binary floating point value of the first characters stored in <string>.
- CVI** <string> R, C
 Returns the binary floating point value of the first two characters stored in <string>.
- EXP**(<numeric expression >) R, C
 Returns the exponential of base E of the value of <numeric expression >
 e. g.:EXP(0)=1, EXP(2)=7.3890562, EXP(-2.3025851)=.1, EXP(1)=2.7182817
- FIX** <expression> R, C
 Truncates all fractionary digits of <expression>.
 e. g.: PRINT FIX (123.456)=123
- FN** <name> (<expression1>, <expression2>,) R
 Calls a named expression previously defined by **DEF FN** or **LONG FN**.
 e. g.: DEF FN A#= SQR(4) => PRINT FN A# =2
- FRAC** (<expression>) R, C
 Truncates all integer digits of <expression>.
 e. g.: FRAC (123.456) = .456
- INDEXF** (<string>) R
INDEXF (<string>, <expression>)
 Return the index number of <string>, if <expression> is present starts the search for <string> from element <expression>.
 e. g.: INDEX\$ (0) = "CIAO", INDEX\$ (1) = "CIAO1", INDEX\$ (2) = "CIAO2" INDEXF ("CIAO")=0, INDEXF ("CIAO1",2)=-1

INT (<numeric expression>)	R, C
Returns the greatest integer lower or equal than the value of <numeric expression>. e. g.: INT(3)=3, INT(3.9)=3, INT(-3.5)=-4	
LOG (<expr>)	R, C
Returns the natural logarithm of <expr>. It's the inverse function of EXP .	
MAYBE	R, C
Returns TRUE (-1) or FALSE (0) with equal probability.	
MEM	R
Returns the number of bytes available for the INDEX\$ array.	
PAGE [[<expr1>][, [<expr2>][, [<expr3>]]]]	R, C
Formats printer output.	
POS (<expr>)	R, C
Returns the horizontal cursor position for the device specified by <expr>.	
RND (<expr>)	R, C
Returns a random integer number ranging between 1 and <expr>.	
SGN (<expr>)	R, C
Returns the sign of an expression.	
SIN (<expr>)	R, C
Returns the trigonometric sin of <expr> in radians.	
SQR (<expr>)	R, C
Returns the square root of <expr>.	
TAN (<expr>)	R, C
Returns the trigonometric tangent of <expr> in radians.	
VAL (<string>)	R, C
Returns the value of the figure contained in <string>.	
VARPTR (<var>)	R
Returns the address of <var>.	

CURSOR POSITIONING INSTRUCTIONS LIST

LOCATE <exprx>, <expy>, [<exprcursor>]

R

Positions the cursor at the coordinates specified by <exprx> and <expy>. Optionally can turn on or off the cursor. (zero=off, non-zero=on).