# GBASIC 7.40

***Sistema operativo / interprete GBASIC per microntrollori basati su processori ad 8 bit Z-80 e derivati***

# *Manuale Generale*

Rev. A
1993

*di Gianni Becattini*

# Presentazione

## della precedente edizione

*La presente edizione del manuale del GBASIC v.7.35 soprassiede alla precedente della versione 7.30.*

*Le differenze tra il GBASIC versione 7 e le precedenti sono molto consistenti: mai, nella storia del GBASIC, una nuova versione è risultata così differente dalle precedenti.*

*Le variazioni introdotte non sono tanto relative a nuovi comandi quanto all'impostazione generale dell'interprete che finisce sempre più per assomigliare ad un sistema operativo. Si introduce un concetto fondamentali:la netta separazione tra un nucleo standard ("kernel") ed i programmi, adesso modulari, che gestiscono le periferiche ("drivers").*

*Il concetto del "driver" è presente nel GBASIC fino da tempi remoti, ma la linea di demarcazione con il nucleo dell'interprete era quanto mai sfumata ed in pratica i due costituivano un tutt'uno indissolubile risultando le relative interconnessioni complesse, irregolari e molteplici.*

*La nuova filosofia consente una significativa evoluzione del modus operandi: qualunque utente può adattare il kernel al proprio hardware e scrivere propri drivers senza dover disporre del sorgente GBASIC e senza necessità di entrare nei suoi meandri.*

*Risulterà evidente la scomparsa di vari comandi, in particolare quelli del video driver 9129 che, quando e se sarà implementato per la versione 7, avrà un proprio manuale.*

*Molti comandi sono stati poi aggiunti e varie opzioni che rendono ancora più potente l'impiego del sistema.*

Libera riproduzione e libero uso

*L'impegno di migliorare, mantenere e documentare il GBASIC è molto gravoso: gli utenti possono contribuire comunicando i loro problemi, gli errori riscontrati sulla documentazione ecc. a mezzo del modulo che si allega in fondo al manuale o meglio, quando possibile, inviando una fotocopia della pagina errata con le correzioni o le aggiunte.*

*Questa nuova edizione è relativa alla versione 7.35 che presenta molte innovazioni minori rispetto alla 7.30.*

Firenze, Ottobre 1990

Libera riproduzione e libero uso

# *Presentazione*

*Quest'anno il GBASIC compie dieci anni. Nato per sfruttare le caratteristiche grafiche di un chip allora in voga (la "G" stava per "grafico"), ha dato poi il meglio di sè nel campo dell'automazione industriale e del controllo di processo.*

*La presente edizione del manuale del GBASIC v.7.40 viene presentata esattamente tre anni dopo quella della versione 7.30.*

*In tutto questo tempo il "piccolo interprete" ha dato ulteriore prova delle sue qualità consentendo di realizzare diecine di applicazioni perfettamente funzionali in tempi veramente ridotti.*

*Le differenze tra il GBASIC versione 7.4 e 7.3 sono consistenti ma non rivoluzionarie. Il prodotto è oramai ben stabile ed affidabile; il passaggio di versione non introduce inoltre incompatibilità nei programmi sorgenti, se non in aspetti estremamente marginali.*

Firenze, Ottobre 1993

Libera riproduzione e libero uso

# Introduzione

Questo manuale descrive il GBASIC vers. 7.41. Comprende le seguenti sezioni:

1) introduzione

2) installazione - livello minimo

3) guida dell'utente - particolarità

4) riferimento

5) come scrivere un driver custom

6) modulo segnalazione errori

## Descrizione generale

GBASIC è un sistema composto da vari moduli destinato ad accelerare lo sviluppo di applicazioni su microprocessori della famiglia Z80 e compatibili utilizzanti hardware generico (nel seguito chiamato "sistema target" o semplicemente "target").

Sinteticamente, il sistema GBASIC si compone dei seguenti elementi:

- un sistema operativo residente in ROM sul sistema target; la sua struttura è modulare e può essere facilmente modificato o espanso dall'utente scrivendo opportuni "drivers" per le proprie periferiche.

- un traduttore GBASIC, sempre residente nella ROM del sistema target e capace di interpretare programmi GBASIC scritti dall'utente direttamente sul sistema target, utilizzando un terminale, o programmi GBASIC residenti in ROM.

- un precompilatore (PREGB), utilizzabile su un sistema tipo PC, che converte un file con un programma sorgente esteso (ossia scritto con una sintassi più libera del GBASIC ROM) in un file che può essere direttamente scritto in ROM ed interpretato dal GBASIC ROM.

- un ambiente integrato di sviluppo, a finestre, denominato GBT, utilizzabile su un PC.

Essendo stato concepito per applicazioni di controllo, il GBASIC possiede molte caratteristiche e funzioni non comuni ai normali interpreti "gestionali", come ad esempio la possibilità di una complessa gestione delle interruzioni ed un facile collegamento con il linguaggio assembler, tramite drivers modulari. Queste caratteristiche sono illustrate nel dettaglio nelle apposite sezioni di questo manuale.

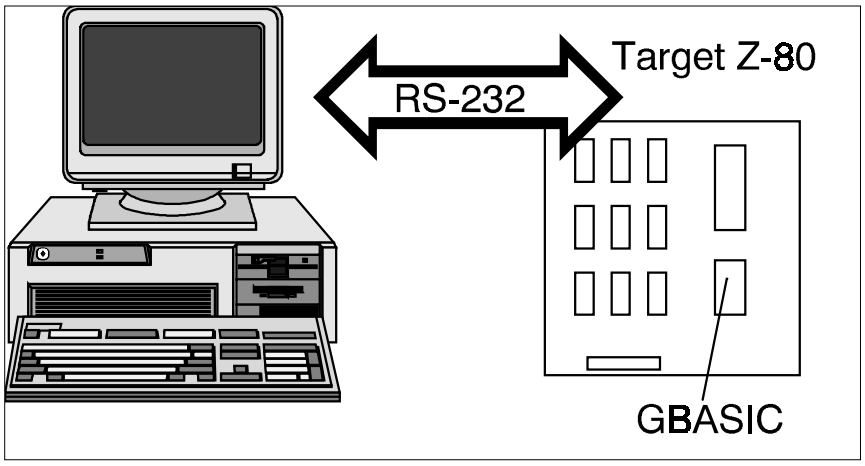
Ciononostante il GBASIC conserva le caratteristiche peculiari di semplicità di impiego e rapidità di sviluppo comuni al BASIC standard.

La struttura del sistema ROM è stata concepita per la massima generalità e si compone fondamentalmente di un "kernel" indipendente dall'hardware e una serie di "drivers" per la gestione dell'hardware specifico. I drivers ed il kernel comunicano tra loro con un meccanismo che non prevede indirizzi comuni. In questo modo non è richiesta una operazione di link tra i vari moduli.

Libera riproduzione e libero uso

# Sistema tipo

Un tipico sistema di sviluppo è costituito da un generico target Z-80 connesso ad un PC attraverso un canale seriale RS-232.Attraverso il PC è possibile dialogare direttamente con il GBASIC presente sulla scheda, compilare dei programmi scritti in GBASIC esteso ed inviarli al target.




# Ciclo di sviluppo

Il normale ciclo di sviluppo è il seguente

1) editazione del programma GBASIC esteso sorgente tramite GBT o altro editor fi proprio gradimento.

2) precompilazione tramite PREGB

3) invio del programma al target in formato ASCII precompilato (.TXT) o in formato memory image (.HEX). Per quanto possa essere utilizzato un qualunque programma di emulazione terminale, GBT svolge questa funzione in modo ottimale, consentendo inoltre molte altre funzioni.

