CBZ
80

COMPILATORE BASIC Z80

GUIDA RAPIDA

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

CBZ-80 Edizione 3.01 *Rel. 15 Giugno 1999*

CBZ 80

COMPILATORE BASIC Z80

GUIDA RAPIDA

CBZ-80 è un potente tool software che consente la programmazione ad alto livello (BASIC), su tutte le schede basate sulla famiglia Z80 Zilog. È provvisto di un completo supporto al Floating Point con precisione regolabile fino a 54 cifre significative, con trattazione anche delle funzioni trigonometriche e trascendentali. L'ambiente di sviluppo è estremamente amichevole e l'impiego di tempo per la generazione degli applicativi risulta notevolmente ridotto. Tale BASIC viene eseguito da eprom e genera un codice che, in abbinamento al **GDOS®**, viene eseguito dall'EEPROM o EPROM parallela di bordo; si riduce così la necessità di hardware esterno (in circuit emulator, EPROM programmer, etc.) e allo stesso tempo si velocizza la fase di debug del programma applicativo. La produttività del codice e l'immeditatezza con cui è possibile intervenire sull'hardware rendono questo BASIC un impagabile strumento professionale ad ogni livello.

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

CBZ-80 Edizione 3.01 *Rel. 15 Giugno 1999*

Vincoli sulla documentazione **grifo**® Tutti i Diritti Riservati

IMPORTANTE

Tutte le informazioni contenute nel presente manuale sono state accuratamente verificate, ciononostante **grifo**® non si assume nessuna responsabilità per danni, diretti o indiretti, a cose e/o persone derivanti da errori, omissioni o dall'uso del presente manuale, del software o dell' hardware ad esso associato.
grifo® altresi si riserva il diritto di modificare il contenuto e la veste di questo manuale senza alcun preavviso, con l' intento di offrire un prodotto sempre migliore, senza che questo rappresenti un obbligo per **grifo**®.
Per le informazioni specifiche dei componenti utilizzati sui nostri prodotti, l'utente deve fare riferimento agli specifici Data Book delle case costruttrici o delle seconde sorgenti.

LEGENDA SIMBOLI

Nel presente manuale possono comparire i seguenti simboli:

Attenzione: Pericolo generico

Attenzione: Pericolo di alta tensione

Marchi Registrati

INDICE GENERALE

INDICE DELLE FIGURE

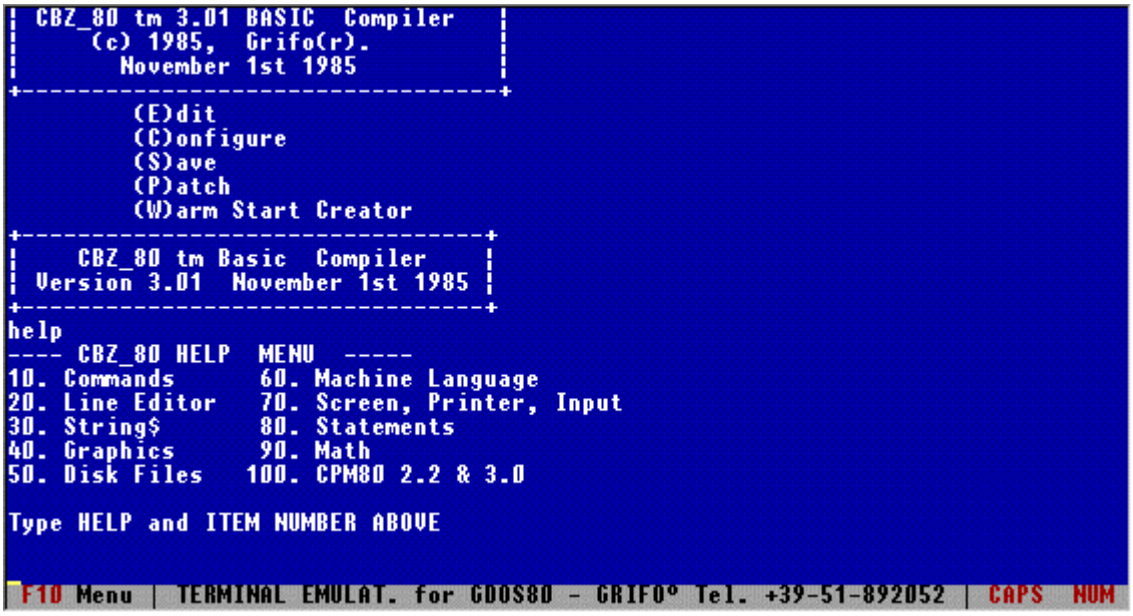

FIGURA 1: SCHERMATA INIZIALE DELL'HELP IN LINEA

GUIDA RAPIDA AL CBZ-80

Questa guida rapida al linguaggio di programmazione **CBZ-80** elenca le parole chiave dando per ciascuna una breve descrizione di uso e funzione.

Il **CBZ 80** è un potente pacchetto di sviluppo software che consente una programmazione ad alto livello in BASIC, per tutte le schede del carteggio **grifo®** basate sulla famiglia di microprocessori Z80. Il codice generato richiede l'ausilio del sistema operativo romato **GDOS 80** di cui il **CBZ 80** sfrutta le funzioni e le modalità di funzionamento. L'ambiente di sviluppo è estremamente amichevole e, rispettando le normali caratteristiche dei BASIC, riduce notevolmente il tempo di generazione degli applicativi. Gli utenti che non conoscono il BASIC potranno usufruire delle sue intuitive e numerose istruzioni e comandi diventando operativi in poche ore di lavoro, mentre i programmatori esperti non avranno bisogno di alcun tempo di apprendimento. In ogni caso l'efficienza del codice e l'immeditatezza con cui è possibile intervenire sull'hardware, rendono il **CBZ 80** un impagabile strumento professionale ad ogni livello.
Il compilatore supporta le applicazioni matematiche, quelle di controllo, la gestione di data base, l'interfacciamento con una console generica, le chiamate al sistema operativo e tante altre funzioni adatte a risolvere i problemi di automazione industriale. La sua libertà consente di scrivere sia programmi strutturati che non, raggiungendo livelli di efficienza e flessibilità difficilmente disponibili in altri pacchetti di sviluppo di pari prezzo.