4) prova del programma con possibilità di ispezione dei simboli. In questa fase si trae il massimo beneficio dalla implementazione ad interprete, potendo interagire direttamente con il programma.

Libera riproduzione e libero uso

L'interfaccia utente GBT consente di eseguire tutte le fasi del ciclo restando sempre nel medesimo ambiente.

## Lo Z-80

L'immortale Z-80 è il cuore dei sistemi GBASIC. Le più recenti versioni di Zilog e Toshiba gli consentono tuttavia prestazioni di tutto rispetto anche a velocità di 10 MHz ed oltre.

Il GBASIC è stato collaudato su sistemi Z-80 e 84C015. Non può invece operare completamente su Z-280 che ha alcuni bugs incompatibili con il sistema di interruzione.

## I drivers

Sono stati scritti molti drivers per GBASIC ma non tutti sono ancora documentati; molto interessanti per esempio il mini file system per RAM paginata, il driver per orologio calendario ecc.

Completamente documentati sono invece i drivers della rete locale G-LAN ed i drivers del controllo motore National LM-628.

Contattate la AEP per ogni informazione prima di iniziare a scrivere un driver; è probabile che già esista.

## Limiti

L'interprete GBASIC, malgrado le notevoli potenzialità, è contenuto in 18-32K di eprom (a seconda dell'implementazione). Molti sforzi sono stati fatti per contenere la dimensione dei programmi, anche in vista della limitazione di indirizzabilità dello Z-80.

Libera riproduzione e libero uso

Questo ha implicato alcune rinunce, in particolare alla diagnostica che risulta piuttosto limitata.

Esistono poi, come nel C, condizioni di programma che possono portare a crash di sistema anzichè all'emissione di messaggi di errore.

Malgrado tutto, con un minimo di attenzione, il GBASIC risulta perfettamente stabile anche in fase di sviluppo.

Libera riproduzione e libero uso

# Precompilatore PREGB

## Generalità

Programmi GBASIC possono essere scritti direttamente sul sistema target per mezzo del terminale di console. Ogni volta che una linea viene digitata, il kernel provvede alla sua codifica in un formato più compatto del semplice ASCII. Questa operazione detta *tokenizzazione* ha anche lo scopo di migliorare il tempo di esecuzione e viene eseguita in senso inverso quando si impartisce il comando LIST per ricostruire le linee originali. Il formato tokenizzazto è detto GBA (GBASIC Assoluto).

Normalmente, tuttavia, i programmi sono scritti su un personal computer (286 o maggiore) per mezzo di un qualunque text editor e convertiti quindi nel formato tokenizzato per mezzo del pre-procompilatore PREGB (pregb non opera su PC 8088).

PREGB accetta in ingresso un programma scritto in una sintassi più estesa del GBASIC standard (ad esempio accetta nomi di variabili lunghi fino a 30 caratteri ecc) mentre genera un file oggetto del tutto compatibile con il formato GBA. Il file di ingresso ha suffisso .GBE (GBASIC Esteso).

Libera riproduzione e libero uso

Il formato GBA non prevede indirizzi assoluti; il programma codificato è quindi completamente rilocabile in qualunque posizione di memoria. Di solito viene posto in eprom ed eseguito con il comando RUN <exp> senza nemmeno caricarlo nell'area di lavoro. Procedendo al caricamento in ram (con LOAD ABS), si ha invece la possibilità di modificarlo a scopo di debug.

I vantaggi derivanti dall'uso di PREGB sono molto consistenti e si suggerisce di non prendere nemmeno in considerazione l'ipotesi di sviluppare il programma direttamente sul target.

## *Altri files generati*

PREGB genera vari tipi di files:

**.GBA** - sopra descritto

**.HEX** - è il file .GBA convertito in formato Intel hex. Viene usato normalmente per il trasferimento al target (comando LOAD HEX). Questo file viene ottenuto con un'utility dal .GBA.

**.TXT** - con una opzione di compilazione (/l) è possibile generare assieme al file .GBA anche un file .TXT che contiene il programma scritto, in ASCII puro, nella sintassi ristretta propria del target; insomma un listato sorgente identico a quello che otterrebbe eseguendo un LIST sull'unità target.

Questo file può essere usato per eseguire il caricamento sul target, in modo alternativo, attraverso un programma di emulazione terminale anzichè attraverso la eprom. L'interprete, ricevendo le linee dal canale di console, non sarà infatti in grado di distinguerle da quelle che avrebbe ricevuto se le avesse direttamente digitate l'operatore. E' tuttavia preferibile usare il trasferimento .HEX.

Con una diversa opzione (/s) il file .TXT conterrà anche tutte le linee di commento ed altre informazioni ausiliarie come la tavola dei simboli.

Libera riproduzione e libero uso

**.SYM** - tavola dei simboli letta automaticamente da GBT.

**.GBX** - il GBASIC prevede la possibilità di suddividere programmi .GBA troppo lunghi in due pagine di memoria attivabili con un meccanismo di bank switching (vedi direttiva #PAGE). Il file .GBX contiene in questo caso il .GBA da registrare nella pagina estesa.

## Contenuto del dischetto

Il dischetto fornito è registrato nel formato MS-DOS 360K e contiene i seguenti files:

PREGB.EXE - pre-processore PREGB

HEX.EXE - file ausiliario

PREGB.OVL - file ausiliario

DEMO.GBE - file sorgente dimostrativo

GBT.EXE - ambiente integrato (è descritto nel relativo manuale).

## Installazione ed uso

Per installare PREGB, basta copiaretutti i suddetti files in un qualunque direttorio (di solito chiamato IDE).

Se XYZ.GBE è il file sorgente, lo si pre-processa con:

pregb xyz

Si noti l'omissione del suffisso dopo XYZ; esso viene automaticamente assunto come .GBE da PREGB.

Eventuali errori sono listati sul video assieme alla loro spiegazione (vedi paragrafi seguenti).

Come molti compilatori, PREGB possiede limitate capacità di rientro e può in certi casi dar luogo, dopo il primo, alla segnalazione di errori inesistenti. In questo caso ritentare la compilazione dopo aver eliminato il primo errore.

Desiderando anche la generazione del file .TXT si deve aggiungere uno switch "/l" come segue:

pregb xyz /l

Usando /s invece di /l, nel file .TXT verrà inserita, alla fine, anche la tavola dei simboli in modo da poter risalire rapidamente al simbolo originario dal nome di variabile GBASIC alfa - numero.In più vengono conservati tutti i commenti che iniziano a colonna 1.

Gli switches /l e /s devono necessariamente usare la lettera minuscola.

Normalmente si utilizza comunque l'ambiente integrato GBT per cui non è necessario impartire i comandi a livello di prompt.

# La sintassi estesa

La sintassi estesa comprende quanto segue:

a) possibilità di inserire commenti e linee vuote;

b) omissione dei numeri di linea;

c) etichette simboliche riferibili nei GOTO e GOSUB;

d) identificatori lunghi per le variabili;

e) possibilità di definire costanti;

f) pre dichiarazione dei nomi di variabile.

g) marcature data ed ora

h) compilazione condizionale

i) inclusione di files

j) procedure con variabili locali

## File sorgente

Il file sorgente può essere scritto in formato non-documento (ASCII puro) con qualunque text editor.

Le linee troppo lunghe possono essere continuate sulla riga successiva con il carattere '&' messo all'inizio del nuovo rigo. Non è ammesso continuare ulteriormente su un terzo rigo.