GENERALITÁ

Il **CBZ 80** è un ambiente di sviluppo e programmazione composto da una serie di sottogruppi indipendenti che interagiscono nei confronti dell'utente che puo' quindi decidere se utilizzarli o meno. Volendolo paragonare ad ambienti di programmazione BASIC universalmente conosciuti, il **CBZ 80** ha una modalità di sviluppo simile a quella del GWBASIC ed un elenco di istruzioni equiparabile a quello del QUICK BASIC.
Con questo pacchetto si ha la possibilità di sfruttare tutte le risorse hardware della scheda prescelta, direttamente con le istruzioni ed i comandi ad alto livello, senza doversi preoccupare di sviluppare firmware di gestione specifico. Ad esempio il **CBZ 80** si occupa della gestione di risorse hardware come le linee seriali, le stampanti, le memorie di massa, le interfaccie operatore, ecc.
Il pacchetto **CBZ 80** è composto da software su dischetti, da un ricco manuale d'uso e da una serie di esempi (in formato sorgente ed eseguibile) che illustrano come gestire le risorse della scheda di controllo.

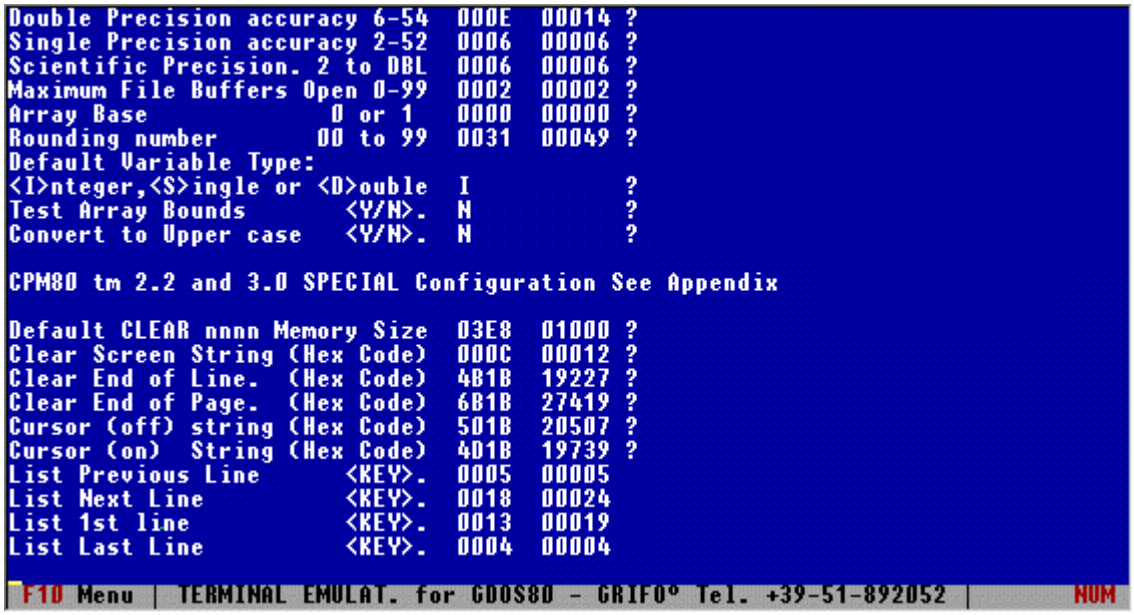

FIGURA 2: REQUESTER PER LA CONFIGURAZIONE DEI PARAMETRI INTERNI

CARATTERISTICHE DEL CBZ 80

Fondamentalmente il **CBZ 80** include due modalità principali: quella di configurazione e quella di sviluppo vero e proprio del programma applicativo. Le caratteristiche fondamentali di queste modalità, vengono di seguito descritte.

- Configurazione: è una modalità di settaggio di alcuni parametri del compilatore che riguardano direttamente o indirettamente il codice generato (la figura 2 mostra la schermata del requester di configurazione):
  - Precisione delle variabili in floating point (da 6 a 54 cifre).
  - Numero massimo di files contemporaneamente aperti (da 0 a 99).
  - Tipo di numerazione degli indici dei vettori e delle matrici.
  - Soglia di approssimazione delle variabili reali.
  - Tipo delle variabili non dichiarate.
  - Abilitazione della verifica di validita' sugli indici dei vettori.
  - Conversione automatica in maiuscolo.
  - Dimensioni della memoria indicizzata.
  - Personalizzazione dei tasti associati alle funzioni di editor piu' frequentemente usate.
  - Configurazione delle sequenze di comando per la console. Normalmente il **CBZ 80** viene fornito con questi parametri configurati per lo standard ADDS VIEWPOINT, utilizzato dal **GET 80** e da tutte le interfaccie operatore **QTP xxx**.
  - Area di memoria dedicata agli eventuali chain.
- Sviluppo: è la modalità normalmente utilizzata dall'utente finale e comprende un editor, il compilatore e l'ambiente di debug per il programma applicativo da sviluppare. L'uso di questa modalità è comune a tutti i linguaggi di programmazione e prevede:

  1) la stesura o la correzione del programma applicativo (questa fase puo' essere realizzata con l'editor integrato o un normale editor ASCII esterno, come ad esempio quello del **GET 80**);

2) il caricamento del programma applicativo usando il file system del **GDOS 80**;
3) la compilazione del programma caricato, da cui si ottiene un codice eseguibile. Nel caso in cui la compilazione individui degli errori sintattici si ritorna al punto 1;
4) l'esecuzione del codice ottenuto direttamente a bordo della scheda di controllo. Se dalla verifica funzionale del programma applicativo emergono dei problemi si ritorna al punto 1;
5) la ricompilazione finale che fornisce un programma direttamente gestibile dal **GDOS 80** (ad esempio per la programmazione in EPROM o FLASH EPROM).

Tra le caratteristiche dell'ambiente di sviluppo appena descritto, si ricorda:

- Sorgente BASIC con o senza numeri di linea; quando i numeri di linea non sono utilizzati, vengono rimpiazzati da etichette.
- Sintassi convenzionale, che consente di riutilizzare codice già sviluppato e provato con altri linguaggi BASIC.
- Quattro diversi tipi di variabili: intere, reali a singola precisione, reali a doppia precisione e stringhe.
- Ricca serie di operatori che comprendono quelli matematici, relazionali, logici e di shift.
- Completa lista di funzioni matematiche che includono anche le funzioni trigonometriche e trascendentali.
- Supporto delle piu' diffuse basi di numerazione (decimale, esadecimale, ottale e binario).
- Serie di istruzioni per la gestione di una console ad alto livello (posizionamento cursore, cancellazione schermo totale e parziale, verifica tasti premuti, inserimento di dati, ecc.). Tali istruzioni consentono di gestire direttamente tutti i terminali della serie **QTP xxx**.
- Ricca serie di istruzioni per la gestione del file system del **GDOS 80**. Grazie a queste istruzioni l'utente non deve piu' gestire le aree di memoria a basso livello tramite complicati indirizzi e mappaggi, bensi' comodamente tramite file dati che possono essere creati, ampliati, cancellati, rinominati, copiati, prelevati, ecc.
- Interessante insieme di funzioni e procedure per la gestione delle stringhe (concatenamento, conversione, frammentazione, ricerca, elaborazione, ecc.).
- Gestione di una area di memoria in modalità indicizzata gestibile a livello di puntatori.
- Completo e potente elenco di istruzioni di controllo che consentono di eseguire cicli, effettuare verifiche singole o multiple, creare funzioni e procedure, lanciare altri programmi, ecc.
- Basilari istruzioni per la gestione a basso livello delle risorse hardware della scheda, come istruzioni di I/O, gestione diretta della memoria, codice in linguaggio macchina, chiamata assoluta a procedure esterne, ecc.
- Istruzioni ad alto livello che tramite il sistema operativo **GDOS 80** gestiscono periferiche hardware come le linee seriali ed una stampante.
- Diverse modalità di compilazione che consentono di ottimizzare il tempo di sviluppo ed il codice generato.
- Gestione completa della tecnica del chain che consente di poter eseguire in sequenza diversi programmi applicativi che si possono scambiare parametri di lavoro. Grazie a questa possibilità sono facilmente affrontabili anche grossi problemi di automazione, caratterizzati da grosse quantità di dati e di codice.

- Comodo aiuto in linea che descrive tutte le parti del **CBZ 80**, con un notevole risparmio di tempo.
- Nessuna licenza o costo aggiuntivo. L'utente è libero di realizzare tutte le applicazioni che desisdera senza nemmeno informare la **grifo®**.

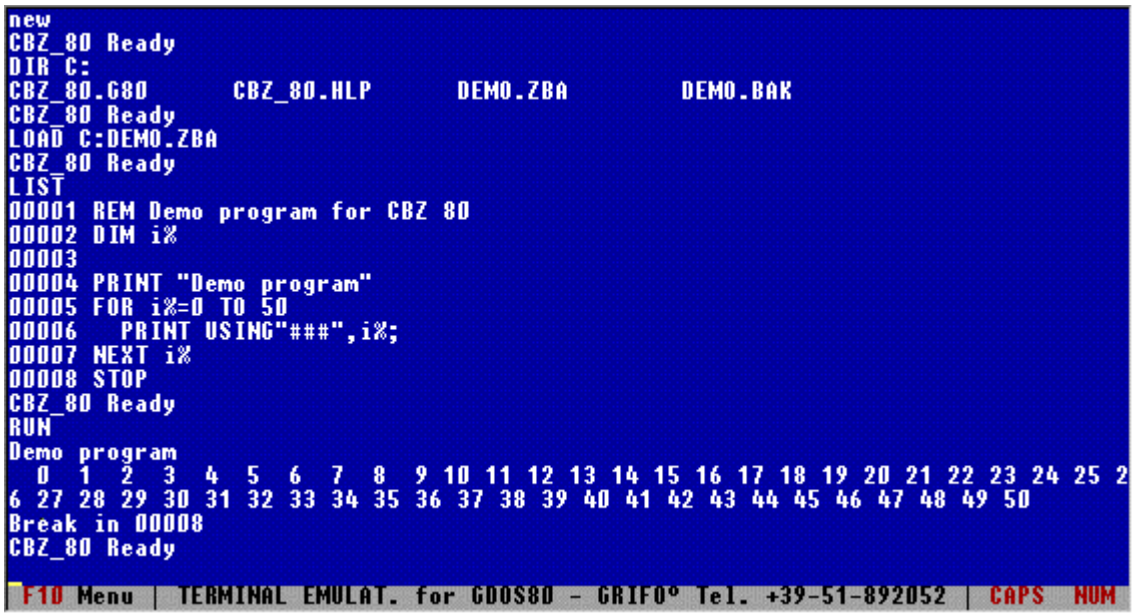

FIGURA 3: L'EDITOR A LINEA DI COMANDO

REQUISITI DEL CBZ 80

In termini operativi il **CBZ 80** richiede solamente tre elementi separati che vengono di seguito elencati.

- Una scheda di controllo basata sullo Z80 come:
  - **GPC® 80F**
  - **GPC® 81F**
  - **GPC® 011**
  - **GPC® 15A**
  - **GPC® 15R**
  - **GPC® 153**
  - **GPC® 154**
  - **GPC® 150**
- Il sistema operativo **GDOS 80** per la scheda di controllo utilizzata;
- Un personal computer collegato in seriale, alla scheda di controllo.