## Identificatori

Gli identificatori (etichette, variabili e costanti) possono essere lunghi a piacere ma solo i primi 30 caratteri sono riconosciuti. Devono essere composti da lettere e numeri ed iniziare per lettera. Viene riconosciuto il maiuscolo / minuscolo; ad esempio i seguenti nomi sono relativi a variabili diverse:

nomeprodotto

NomeProdotto

NOMEPRODOTTO

Non possono essere usati come identificatori i nomi delle parole chiave del GBASIC o quelli delle direttive del precompilatore o la parola STAMP che ha un significato particolare. Queste sono riconosciute indipendentemente dal carattere maiuscolo o minuscolo.

Il numero massimo complessivo di identificatori è 300. Gli identificatori, con esclusione delle etichette, devono essere dichiarati prima dell'uso come sotto indicato.

Per maggiore leggibilità, si conviene di scrivere gli identificatori di etichetta e di variabile in caratteri minuscoli, usando però la lettera maiuscola per il primo carattere di ogni parola componente. Ad esempio

NomeProdotto

PrintLoop

VolTotErog

Gli identificatori senza particolare significato, invece, come a$, si scrivono con l'iniziale maiuscola.

Si propongono le seguenti convenzioni:

**costanti** - tutte maiuscole

**variabili** - iniziale minuscola - uso intermedio delle maiuscole per chiarire la composizione; es. i, j, k, tempLim

**etichette** - iniziale maiuscola - uso intermedio delle maiuscole per chiarire la composizione; es. PrintTable

**procedure** - come le normali etichette ma precedute da 'Pr'. Es. PrEvalTime.

## *Campo di validità*

Ogni simbolo definito dall'utente è caratterizzato anche dal nome della procedura cui appartiene (il nome della procedura è costruito con le stesse regole degli altri simboli).

I simboli della procedura Main, il programma principale che contiene le procedure, sono globali ed accessibili a tutte le procedure.

I simboli definiti nelle procedure (vedi dopo) hanno validità solo all'interno della procedura. Ci si riferisce ad un simbolo di una procedura con l'operatore "#procedure" seguito dal nome della procedura desiderata. Es..

```
gosub #procedure Prova Prova
```

significa "vai alla etichetta Prova della procedura Prova" (idem per variabili e costanti).

Una procedura può ridichiarare come propri gli stessi simboli usati nella procedura Main o in altre procedure. I simboli vengono ricercati prima tra quelli locali e poi, se non trovati, tra quelli globali.

Le variabili, pur se locali, sono allocate dal GBASIC in modo tradizionale. Non sono quindi possibili chiamate ricorsive o simultanee da parte di più task.

Pregb v.2 produce sempre il file .SYM contenente l'elenco ed i valori di tutti i simboli utilizzati dal programma e detto file viene caricato automaticamente da GBT e reso accessibile al programmatore.

# Struttura di un programma .GBE

Un programma .GBE ha la seguente struttura:

```
#DECLVAR
          (dichiarazione delle variabili)
#END

#DECLCONST
          (dichiarazione delle costanti)
#END
#CODE
          (programma)
#END
```

Si notino le "direttive" #DECLVAR, #DECLCONST, #CODE, #END che non vengono tradotte in frasi GBASIC ma sono delle semplici istruzioni per il pre processore. Le direttive devono iniziare sempre sul primo carattere della linea. Altre direttive sono illustrate nei paragrafi seguenti.

Il carattere apice (" ' ") indica che tutti i caratteri conseguenti sono di commento e possono essere ignorati. Linee di puro commento sono ammesse così come linee vuote. In questo caso è consigliabile apporre l'apice sulla prima colonna e non inserire spazi o tabs nelle linee vuote per accelerare l'operazione di precompilazione.

Per altre informazioni vedere quanto illustrato nel paragrafo precedente "Identificatori".

## Procedure

All'interno della sezione #CODE, detta procedura Main, è possibile definire delle **procedure** nel modo seguente:

```
#PROC Prova
#DECLCONST   'def.di cost. locali (opzionale)
........
#END
#DECLVAR     'dichiarazioni variabili locali
........
#END
#-> Prova    'entry point della procedura
........     'corpo della procedura
........
........
#ENDPROC
```

Valgono le seguenti regole:

a) il nome dell'entry point di una procedura non deve necessariamente essere uguale a quello della procedura. Poichè è possibile riferirsi dall'esterno a qualsiasi label interna ad una procedura, è possibile avere anche più entry points.

b) una procedura può contenere sottoprogrammi.

c) ad esclusione della procedura Main, una procedura non può contenere una procedura.

## Dichiarazione delle variabili

Tutte le variabili utilizzate nel programma devono essere dichiarate nella sezione DECLVAR.

Si riporta un esempio della dichiarazione delle variabili a, b, c, Radice e Delta.

```
#DECLVAR
       a,b,c           'coefficienti
       Radice, Delta
#END
```

Il pre processore assegnerà a ciascuna delle variabili simboliche di cui sopra una variabile GBASIC iniziando da A0 per la prima, A1 per la seconda ecc. e convertirà di conseguenza tutte le loro occorrenze nel corso del programma.

Si noti che di conseguenza:

a) non viene dichiarato il tipo: nel corso del programma avremo la libertà di usare l'identificatore seguito da $ come stringa o da parentesi ( come vettore. Per il GBASIC

Radice, Radice$ e Radice(n)

sono tre variabili differenti. Difatti se "Radice" viene convertita in A3, avremo che "Radice$" viene convertita in A3$ e "Radice(n)" viene convertita in A3(n);

b) non possono essere dichiarate più di 260 variabili (poiché possono essere convertite solo in A0..Z9); non è una grande limitazione anche tenuto conto di quanto espresso sopra e comunque della limitazione della memoria effettivamente a disposizione.

c) le variabili stringa o vettore devono essere dimensionate con DIM, come di solito.

# Dichiarazione delle costanti

E' possibile dichiarare delle costanti. L'uso di costanti simboliche presenta molti vantaggi tra cui:

- maggiore leggibilità del programma;

- possibilità di eseguire variazioni agendo in un solo punto

Descriviamo con un esempio: si vogliono dichiarare due costanti, una numerica, "PiGreco" = 3.14 ed una alfanumerica "Prodotto" = COCACOLA.

```
#DECLCONST
        PiGreco = "3.14"
        Prod $ "COCACOLA"
#END
```

**Note:**

a) il '$' che sostituisce lo '=' nella seconda assegnazione specifica che "Prodotto" è una costante alfanumerica.

b) tutte le volte che nel corso del programma viene incontrato il nome di una costante, esso viene sostituito dalla stringa dichiarata, senza alcun ulteriore controllo; le costanti assegnate con $ tuttavia sono rimpiazzate anche con le virgolette (") iniziali e finali;

c) una costante non può superare i 40 caratteri;

**Esempio**

```
let Area = 2 * PiGreco * Raggio
let a$ = "Bevete "
let b$ = prodotto
print concat$(a$,b$)
```

## *Costanti numeriche nuovo tipo*

Una costante numerica può essere alternativamente dichiarata nella forma seguente:

```
K8279 = _123
```

Si osservi come, al posto delle virgolette, si è usato il carattere underscore davanti al numero. Le costanti definite in questa seconda maniera sono note anche in valore al precompilatore che può quindi utilizzarle per valutare espressioni.

La vecchia notazione con le virgolette resta in uso e serve tanto per compatibilità con i vecchi programmi quanto per definire quantità non intere (o fuori del rango +/- 32k)

**Esempio:**

```
#DECLCONST
  SCRIBA = _1
  PASSW = "71215"
```

Le espressioni di compilazione possono essere utilizzate solo nella nelle #IF (vedi dopo) o nella sezione DECLCONST; in questo ultimo caso non devono però iniziare con una costante generica (quella tra virgolette).

Le espressioni di compilazione possono essere composte con:

- costanti di compilazione numeriche intere (es. _123)

- costanti di compilazione generiche (es. "123")

- identificatori di costanti di compilazione (intere o generiche, purchè nel rango +/-32k) già definite (es. SCRIBA)

- gli operatori ! (not), | (or), & (and), +, -, *, /.

L'ordine di valutazione è strettamente sequenziale. Tra due termini ci può stare solo un operatore. Es.

```
SCRIBA | ! FMDRC
```

non è ammessa; si scriverà invece

```
! FMDRC | SCRIBA.
```

### *STAMP*

Una costante alfanumerica cui si assegna il valore STAMP verrà a contenere ora e data della compilazione (vedi sotto).

## Il programma

Il programma viene scritto nel solito modo con le seguenti eccezioni:

a) la sezione programma inizia con #CODE e termina con #END;

b) non vengono indicati i numeri di linea;

c) il testo può essere scritto in maiuscole o in minuscole. Le parole chiave vengono comunque riconosciute, mentre, come detto sopra, per gli identificatori vengono eseguite le necessarie distinzioni. Possono essere usati liberamente spazi, tab e linee vuote. Il carattere apice (') indica l'inizio di un commento. Tutto ciò che segue fino a fine linea viene ignorato;

d) quando è necessario identificare un punto del programma si ricorre alla direttiva "label" con il simbolo #-> seguito da un identificatore, creato con le regole già espresse.

#### Esempio

```
#-> Loop
  print "sono in loop!"
  goto Loop
```

e) durante la conversione vengono generati automaticamente i numeri di linea (di 10 in 10) ed i riferimenti sono opportunamente aggiustati. Ad esempio il programma precedente potrebbe diventare:

```
10 PRINT "sono in loop"
20 GOTO 10
```

# Compilazione condizionale

E' possibile determinare la compilazione condizionale di uno o più blocchi di istruzioni, in funzione del valore di una certa costante mediante l'uso della direttiva "#IF". Il blocco di istruzioni viene racchiuso tra una "#IF" ed una "#ENDIF". La "#IF" viene seguita dalla costante di comando. Se questa vale 0 il blocco non viene compilato, viceversa se vale 1.

### Esempio

```
#IF DEBUG
  print a,b,c
#ENDIF
```

In un programma possono essere contenute tante direttive "#IF" quante si vuole ma non possono venire nidificate. Le uniche direttive che un blocco "#IF" può contenere sono le etichette di programma.

Le linee di commento contenute in un blocco IF non sono riportate nel TXT in caso di espressione falsa.

# Inclusione di files

Il file contenente il programma GBASIC può includere altri files con la direttiva #INCLUDE nonfile.

Il file incluso deve terminare con #ENDINCLUDE.

Un file incluso non può includere altri files.

# Time-stamping

Ogni file generato .GBA con PREGB è marcato con il "time-stamp", ossia una stringa ASCII di 26 caratteri che indica esattamente giorno, mese, anno, ora, minuto e secondo della sua "pre-processazione". Questo rende possibile la sua esatta identificazione anche sulla eprom. Il time stamp si trova in coda al programma dopo la sequenza 1,0,0. La lettura del time-stamp pur avvenire in vari modi: pc-tools, debug, eprom1 ecc.

Il time stamp è accessibile anche al programma definendo una costante alfanumerica uguale alla parola riservata STAMP.

### Esempio

```
VERSIONE $ "STAMP"
```

In questo modo la costante "VERSIONE" assume il valore del time-stamp.

# Diagnostica

PREGB non è un compilatore inteso in senso tradizionale in quanto si limita ad eseguire una semplice pre-elaborazione del testo in ingresso .GBE. Non vengono quindi effettuati controlli di alcun tipo sulla sintassi del GBASIC, per cui ad esempio la frase

```
THEN 12+PRINT
```

risulta perfettamente legale.

Ciononostante è prevista una diagnostica abbastanza accurata ed estesa degli errori che possono essere rilevati nei limiti sopra indicati. Si riportano i principali messaggi di errore che possono comparire:

```
- previsto identificatore
- identificatore non dichiarato
- bug interno
- file sorgente non trovato
- troppi argomenti - ignorati
- impossibile creare file di uscita
- del file delle parole chiave (PREGB.OVL)
- non più simboli GBASIC disponibili
- direttiva sconosciuta
- simbolo definito più volte
- previsto #END
- costante stringa non terminata da
  virgolette
- previsto =
- prevista costante stringa
- costante troppo lunga (max 40 car.)
- prevista #ENDIF
- prevista costante
- #IF dentro blocco #IF/#ENDIF
- espressione errata
- usare '_' davanti alle costanti numeriche di
  compilazione
```

I messaggi di errore sono inclusi nel file .TXT ma se nessun errore viene riscontrato non viene aggiunto in esse alcun messaggio tipo "nessun errore".

Si noti che caratteri anomali possono provocare lo sfasamento dei numeri di linea nei files prodotti.

# Conclusione

PREGB estende notevolmente la facilità di impiego del GBASIC. I programmi diventano più leggibili e di conseguenza assai più facili da scrivere e da manutenere, specie se l'utente ha l'accortezza di scriverli in modo opportuno, con rientri e commenti.

Libera riproduzione e libero uso

# Guida dell'utente

## Generalita'

Sono stati compiuti dei notevoli sforzi per rendere quanto più generale la struttura del kernel e quindi indipendente dall'hardware. Per poter operare a livello minimo, il kernel richiede tuttavia almeno un driver funzionante, quello del terminale di console. Assieme al kernel, viene fornito un file con il sorgente di un possibile driver di console che l'utente può quindi facilmente modificare secondo le proprie necessità (vedi relativo capitolo).

Si è constato che frequentemente l'hardware di sistemi Z-80 ricalca linee comuni e che di solito viene fatto impiego di periferiche della famiglia stessa: SIO, CTC, PIO.

In questo caso è possibile portare molto rapidamente il sistema a livello di funzionamento minimo variando solo i dati di alcune tabelle ammesso che:

a) come interfaccia per la console sia utilizzata una SIO;

b) che come baud rate generator per la stessa sia impiegato un CTC o che comunque il clock venga fornito senza necessità di programmare altre periferiche (ad esempio con

un oscillatore autonomo).

Il kernel, inoltre, incorpora la possibilità di usare in vari modi dei CTC in funzione di timers, contatori ecc. senza la necessità di scrivere i relativi drivers.

Dalla versione 7.3 è infine possibile per l'utente aggiungere delle proprie routines per la gestione delle condizioni di power fault. La versione distribuita di kernel non include di solito una gestione della mancanza di alimentazione. E' opportuno comunque eseguire le semplici verifiche indicate nel seguito per essere sicuri che il power fault sia escluso. Diversamente, come intuibile, si possono avere dei problemi di funzionamento conseguenti alle differenze tra il sistema target utilizzato e quello previsto.

# Le stringhe di configurazione

Ad indirizzi fissi, nel kernel, sono previste alcune stringhe di configurazione per la SIO della console e per il CTC (opzionale) usato come baud rate generator (BRG).