CONVENZIONI

La guida usa le seguenti convenzoni tipografiche:

PAROLE CHIAVE (grassetto maiuscolo)
Parola chiave di CBZ-80.

<*riferimento*> (racchiusi in parentesi circonflesse)
Variabili, espressioni, costanti o altre informazioni che vanno sostituite a seconda delle esigenze.

[*valori opzionali*] (racchiusi in parentesi quadre)
Valori non obbligatoriamente richiesti.

elementi ripetitivi . . . (seguiti da tre puntini)
Bisogna inserire altri parametri con la stessa forma.

TIPI DI DATI E LORO LIMITI

Viene di seguito fornito un elenco dei tipi di dati posseduti dal **CBZ-80** con a fianco specificato il campo di valori che possono assumere e la loro occupazione in memoria.

Tipo	Valore Minimo	Valore Massimo	Memoria occupata (bytes)
INTERO	-32768	+32767	2
REALE (BCD)	-9.999E+63	+9.999E+63 (Da 6 a 54 cifre)	Da 2 a 28
ESADECIMALE	&H0000	&HFFFF	2
OTTALE	&O000000	&O377777	2
BINARIO	&Xbbbbbbbbbbbbbbbb		2
STRINGA	Da 0 a 256 caratteri (vedere **DEF LEN**)		Da 0 a 256

Gli errori di overflow vengono segnalati solo per il tipo BCD.

Per dichiarare implicitamente il tipo di una variabile è sufficiente far seguire al suo nome un carattere opportuno, es. **DIM** B% dichiara B come variabile intera.

Tipo	Carattere
INTERO	%
REALE (singola precisione)	!
REALE (doppia precisione)	#
STRINGA	$

ELENCO OPERATORI

C = command mode
R = run mode

^ oppure **[** R, C
Elevamento a potenza, solo per numeri reali.

\+ - * / \ R, C
Somma, sottrazione, moltiplicazione, divisione e divisione tra numeri reali.

< = > <= oppure =< >= oppure => <> R, C
Operatori condizionali. Zero significa falso, uno significa vero.

<espressione numerica1> **AND** <espressione numerica2> R, C
Se le due espressioni sono vere (non-zero), il risutato è vero, dove vero=1 e falso=0.
Anche usato per confrontare dei bit con dei numeri binari.

<espr1> **MOD** <espr2> R, C
Restituisce il resto della divisione intera di <espr1> ed <espr2> col segno di <espr1>.

NOT <espr> R, C
Restituisce il valore opposto di <espr>.

<espr1> **OR** <espr2> R, C
Esegue l'OR logico o l'OR binario.

<espr1> **XOR** <espr2> R, C
Esegue lo XOR logico o lo XOR binario.

<espr1> **>>** <espr2> R, C
Esegue lo scorrimento a destra dei bit di <espr1> di <espr2> posizioni.

<espr1> **<<** <espr2> R, C
Esegue lo scorrimento a sinistra dei bit di <espr1> di <espr2> posizioni.

ELENCO FUNZIONI PER ELABORAZIONE STRINGHE

ASC(<costante, variabile stringa, o riferimento sottostringa>) R, C
Fornisce il valore decimale corrispondente, utilizzando la tabella ASCII, al primo carattere contenuto nell'argomento.
Es.: ASC("B")=66, ASC("CLUNK")=67 (da notare, che ASC, e` l'inverso della funzione CHR$).

BIN$(<espressione numerica>) R, C
Calcola il valore binario dell' espressione numerica. Es: BIN$(255)=0000000011111111
Viceversa &X0000000011111111=255

CHR$(<espressione numerica>) R, C
Fornisce il carattere corrispondente, secondo la tabella ASCII, al valore dell'espressione numerica, l' argomento deve essere un intero compreso tra 0 e 255.
Es.: CHR$(65)="A", CHR$(97)="a", CHR$(32)=" " (spazio)

DATE$ R, C
Restituisce la data nel formato mese/giorno/anno.

HEX$ (espressione) R, C
Converte il valore dell' espressione in un numero in esadecimale.
Es.: HEX$ (255)=FF , HEX$ (170)=AA ,HEX$ (85)=55

INDEX$ (<stringa>) R
INDEX$ (<stringa>, <espressione>)
Restituisce il numero associato alla stringa definita, se viene messa l' espressione dopo la stringa definita, verrà restituito un valore che indica la distanza che intercorre fra il valore e la stringa definita. Es.: INDEX$ (0) = "CIAO", INDEX$ (1) = "CIAO1" ,INDEX$ (2) = "CIAO2"
INDEXF ("CIAO")=0, INDEXF ("CIAO1",2)=-1

INSTR (<espressione>, <stringa1>, <string2>) R
Restituisce un valore ricavato dalla ricerca della stringa2 sulla stringa1, il valore rappresenta la posizione del cursore nella stringa, l' epressione decide la posizione di partenza della ricerca.
Es.: A$="Prova ciao"B$="ciao"INSTR (1, A$, B$)=7

LEN (<stringa>) R, C
Restituisce la lunghezza della stringa definita.
Es.: A$="Prova ciao"PRINT LEN (A$)= 10

LEFT$ (<stringa>, <espressione>) R
Permette di selezionare un numero di caratteri, della stringa scelta, definiti dall' epressione.
Es.: A$="Prova ciao"PRINT LEFT$ (A$,3)="Pro"PRINT LEFT$ (A$, 5)="Prova"

MID$ (<stringa>, <espr1>, <espr2>) R, C
Restituisce il sottinsieme di <stringa> che inizia a <espr1> ed è lungo <espr2> caratteri.

MKB$ (<espr>) R, C
Restituisce una stringa contenente il valore in virgola mobile compresso di una espressione BCD CBZ-80.

MKI$ (<espr>) R, C
Restituisce una stringa di due caratteri specificata dall'intero a due bytes <espr>.