## *Stringa di configurazione della SIO*

Ha il seguente formato:

SIO <m> DATA <n> CONTR <o> VECT <p> "

dove

<m> è il numero della tabella vettori da utilizzare. Per adesso non si forniscono ulteriori spiegazioni su questo punto. Utilizzare il valore 0 per un canale A ed un valore 1 per un canale B.

<n> è l'indirizzo della porta dati, espresso come stringa di caratteri ASCII, in decimale.

<o> è l'indirizzo della porta controllo, espresso come stringa di caratteri ASCII, in decimale.

<p> è l'indirizzo della porta dove deve essere programmato il vettore di interrupt, espresso come stringa di caratteri ASCII, in decimale.

Le parole **SIO**, **DATA**, **CONTR** e **VECT** possono essere abbreviate e vengono di solito ridotte alla sola iniziale.

La stringa di configurazione della console viene registrata in ROM nell'apposito spazio previsto all'indirizzo 10H. Deve essere terminata con il doppio apice e può essere lunga fino a 32 caratteri, terminatore incluso.

### Esempio

```
S 0 D 120 C 121 V 123"
```

Anche per il BRG è prevista una stringa di configurazione simile alla precedente. Il suo formato è:

CTC <m> DATA <n>" dove:

<m> è il numero della tabella vettori da utilizzare. Per adesso non si forniscono ulteriori spiegazioni su questo punto. Utilizzare il valore 0.

<n> è l'indirizzo della porta dati, espresso come stringa di caratteri ASCII, in decimale.

La stringa di configurazione della console viene registrata in ROM nell'apposito spazio previsto all'indirizzo 40H. Deve essere terminata con il doppio apice e può essere lunga fino a 16 caratteri, terminatore incluso.

Esempio

C 4"

Nel caso che alla locazione assoluta 41H non sia presente una "C" il BRG non viene inizializzato. Se viceversa la stringa di configurazione è presente, il contenuto della locazione assoluta 40H viene interpretata come la costante da programmare nel divisore del CTC. Il suo valore è funzione del clock di sistema secondo la formula

Libera riproduzione e libero uso

baud rate = freq. clock / 256 / costante

Ad esempio, con un clock a 3.686.400 Hz si ottengono 2400 baud con una costante pari a 6.Pur essendo derivato dal BASIC standard (Dartmouth), il GBASIC presenta delle notevoli differenze rispetto ad esso e rispetto ai BASIC di solito utilizzati per programmazione gestionale.

In questa sezione sono indicate alcune delle più significative differenze e sono illustrate quelle particolarità utili per la realizzazione di sistemi di controllo. Si rimanda il lettore alla sezione di riferimento per la spiegazione dettagliata dei vari comandi.

Attenzione: la sezione che segue è stata organizzata "a schede" per una migliore comprensibilità: è da leggere tutta con attenzione perché contiene concetti importanti.

**Si suppone che ogni utente utilizzi il precompilatore per scrivere i propri programmi. Per questo motivo in questo manuale riporteremo spesso esempi che si suppongono essere precompilati prima dell'uso.In questi esempi le parole chiave (PRINT, LET ecc.) sono scritte in caratteri minuscoli. Nella sezione di riferimento invece, per una migliore comprensione, si riportano esempi scritti per essere digitati direttamente dal terminale di console. In questo caso le parole chiave sono scritte in caratteri maiuscoli.**

Libera riproduzione e libero uso

# Frase let

### Differenza con BASIC standard

A differenza del BASIC standard la frase **let** è sempre obbligatoria.

### Esempio

```
let a=3.
```

Libera riproduzione e libero uso

# Vettori e stringhe

### Differenza con BASIC standard

I vettori possono avere solo una dimensione. Se non dimensionati da programma assumono automaticamente dimensione 10.

Le stringhe devono essere dimensionate come i vettori indicandone la dimensione massima. Il tentativo di utilizzare stringhe non dimensionate determina un ERR STG. Nel dimensionamento prevedere un carattere in più del necessario per il terminatore.

In generale, costanti e variabili stringa non possono essere combinate per formare espressioni anche se in taluni casi espressioni stringa possono operare correttamente. Se ne sconsiglia tuttavia l'impiego.

### Esempio

```
dim a$(20)
```

Nelle IF possono adesso essere utilizzati gli operatori di relazione uguale, diverso, maggiore e minore

E' stato eliminato il bug relativo alle stringhe nulle della precedente versione (stringa nulla è ad esempio A$=""). E' inoltre possibile adesso il costrutto

```
let a$ = concat$(a$, "XXX")
```

(in precedenza non operava correttamente).

# Editing

### Uso pratico

Il programma GBASIC viene di regola scritto utlizzando un buon text editor (modo non-documento).Quando si è collegati con il sistema target, sono possibili ulteriori operazioni di editing:

- inserimento linea - semplicemente battendola

- eliminazioni linea - digitare il numero di linea seguito da return

- cancellazione ultimo carattere digitato - tasto backspace (od altro ridefinito con la frase BACKSP IS.. (vedi)).

L'esecuzione del programma può essere arrestata con ESC a meno che questa funzione non sia stata soppressa con BREAK 0 (vedi).

E' disponibile un programma denominato GBT destinato ad operare su PC che consente di eseguire le funzioni di editor a pieno schermo e di trasmissione dei programmi .TXT.

# L'ingresso / uscita (I-O)

### Concetti

A differenza dei sistemi gestionali, i sistemi di controllo dispongono in genere di periferiche di tipo non standard. In considerazione di ciò, il GBASIC offre una consistente varietà di strumenti per adattarsi alle particolari esigenze delle reali applicazioni.

Tutte le operazioni di ingresso uscita (con la sola esclusione delle periferiche autogestite che descriveremo sotto) avviene attraverso l'impiego di opportuni programmi denominati "drivers" esterni al nucleo centrale del GBASIC detto "kernel". In questo modo diventa più facile scrivere applicazioni destinate a funzionare su hardwares diversi. Il programma GBASIC infatti può rimanere invariato previa la riscrittura dei soli drivers. Alternativamente è possibile variare alcuni parametri che il GBASIC passa ai drivers (come ad esempio i numeri delle porte di I/O) per adattare il programma ad un sistema simile ma diverso nell'indirizzamento.

Ogni driver è identificato da un nome che deve necessariamente essere di 8 caratteri che non contiene spazi nè altri simboli ed inizia per lettera.

I drivers vengono posti fisicamente contigui al GBASIC immediatamente dopo la sua ultima locazione.

Il GBASIC non è a conoscenza della presenza dei drivers fintanto che non gli venga richiesto di ricercarne uno specifico con la frase **open**. Per mezzo di essa si ricerca il driver di nome indicato e gli si assegna un numero di canale scelto dall'utente. Successivamente, nel corso del programma, ci si riferirà al driver sempre attraverso il numero di canale e non più attraverso il nome. Esempio di una frase **open** è il seguente:

```
open ch 1, "CCAL:..."
```

Almeno un driver è sempre e comunque presente: il driver di console, identificato sempre dal numero 0 e dal nome "CONS:...".

Libera riproduzione e libero uso

# I dispositivi logici

### Concetti

Il GBASIC prevede una serie di dispositivi logici standard per l'I/O che possono essere diversamente ed anche dinamicamente associati con drivers differenti, purché ovviamente compatibili e sensati (non avrebbe ad esempio alcun senso associare il dispositivo logico orologio / calendario ad una stampante).

I dispositivi logici principali sono:

**con:** console (ingresso ed uscita)

**lst:** lista (solo uscita)

**aux:** ausiliario (ingresso ed uscita)

**timdat:** orologio calendario (ingresso ed uscita)

**err:** emissione dei messaggi di errore (solo uscita)

Un numero indefinito di dispositivi possono essere poi gestiti con altri metodi che saranno descritti nel seguito.