OCT$ (<espr>) R, C
Restituisce in una stringa il valore ottale (base 8) della parte intera di <espr>.

PSTR$ (<var>) R
Restituisce la stringa il cui indirizzo è memorizzato nella variabile <var>.

RIGHT$ (<stringa>, <espr>) R, C
Restituisce gli ultimi <espr> caratteri di <stringa>.

SPACE$ (<espr>) R, C
Restituisce una stringa di <espr> spazi.

STR$ (<espr>) R, C
Converte il valore numerico <espr> in una stringa.

STRING$ (<espr1>, <stringa> oppure <espr2>) R, C
Crea una stringa lunga <espr1> ripetendo il primo carattere di <stringa> oppure l'ASCII di <espr2>.

TIME$
Restituisce l'orario nel formato ora:minuti:secondi.

UCASE$ (<stringa>) R, C
Converte <stringa> in maiuscolo.

UNS$ (<espr>) R, C
Converte <espr> in stringa in formato decimale senza segno.

ELENCO ISTRUZIONI DI INPUT/OUTPUT

CLS R, C
Pulisce tutto lo schermo, e posiziona il cursore in alto a sinistra
CLS <espressione numerica>: Riempie lo schermo con caratteri ASCII specificato dall' espressione .
CLS LINE : Cancella la linea dove è situato il cursore .
CLS PAGE : Cancella la pagina dalla posizione del cursore fino alla fine dello schermo .

DEF TAB =<numero> R
Definisce il numero di caratteri (spazi) fra le tabulazioni dei comandi PRINT , PRINT# or LPRINT , da 1 a 255 (default 16).

I**NPUT** [@ (espres.^{x},espres.^{y})] [;] [!] [&espres.] ["stringa";] <variabile> [, <var.> .....]
R

@ (espres.^{x},espres.^{y})	Opzionale, permette di posizionare il cursore alle coordinate x, y.
;	Opzionale, sopprime il carriage return/line feed.
!	Opzionale, abilita il carriage return automatico dopo un numero massimo di caratteri inseriti.
&<espr>	Opzionale, abilita il carriage return automatico dopo <espr> caratteri inseriti.
"stringa"	Opzionale, stampa la domanda o l'informazione legata all'input.
<variabile>	Deve essere una variabile intera, singola o doppia densità o stringa.

L' INPUT è usato per inserire valori (numerici o stringa) dalla tastiera all' interno delle variabili.

INKEY$ R
Restituisce il corrispondente codice ASCII del tasto premuto, se nessun tasto è premuto la stringa associata rimarrà vuota.
Es.: Premi "A" ,A$=INKEY$, A$="A"

LINEINPUT [@ (espres.x,espres.y)] [;] [!] [&espres.] ["stringa";] <variabile stringa>
R
A differenza dell INPUT , LINEINPUT può inserire caratteri da tastiera in una variabile esclusivamente di tipo stringa, i vantaggi sono che può accettare tutti i caratteri ASCII.

LPRINT [variabili, costanti, ...] R, C
Scrive il valore dei suoi parametri sulla stampante. Per usarlo nell'editor di riga farlo precedere da un carattere di due punti (:LPRINT).

PAGE R, C
Restituisce la linea corrente sulla stampante.

PRINT [{ @ oppure % } (<espr1>, <espr2>)] **USING** <stringa formato> ; <espr> ;
[**USING** ...][<lista di valori da stampare>] R, C
Scrive informazioni sul dispositivo corrente, normalmente il video.
@ e % con <espr1> e <espr2> permettono di posizionare il cursore sullo schermo, **USING** <stringa formato> permette di formattare il testo.

SPC (<espr>) R, C
Stampa <espr> spazi.

TAB (<espr>) R
Muove il cursore alla posizione <espr>.

WIDTH [**LPRINT**] [=] <espr>
Imposta la larghezza di stampa per **PRINT** o **LPRINT**.

ELENCO COMANDI

APPEND <numero linea> <nome file>
Inserisce una parte di programma o un subroutine ASCII (senza numeri riga) a partire dalla linea impostata.Es: APPEND 1000 PROGRAM.ASC

APPEND* <numero linea> <nome file>
Permette anche di eliminare dal file ASCII i REM e gli spazi, liberando la memoria da questi.

AUTO
Attiva l' auto numerazione del programma da editare.
AUTO <numero> Attiva l' auto numerazione del programma a partire dal numero specificato.
AUTO <numero>,<incremento> Attiva l' auto numerazione del programma a partire dal numero specificato, con un passo pari all' incremento.
AUTO ,<incremento> Attiva l' auto numerazione del programma con un passo pari all' incremento.

DELETE <numero linea> - <numero linea>
Tutte le linee di programma comprese nell'intervallo sono cancellate dal programma corrente. Il secondo numero linea specificato nell'intervallo, deve essere maggiore del primo.

DIR <drive>
Visualizza la directory del disk drive specificato .

EDIT <line>
E <line>
Viene usato per correggere una linea specificata, tramite i sotto comandi, sotto riportati, è possibile operare le opportune modifiche in modo molto semplice e veloce .
I-Dalla posizione del cursore si procede nel modo insert, si esce con <ESC>.
X-Va alla fine della riga e passa in modo insert.
<n>D-Cancella <n> caratteri a partire dalla posizione del cursore.
<n>C<key>-Cambia il carattere posizionato dal cursore con <key> per <n> volte.
H-Cancella tutto dalla posizione del cursore fino alla fine e passa in modo insert.
<n>S<key>-Posiziona il cursore <n> all' ennesimo tasto specificato <key> .
L-Visualizza tutta la riga prescelta, il cursore si porta a capo per continuare l' EDIT.
A-Tasto di abort, ripristina tutte le modifiche.
<n>K<key>-Cancella dalla posizione <n> dell' ennesimo tasto specificato <key> .
<n> <SP>-Muove il cusore di <n> caratteri verso destra.
<n> <BS>-Muove il cusore di <n> caratteri verso sinistra.
<ESC>-Esce dal modo insert e attende un nuovo comando.
<ENTER>-Esce dal EDIT e visualizza la riga con i cambiamenti riportati.
<BREAK>-Tasto di abort, esce dall' EDIT e ripristina tutte le modifiche.