Per default il dispositivo **con:** è associato con il driver di console (ch 0), ma esso può essere riassegnato semplicemente con

con: <m> is ch <n>

dove <n> è il numero del canale del driver che si intende usare al posto di "CONS:..." e <n> vale 0 (per Output) o 1 (Input). E' infatti ammesso l'uso di un driver in ingresso e di un altro driver in uscita.

**Esempio**

Si riassegna l'uscita del dispositivo logico **con:** verso un driver di nome "LCD1:..."

```
open ch 2,"LCD1:..."
con: 0 is ch 2    'uscita consolle verso lcd
```

Le frasi **PRINT** ed **INPUT** utilizzano il dispositivo logico **con:** rispettivamente per l'uscita e per l'ingresso.

Le frasi INPUT e PRINT consentono la redirezione dell'ingresso e dell'uscita su uno specifico driver con gli appositi operatori (vedi PRINT ed INPUT)

Uno stesso driver può essere utilizzato per due dispositivi logici diversi.

**Esempio:**

```
open ch 2,"LCD1:..."
con: 0 is ch 2    'uscita consolle verso lcd
aux: 0 is ch 2    'uscita aux verso lcd
```

Il dispositivo **lst:** deve essere necessariamente associato ad un driver ed è usato di solito per la stampante.

**Esempio:**

```
open ch 4,"SERPRIN:"
lst: is ch 4
```

Il dispositivo **aux:** richiede anche esso una associazione obbligatoria con un driver. Usa il dispositivo **aux:** la frase **display** e la funzione **get$()**.

Il dispositivo **timdat:** è usato per l'orologio calendario qualora il sistema ne sia provvisto. Usano il dispositivo logico timdat: le frasi **time$()** e **date$()** nonché le funzioni **time$()** e **date$()**.

**Esempio:**

```
open ch 4,"CCAL:..."  'orologio calendario
```

```
timdat: is ch 4  print time$(), date$()
```

Il dispositivo err: è usato per l'emissione dei messaggi di errore. Viene inizializzato verso la console ma può essere modificato ad esempio per inviare i messaggi di errore ad una stampante. Questo è particolarmente utile a scopo diagnostico.

Libera riproduzione e libero uso

# Preliminari sull'interrupt

### Concetti

Il GBASIC prevede due tipi di interruzioni:

a) **interruzioni a basso livello**, gestite dai drivers e, per le periferiche autogestite, dal kernel stesso.

b) **le interruzioni ad alto livello**, generate di solito dai programmi di servizio delle interruzioni a basso livello e riconosciute alla fine della corrente frase GBASIC (salvo casi particolari o disabilitazioni). Le interruzioni ad alto livello provocano di regola l'esecuzione di una prestabilita subroutine GBASIC.

Le interruzioni a basso livello, delle quali adesso ci occupiamo, devono ovviamente essere previste dal programma. In particolare è necessario:

a) che ogni periferica che può richiedere interrupt sia opportunamente programmata;

b) che per ogni interrupt sia prevista la relativa routine di servizio (detta ISR = Interrupt Service Routine);

c) che l'indirizzo della ISR sia opportunamente registrato nella tavola dei vettori.

d) che l'indirizzo della tavola dei vettori sia programmato (1) nella periferica e (2) nel registro I.

L'utente che scrive il driver deve provvedere a sopra quanto specificato ma il GBASIC fornisce i necessari supporti in particolare per c) e d).

Libera riproduzione e libero uso

Tutti i vettori di interruzione utilizzati dalla CPU al momento dell'interrupt si trovano in RAM nella stessa area di memoria. La tabella contiene spazio per 3 SIO, 3 CTC e 3 PIO. La scrittura dei dati nella tabella non viene fatta direttamente dall'utente (che tra l'altro ne ignora la posizione). Invece, l'utente richiede al GBASIC di installare il vettore o i vettori desiderati specificando la posizione nella tabella prescelta (ossia indicando quale SIO delle tre possibili, quale CTC o PIO). Deve essere cura dell'utente evitare conflitti: ogni vettore può ovviamente servire una sola periferica. La funzione GBASIC, accessibile all'utente, che esegue l'installazione del vettore in tabella provvede anche a ritornare il vettore (parte bassa dell'indirizzo della tabella) che deve essere programmato nella periferica.

Libera riproduzione e libero uso

# Le stringhe di configurazione

### Concetti

Un driver può utilizzare, nell'accesso alle periferiche, indirizzi assoluti delle porte definiti nel driver stesso. E' però evidente che un driver cosiffatto deve essere modificato ogni qualvolta si esegua una variazione dell'hardware. E' quindi consigliabile utilizzare uno strumento posto a disposizione dal GBASIC che consente di fornire una serie di parametri ausiliari alla frase **open** che, nel loro insieme,prendono il nome di **stringa di configurazione**. Il meccanismo è molto simile a quello indicato al paragrafo 2.2 per la configurazione iniziale della console. Ad esempio:

```
open ch 4,"CCAL:...","D 12" 'orologio calendario
```

La stringa di configurazione non viene in alcun modo interpretata dal GBASIC che si limita a passarla inalterata al driver. Tuttavia il GBASIC mette a disposizione delle funzioni ausiliarie per analizzare la stringa stessa e per ricavarne i parametri in modo semplice e veloce (diversamente ogni driver verrebbe inutilmente appesantito se queste funzioni dovessero essere realizzate internamente).

Le funzioni ora dette sono utilizzabili per stringhe di configurazione che seguono uno schema convenzionale, ossia:

per SIO: "SIO <m> DATA <n> CONTR <o> VECT "

per CTC: "CTC <m> DATA "

per PIO: "PIO <m> DATA "

Si osservi che PIO e CTC non viene di solito fornita l'indicazione della porta vettore in quanto comunque ricavabile autonomamente dal driver (nel caso della SIO il vettore è presente solo nella porta B).

Libera riproduzione e libero uso

Le parole chiave (DATA, CONTR, VECT) sono identificate solo dalla prima lettera e possono pertanto essere abbreviate.

In certi casi può essere opportuno fornire stringhe combinate, come ad esempio:

```
open ch PRINTER,"TKTSPL:.","SIO 0 D 161 C 163 V 0 CTC 4 D 144"
```

Si noti che la parola PRINTER non è una parola chiave bensì una costante definita dall'utente per migliorare la leggibilità del programma e viene poi rimpiazzata da un numero dopo la precompilazione.

Libera riproduzione e libero uso

# Periferiche autogestite

Alcune unità logiche, se supportate dall'hardware, possono essere gestite senza la necessità di scrivere alcun driver. Queste unità logiche sono dette pertanto "autogestite".

Le unità logiche autogestite sono:

a) i contatori;

b) i timers;

c) gli oscillatori

Tutte le unità logiche autogestite della presente versione usano dei CTC. Per utilizzarle basta eseguire delle frasi **open** accompagnate da una opportuna stringa di configurazione per indicare al GBASIC gli indirizzi delle porte e le posizioni da utilizzare in tabella vettori.

Nell'esempio che segue si illustra una possibile configurazione. Non ci soffermiamo invece sull'uso di timers, contatori e contasecondi che sarà illustrato a parte.

### Esempio

```
' CTC U28 è usato come segue:
'      ch 0 (128) - counter 1- tabella CTC0
'      ch 1 (129) - counter 2- tabella CTC1
'      ch 2 (130) - baud rate- tabella CTC2
'      ch 3 (131) - counter 3- tabella CTC3

' CTC U33 è usato come segue:
'      ch 0 (144) - non usato - tabella CTC4
'      ch 1 (145) - timer 1- tabella CTC5
'      ch 2 (146) - timer 2- tabella CTC6
'      ch 3 (147) - timer 3- tabella CTC7
'(in parentesi sono indicati gli indirizzi
fisici)
```

```
open timer 1,"CTC 5 DATA 145"
open timer 2,"CTC 6 DATA 146"
open timer 3,"CTC 7 DATA 146"

open counter 1,"CTC 0 DATA 128"
open counter 2,"CTC 1 DATA 129"
open counter 3,"CTC 3 DATA 129"

on timer 1 gosub Timer1
on timer 2 gosub Timer2
on timer 3 gosub Timer3
```

Libera riproduzione e libero uso

# Interruzione ad alto livello

### Concetti

Un problema tipico legato all'uso di un'interprete residente è quello della velocità di esecuzione. La facilità di impiego e l'elevata interattività "costano" infatti un impiego non trascurabile del tempo della CPU, in particolare per il processo di traduzione. E' stato forse questo finora il principale ostacolo al loro impiego che pur presenta tanti vantaggi.

Il GBASIC, conservando le caratteristiche positive e negative degli interpreti, possiede però degli strumenti per una gestione veloce delle interruzioni che ne estendono l'uso ben oltre i limiti di un comune interprete.

Questo è stato ottenuto introducendo la possibilità di gestire interruzioni ad alto livello.

Le interruzioni a basso livello vengono gestite, come accennato in precedenza, dallo stesso driver. E' quindi il driver che predispone tutte le condizioni per l'interruzione e che contiene la routine di servizio.

Si hanno due casi:

a) la gestione dell'interruzione può essere integralmente affidata al driver. Questo è ad esempio il caso dell'accumulo caratteri in un buffer. Ogni volta che un carattere pervenuto genera una interruzione, la ISR lo preleva dalla periferica e lo inserisce in un buffer.

b) l'interruzione determina delle condizioni che richiedono una interruzione ad alto livello all'interprete GBASIC. Ad esempio l'arrivo di un carattere di fine messaggio che deve determinare in qualche modo una segnalazione al programma GBASIC per informarlo che il buffer è pronto per essere letto.

Libera riproduzione e libero uso

In questo ultimo caso, il driver provvede ad inserire nella coda di attesa la propria richiesta di interruzione. Il GBASIC esamina alla fine della esecuzione di ogni linea, se l'interruzioni non sono state disabilitate, la stato della coda delle richieste ed a servirle nell'ordine esatto in cui sono pervenute.

E' chiaro che in questo modo il GBASIC può far fronte a richieste che arrivino a velocità di punta elevata, ossia può acquisire richieste di interruzione in tempi brevi e processarle poi alla velocità che gli è propria.

In molti casi tuttavia non basta acquisire solo le richieste di interruzione. Riprendendo l'esempio prima riportato si comprende come, oltre alla richiesta di interruzione, sia necessario trasmettere al GBASIC anche il contenuto dell'ipotetico buffer con i dati pervenuti. Se infatti, nel frattempo che il GBASIC esegue i suoi processi, arriva un nuovo record, il precedente se ne va perduto.

Questo problema viene risolto con l'impiego di una struttura FIFO (First In First Out) (o "coda") che è assimilabile intuitivamente ad un tubo. Il GBASIC fornisce al driver le funzioni per inserire messaggi (nel nostro caso un record) ad una estremità ed al programma utente quelle per prelevare il prossimo messaggio all'altra estremità del "tubo".

In questo modo non si ha perdita di dati (almeno finche il driver riesce a mantenere il ritmo imposto dagli eventi esterni) e tutti i messaggi vengono accodati in parallelo alle richieste di interruzione.

Esaminiamo adesso più nel dettaglio la gestione delle interruzioni dal punto di vista del programma GBASIC. La scrittura dei driver è invece illustrata nella relativa sezione.

Sono disponibili 4 livelli di interrupt ad alto livello (più 3 riservati ai timers che al momento non ci interessano). E' possibile associare un driver ad un livello di interruzione con la frase ch.

Libera riproduzione e libero uso

### Esempio

```
open ch 1,"LANSLAV:"
ch 1 to interrupt 2
on interrupt 2 gosub ListenLan
```

Il programma di cui sopra attiva il driver "LANSLAV:" e gli assegna il canale 1. Quindi specifica che eventuali interruzioni dal driver **ch 1** devono provocare una interruzione a livello 2. Infine indica che eventuali interruzioni a livello 2 devono provocare il salto al sottoprogramma **ListenLan.**

Il sottoprogramma **ListenLan** avrà la struttura di un normale sottoprogramma ma sarà terminato dalla frase

return interrupt 2

essendo l'interrupt in questione di livello 2.

Se il driver **LANSLAV:** provvede ad inserire dati nel FIFO, il sottoprogramma **ListenLan** dovrà eseguire necessariamente una

pop a$ (o altra variabile stringa)

per prelevare dal FIFO la stringa che è stata inserita dal driver. La stringa **a$** dovrà ovviamente essere stata opportunamente dimensionata per contenere i dati in arrivo.

Le strutture FIFO utilizzate dal GBASIC (FIFO dei dati e FIFO delle richieste di interruzione) non sono inizializzate all'accensione al fine di evitare la perdita di informazioni e di richieste di interruzione in esse eventualmente accumulate qualora si faccia uso di memoria non volatile.

Il programma utente deve quindi eseguire almeno una volta la frase

reset interrupt

La frase

reset interrupt clear

,invece, "sprogramma" tutte le assegnazioni eseguite con le frasi "ON INTERRUPT" o "ON TIMER" e arresta i timers.

La funzione INFO(10) ritorna il numero di interruzioni pendenti presenti ancora nel FIFO.

Libera riproduzione e libero uso

# Timers e contatori

Il GBASIC gestisce anche interruzioni provenienti da dai timers delle periferiche autogestite. I timers hanno propri livelli di interrupt e non limitano quindi il numero dei drivers che usano l'interruzione.

Si riporta un esempio di programma che impiega un timer per stampare un messaggio ogni 3 secondi.

```
  open timer 1,"CTC 5 DATA 145"
  on timer 1 gosub Stampa
  load timer 1 with 3 freerun

#-> Loop
  goto Loop
#-> Stampa
  print "Timer scattato"
  return timer 1
```

L'espressione che segue la parola with indica il numero dei secondi che devono trascorrere prima dello scatto del timer. L'effettiva corrispondenza tra unità e secondi è conservata solo se il clock del sistema è a 3.68 MHz. Con clock diverso, per conservare la corrispondenza, si usa la frase

prescale <n>

dove <n> è una espressione il cui risultato deve essere compreso tra 0 e 255. Il valore di default è 56; valori più grandi rallentano i timers e viceversa (vedi PRESCALE).

La clausola opzionale **freerun** determina l'automatico ricaricarsi del timer allo scatto.

Un timer può essere arrestato con la frase

reset timer

Il valore corrente di un timer può essere letto con la funzione **timcnt(<exp>)** con <exp> numero del timer mentre la funzione **secs(<exp>)** ritorna il numero dei secondi trascorsi dall'avvio del timer stesso.

Oltre che come timers, i CTC possono essere usati come contatori di impulsi totalizzatori.