FIND <stringa>
FIND #<stringa>
Permette di localizzare una zona del testo del programma, molto utile in abbinamento al comando EDIT, per cercare la prossima stringa definita premere ";" o FIND <ENTER>.
Il carattere "#" permette di rintracciare una chiamata di una linea di programma.
Es.: FIND #1234508000 GOTO 12345

HELP
HELP <numero>
Richiama il menu HELP stampandolo sullo schermo, premere lo spazio per continuare, per uscire bisogna arrivare alla fine dell' HELP.
Se viene specificato un numero verrà richiamato solo quel numero di HELP.

L[**IST**] [+][*] <numero riga> oppure <label>
Usato all'interno dell'editor, serve per mostrare a schermo tutto o parte del programma in memoria.

LLIST
Stessa sintassi del comando **LIST**, invia l'output verso la stampante.

LOAD [*] ["] <nome file> ["]
Usato all'inteno dell'editor, serve per caricare in memoria un listato in formato ASCII o un programma compilato. Se non ci sono numeri di riga verranno aggiunti con incrementi di uno. LOAD * elimina i commenti e gli spazi superflui per risparmiare memoria. Es. : LOAD PROVA.ZBA

MEM[**ORY**]
In command mode, restituisce informazioni sull'utilizzo della memoria di sistema.

MERGE ["] <nome file> ["]
Fonde un programma presente su disco, e dotato di numeri di riga, col programma in memoria.

NEW
Cancella il programma in memoria.

QUIT
Esce dallo CBZ-80.

RENUM [new] [, [old]][, increment]
Ricalcola i numeri di riga.

RUN [[+ oppure *]["] <nome file> ["]]
Compila ed esegue un programma.

SAVE [[+ oppure *]["] <nome file> ["]]
Salva il programma in memoria.

ELENCO COMANDI SPECIFICI DELL'IMPLEMENTAZIONE

CALL <numero> R, C
Esegue una subroutine in linguaggio macchina all' indirizzo specificato

CALL <numero linea> oppure <label> R
Esegue una subroutine di una linea compilata oppure di una label. Es:
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>=<espressione> R, C
Definisce l' indirizzo superiore delle subroutine in linguaggio macchina (USR0 a USR9).

INP (<espressione>) R
Permette di leggere il valore scritto all' indirizzo di I/O specificato dall' espressione.
Es.: X=INP(65535), X=0-255

LINE <numero riga> oppure <label> R, C
Questa funzione permette calcolare il numero di byte usati in una linea di programma.
Se viene usato in abbinamento a CALL, permette di chiamare una linea in linguaggio macchina.

OUT <porta>, <dato> R
Invia il dato alla porta specificata.

PEEK [**WORD**] oppure **LONG** (<espr>) R, C
Restituisce il contenuto della locazione di memoria specificata da <espr>.

POKE [**WORD**] oppure **LONG** <espr1>, <espr2> R, C
Scrive il valore <espr2> nella locazione di memoria specificata da <espr1>.

MACHLG [bytes] - oppure - **MACHLG** [words] - oppure - **MACHLG** [variabili] R
Inserisce dei bytes direttamente nel programma compilato.

USR <cifra> (<espr word>) R
Funzioni matematiche definite dall'utente.

ELENCO COMANDI DI ACCESSO AL DISCO

ERRMSG$ <espressione> R, C
Restituisce la stringa di messaggio d' errore del disco, associato al numero specificato nell' espressione.

ERROR = <espr>
Imposta il numero di errore del disco.

ERROR R
Restituisce il numero della condizione di errore nel disco, lo zero rappresenta nessun errore incontrato, questa funzione viene usata assieme a ON ERROR. Dopo ogni errore da disco, l' ERROR deve essere settato a zero.

CLOSE <#espressione numerica1>,<#espressione numerica2>,..... R
Viene usato per chiudere i file aperti con OPEN, se non viene specificato nulla vengono chiusi tutti.

INPUT # <espressione> , <variabile> , <variabile1> ..... R
Permette di leggere dal disco (prima il file deve essere stato aperto con **OPEN**) o da altre periferiche specificate dall' espressione fino ad incontrare un carriage return, ",", 255 caratteri letti oppure un END OF FILE.
Es.: OPEN 0, 1, "NOME FILE" INPUT #, A$

KILL <stringa> R, C
Cancella sul disco il file specificato dalla stringa, può essere usato anche in modo diretto come un comando.Es.: KILL "PROVA.COM"

LINEINPUT # <espressione> , <variabile stringa> R
Viene usato come l' INPUT, il vantaggio del LINEINPUT è che riconosce dei caratteri in più, come la virgola ",", EOF e setta ERROR = End Of File ecc.

LOC (<espr>) R
Restituisce il puntatore di posizione entro il record corrente del file specificato dal numero <espr> . Es. : OPEN"R",1,"FILE" RECORD#1,6,8 PRINT LOC(1) stampa 8.

LOF (<espr>) R
Restituisce l'ultimo numero di record valido per il file specificato da <espr>.

ON ERROR [**GOSUB**] <numero linea> oppure <label> oppure 65535 R
Permette di gestire gli errori del disco.

OPEN "I" oppure "O" oppure "R", [#] <numero file>, <nome file> [, <lunghezza record] R
Apre un file.

PRINT # <espr>, <lista di valori da scrivere> R
Scrive informazioni su un file o altro dispositivo specificato da <espr>.

READ # <numero file>, <var> oppure <var stringa>;<lunghezza> [, ...] R
Legge stringhe o numeri salvati in formato compresso con **WRITE#**.

RECORD [**#**] <numero file>, <numero record> [, <posizione nel record>] R
Posiziona il puntatore del file in un punto qualunque del file.

REC (<numero file>) R
Restituisce il puntatore del file relativo al file specificato da <numero file>.

RENAME <nome1> **TO** <nome2> R, C
Rinomina il file <nome1> col nuovo nome <nome2>.

ROUTE [#] <espr> R
Redireziona l'output verso un dispositivo specifico.

RUN [<numero file>] R
Esegue il programma in memoria. Se è presente <numero file> esegue il programma dal file aperto.

WRITE# <espr1>, <var> oppure <stringa>;<lunghezza> [, ..] R
Scrive il valore di variabili e/o stringhe su un file.

ELENCO ISTRUZIONI

CLEAR R, C
Setta tutte le variabili a zero o nulle.
CLEAR <numero> Mette da parte <numero> bytes per l'array INDEX$.
CLEAR END : Setta tutte le variabili , non utilizzate ancora dal programma, a zero.
CLEAR INDEX$ Pulisce il contenuto dell'array INDEX$.

DATA <lista di costanti>, ... ,.... R
Le stringhe o le costanti numeriche, incluse nella lista, sono salvate come dati, ai quali si accede in ordine da programma (essi stessi fanno parte del programma) con la funzione **READ**.
Nella lista, ogni costante, è separata dalla successiva, da una virgola.

DEF R
DEFINT <variabile numerica>, ... , ... :Definisce la/le variabili di tipo % intere
DEFSNG <variabile numerica>, ... , ... :Definisce la/le variabili di tipo ! singole
DEFDBL <variabile numerica>, ... , ... :Definisce la/le variabili di tipo # doppie
DEFSTR <variabile numerica>, ... , ... :Definisce la/le variabili di tipo $ stringhe
DEF FN statement
DEF FN <variabile> = <espressione>
Definisce una funzione, rappresentata dall' espressione, che viene eseguita con FN <variabile> .

DEF LEN <numero> R
Definisce la lunghezza di default delle stringhe da 1 a 255 .

DELAY <espressione> R, C
Determina un ritardo nel programma definito in millisecondi , dipende dal clock della CPU .

DIM R
Riserva dell'area di memoria per dimensionare le stringhe e gli array dichiarati.
Un'istruzione DIM inizializza automaticamente le variabili dichiarate in essa. Tale istruzione, dopo che è stata eseguita, fa sì che la lunghezza di ogni stringa dichiarata in essa, corrisponda ad una relativa area di memoria, in grado di memorizzare un numero di caratteri, pari alla lunghezza della stringa dichiarata.
Es.: DIM A$(30),Q(100),Z(5,2), DIM X7(X,Y),X8(X,X,X), DIM C$(100*3)

DO ..... UNTIL R
Il do è usato per definire l' inizio di un loop che sarà eseguito fino a quando l' espressione definita dopo UNTIL non sarà vera .

ELSE <numero riga> o <label> R, C
ELSE <istruzione>
Viene usato in coppia con IF, permette, in caso che la condizione IF sia falsa, di eseguire ciò che viene dopo l'ELSE.

END R
Viene usato per fermare il programma e tornare dentro l'Operating System.

LONG FN ..... END FN = <espressione> R
Viene usato per chiudere l' istruzione LONG FN, dove l' espressione può essere opzionalmente numerica (#,%,&,!) e deve essere di tipo stringa ($).

END IF R
Vedere LONG IF.

FN <nome> (<espressione1>, <espressione2>, ...) R
Come FN ma in più può passare le variabili alla subroutine creata con DEF FN o LONG FN.
Es.: LONG FN Prova$ (x$) .....HELP command
END FN = x$ => FN Prova$ (A$)

FOR <variabile> = <espr1> **TO** <espr2> **STEP** <espr3>...**NEXT** <variabile> R
Qusto è un loop che si ripete fino a quando la variabile non raggiunge il valore di <espr2>, lo STEP determina l' incremento della variabile, se omesso lo STEP è 1.
Il corpo del loop, non viene eseguito se il valore iniziale è maggiore del limite ed il passo è negativo, inoltre se il valore iniziale è minore del limite ed il passo è negativo.

GOSUB <numero riga> oppure <label> R
Chiama una subroutine del programma, quindi salta alla riga specificata oppure alla label specificata e ritorna quando si incontra un RETURN.

GOTO <numero riga> oppure <label> R
Esegue un salto del programma alla riga specificata oppure alla label specificata.

IF <espressione logica> **THEN** <istruzione> R, C
IF <espressione logica> **THEN** <istruzione> **ELSE** <istruzione>
Quando l'espressione logica è vera, si esegue l'istruzione dopo il THEN, se invece risulta essere falsa, viene eseguita l'istruzione dopo l'ELSE.
In caso si ometta l'ELSE e la condizione risulta essere falsa, viene eseguita, l'istruzione successiva all'IF (in ordine sequenziale).
L'inserimento di un numero linea o label, dopo il THEN o l'ELSE, comporta un salto alla locazione specificata (equivale a GOTO <numero linea>).

INDEX$ (<espr1>) = <stringa1>
INDEX$I (<espr2>) = <stringa2>
INDEX$D (<espr3>)
Sostituisce <stringa1> con la stringa esistente localizzata da <espr1>, inserisce <stringa2> nella posizione specificata da <espr2>; cancella la stringa indirizzata da <espr3>.

LET <variabile> = <espressione> R
Assegna il valore dell' espressione alla variabile di qualsiasi tipo, questo tipo di operazione è valida anche se non viene richiamata la dichiarazione LET.
Es.: LET A=100 => A=100LET A$="Prova ciao" => A$="Prova ciao"

LONG FN <nome funzione> [(var [, var [, ...]])] . . **END FN** [= espr] R
Simile a DEF FN, permette di definire funzioni su più di una riga.

LONG IF . . [**XELSE**] . . **ENDIF** R
Permette di strutturare il costrutto IF - THEN - ELSE su più di una riga.

NEXT R
Vedere **FOR**.

ON <espr> **GOSUB** <numero linea> [, <numero linea> [, ...]] R
Chiama una delle varie subroutines a seconda del valore di <espr>.

ON <espr> **GOTO** <numero linea> [, <numero linea> [, ...]] R
Salta ad uno dei numeri linea a seconda del valore di <espr>.

PSTR$ (<var>) = <stringa costante> R
Memorizza l'indirizzo della stringa nella varibile <var>.

RANDOM[**IZE**] [<espr>] R
Rigenera il seme dei numeri casuali.

READ [<var> oppure **PSTR$**(<var>) [, ...]] R
Legge stringhe o numeri da un'istruzione **DATA**.

REM [<stringa>] R, C
Inserisce un commento.

RESTORE [<espr>] R
Posiziona il puntatore dei dati alla posizione <espr> oppure all'inizio della lista.

RETURN [<numero linea>] R
Prosegue l'esecuzione immediatamente dopo l'ultima istruzione **GOSUB** o **ON GOSUB**.

STEP R
Vedere **FOR**.

STOP R
Interrompe l'esecuzione e stampa il numero dell'ultima riga eseguita.

SWAP var1, var2 R
Scambia il valore di var1 e var2.

TROFF R, C
Disattiva il tracciamento delle istruzioni.

TRON B oppure **S** oppure **X** R, C
Attiva il tracciamento delle istruzioni.

UNTIL R
Vedere **DO**.

USR <cifra> (<espr>) R
Vedere **DEF USR**.

WEND R
Vedere **WHILE**.

WHILE <espr> . . **WEND** R
Ripetizione eseguita finchè <espr> è TRUE.

XELSE R
Vedere **LONG IF**.

ELENCO FUNZIONI NUMERICHE

ABS(<espressione numerica>) R, C
Fornisce il valore assoluto, dell'espressione numerica
Es: ABS(3)=3, ABS(-3)=3, ABS(0)=0

ASC(<costante stringa, variabile stringa, o riferimento sottostringa>) R, C
Fornisce il valore decimale corrispondente, utilizzando la tabella ASCII, al primo carattere contenuto nell'argomento.
Es.: ASC("B")=66, ASC("CLUNK")=67 (da notare, che ASC, e` l'inverso della funzione CHR$).

ATN(<espressione numerica>) R, C
Calcola una approssimazione dell'arcotangente, il valore di ritorno, è espresso in radianti.
Es.: ATN(5)=1.3734007, ATN(1.7)=1.0390722

COS(<espressione numerica>) R, C
Calcola il coseno trigonometrico del valore dell'espressione, che deve essere specificato in radianti.
Es.: COS(0)=1, COS(3.1415926/2)=0

CVB <stringa> R, C
Restituisce il valore binario floating point del primo carattere condensato nel numero <stringa>.

CVI <stringa> R, C
Restituisce il valore binario integer dei primo due caratteri condensati nel numero <stringa>.

EXP(<espressione numerica>) R, C
Fornisce l'esponenziale in base E, del valore dell'espressione:
EXP(0)=1, EXP(2)=7.3890562, EXP(-2.3025851)=.1, EXP(1)=2.7182817

FIX <espressione> R, C
Permette di eliminare tutti i decimali dopo la virgola.
ES.: PRINT FIX (123.456)=123

FN <nome> (<espressione1>, <espressione2>, ...) R
Permette di richiamare con il nome specificato, un espressione definita con DEF FN oppure LONG FN, dove questi ultimi devono presentarsi prima di essere chiamati da FN.
Es.: DEF FN A#= SQR(4) => PRINT FN A# =2

FRAC (<espressione>) R, C
Permette di eliminare la parte decimale dell' espressione.
Es.: FRAC (123.456) = .456

INDEXF (<stringa>) R
INDEXF (<stringa>, <espressione>)
Restituisce il numero associato alla stringa definita, se viene messa l' espressione dopo la stringa definita, la ricerca della stringa inizierà all'indice specificato dal valore di <espressione>. Es.:
INDEX$ (0) = "CIAO", INDEX$ (1) = "CIAO1" ,INDEX$ (2) = "CIAO2"
INDEXF ("CIAO")=0, INDEXF ("CIAO1",2)=-1

INT (<espressione numerica>) R, C
Fornisce il più grande valore intero, minore di o uguale al valore dell'argomento.
Es.: INT(3)=3, INT(3.9)=3, INT(-3.5)=-4

LOG (<espr>) R, C
Restituisce il logaritmo naturale di <espr>. È la funzione inversa di EXP.

MAYBE R, C
Restituisce TRUE (-1) o FALSE (0) con eguale probabilità.

MEM R
Restituisce il numero di bytes disponibili nell'array INDEX$.

PAGE [[<espr1>][, [<espr2>][, [<espr3>]]]] R, C
Formatta l'output verso la stampante.

POS (<espr>) R, C
Restituisce la posizione orizzontale del cursore per il dispositivo specificato da <espr>.

RND (<espr>) R, C
Restituisce un numero intero casuale compreso tra 1 ed <espr>.

SGN (<espr>) R, C
Restituisce il segno di un'espressione.

SIN (<espr>) R, C
Calcola il seno di <espr> radianti.

SQR (<espr>) R, C
Calcola la radice quadrata di <espr>.

TAN (<espr>) R, C
Calcola la tangente trigonometrica di <espr> radianti.

VAL (<stringa>) R, C
Converte una stringa in numero.

VARPTR (<var>) R
Restituisce l'indirizzo di una variabile.

ELENCO ISTRUZIONI POSIZIONAMENTO CURSORE

LOCATE <esprx>, <espry>, [<esprcursore>] R
Posiziona il cursore alle coordinate specificate da <esprx> e <espry>. Opzionalmente accende o spegne il cursore (zero=spento, non zero=acceso).