### Esempio

Il programma che segue attiva un contatore e ne mostra via via il valore:

```
  open counter 1,"CTC 0 DATA 128"
  counter 1,OK
#-> Loop
  print count(1)
  goto Loop
```

**OK** non è una parola GBASIC ma una costante definita dall'utente che vale 1 (che sta per "attiva"). Se la medesima avesse avuto il valore 0, avrebbe inteso significare "disattiva". Si noti che la parola OK non avrebbe potuto essere sostituita con la più ovvia "ON", in quanto ON è una parola chiave GBASIC (ON TIMER ecc.) Un contatore, quando disattivato, conserva il proprio valore accumulato fino all'esecuzione di una frase

reset counter <n>

Il GBASIC gestisce al massimo 3 timers e 3 contatori.

# Oscillatori

### Programmazione

Gli oscillatori sono costituiti fisicamente da dei CTC. Il GBASIC consente di inizializzarli con una opportuna costante di tempo, allo scopo precipuo di costituire dei baud rate generator per le SIO. Il modo di uso segue la stessa filosofia di quello dei contatori. La frase

OSC <m>,<n>

consente di fissare la costante dell'oscillatore <m> pari al valore <n>.

### Esempio

```
10 OPEN OSC 1,"C 1 D 145"
20 OSC 1,40
```

Il GBASIC può gestire al massimo 3 oscillatori. Si noti che essi non usano interrupt a basso livello ma che è indispensabile ugualmente indicare il numero di posizione in tabella nella stringa di configurazione.

Al reset gli oscillatori non sono bloccati: questo per evitare di inibire il clock anche alla console. Se inizializzato, il baud rate generator della console (il n.1) occupa la posizione 0 della tabella dei CTC.

Libera riproduzione e libero uso

# Complemento sui drivers

### Concetti

Fino ad 8 drivers possono essere aperti simultaneamente. Oltre alla frase open già vista esiste la

close ch <n>

che libera una posizione e disabilita il driver.

E' possibile accedere direttamente ai drivers senza passare attraverso i dispositivi logici per mezzo delle frasi

read ch <m>, <n>, <stringa>

write ch <m>, <n>, <stringa>

<m> è il numero del driver

<n> è il numero del record (che il driver può o meno gestire a seconda dei casi)

<stringa> è una stringa di caratteri trasferita da o verso il driver.

Ogni driver ha di solito una variabile di stato. Nel caso di un driver per terminale, ad esempio, la variabile di stato indica se sono arrivati caratteri. Anche in questo caso ogni driver ha un proprio comportamento. La funzione **drvst(<exp>)** ritorna la variabile di stato del driver <exp>. Drvst() può avere due parametri anziché uno per meglio sfruttare le caratteristiche di alcuni drivers.

# Organizzazione della memoria

### Concetti

Il sistema target può essere configurato in due maniere fondamentali:

a) EPROM da 0 a 32K - RAM da 32 a 40K (8K RAM)

b) EPROM da 0 a 32K - RAM da 32 a 64K (16K RAM)

Il GBASIC, all'accensione, provvede automaticamente a determinare quale delle due possibili configurazioni è quella effettivamente usata ed aggiusta i vari puntatori interni di conseguenza.

L'area RAM viene suddivisa in tre sottoaree principali:

a) variabili e tabelle interne del GBASIC

b) area di lavoro (programma e variabili utente)

c) area pseudo file RAM.

All'inizializzazione la parte residua di memoria sottratta la parte (a) viene equamente suddivisa tra (b) e (c) in parti uguali.

Per questo motivo, essendo la zona (a) variabile da versione a versione di GBASIC, quando si aggiorna il kernel è necessario riscrivere i dati eventualmente presenti nello pseudo file.

Libera riproduzione e libero uso

# Autostart

E' spesso auspicabile che al reset l'esecuzione di un certo programma residente in eprom venga avviata in modo automatico e trasparente per l'utente.

Il GBASIC può compiere questa funzione; di solito è uno switch collegato ad un bit di una porta di ingresso che viene utilizzato per determinare se procedere alla normale inizializzazione in modo console o se avviare automaticamente il programma (autostart).

Per eseguire l'autostart, il GBASIC deve possedere due importanti infornazioni: quale switch leggere e quale è la prima locazione ove risiede il programma da eseguire. Per questo motivo il funzionamento dell'autostart richiede un intervento di personalizzazione. La funzione di Autostart è contenuta nel driver di console. Riferirsi alla sezione 5 per ulteriori informazioni.

Il driver può determinare l'avvio del programma residente in RAM (se questa è ovviamente di tipo non volatile). In questo caso la prima istruzione del programma deve essere una REM per evitare l'azzeramento automatico della memoria.

La frase AUTOSTART (vedi) consente di abilitare la ripartenza automatica del programma in caso di errore o di interruzione per ESC.

Quando il GBASIC parte in autostart non vengono emessi messaggi sulla console.

Fino alla versione 7.4x il controllo dell'autostart era demandato al driver di console, che forniva anche l'indirizzo di partenza.

Libera riproduzione e libero uso

Adesso tutto funziona come prima ma l'utente ha la possibilità registrare nello storage sys (vedi frase STORAGE) una frase da eseguire all'accensione, ad esempio "RUN 23000" o altro.

Basta inserire, dopo le altre voci già indicate al punto 3, la frase AUTO:RUN 23000. Poichè la linea deve essere terminata da CR, si puÿ ad esempio scrivere

```
PRINT LDEV 5,"SYS: STOR:05K BRG:1 AUTO:RUN
23000";CHR$(13)
```

Si può inserire qualunque comando (tenendo però presente che lo storage sys contiene solo 50 caratteri); ad esempio LIST.

Ricordarsi di inserire una linea REM all'inizio del programma o si avrà la cancellazione automatica del programma in memoria.

Più precisamente, se è selezionato l'autostart si controlla se esiste un setup AUTO: nello storage SYS; se vi è viene utilizzato. Diversamente si usa la stringa del driver di console.

I comandi LOAD ABS e RUN sono concatenabili separandoli con i ":".

Se il sistema non dispone di ram non volatile (o se si teme una sua insufficiente affidabilità) è possibile registrare il contenuto dello storage SYS in ROM a partire dalla locazio C0 hex.

Se il sistema trova alla locazione C0H qualcosa diverso da 0, si ha la copia automatica al boot dalla C0H fino al terminatore 0 (che deve essere presente).

**Esempio:**

```
PRINT LDEV 5,"SYS: STOR:05K BRG:1 AUTO:LOAD
ABS 0:RUN";CHR$(13)
```

Libera riproduzione e libero uso

# Segmenti critici

## Concetti

I sistemi reali in cui viene utilizzato il GBASIC devono sovente conservare inalterate alcune informazioni al momento della perdita di alimentazione. Queste informazioni vengono di solito immagazzinate in memoria RAM munita di alimentazione non interrompibile.

La semplice conservazione dei dati non è tuttavia sufficiente in alcuni casi per garantire l'integrità del sistema. Si immagini ad esempio un programma che registra i dati relativi a certe operazioni in uno storage, aggiungendo un record ad ogni operazione, e che mantenga un puntatore all'ultimo storage scritto in un altro storage. Se la mancanza di alimentazione sopravviene tra la scrittura del record dati e la scrittura del puntatore si ha in pratica la perdita dell'ultimo record.

Per ovviare a questo inconveniente, GBASIC possiede un meccanismo di gestione di queste situazione denominato dei "segmenti critici". Grazie ad esso, il programmatore può definire delle sezioni di programma che non devono essere interrotti a metà per nessuna ragione. Se un'allarme di prossima mancanza di alimentazione si verifica durante una di queste sezioni, non viene servita la routine di gestione della mancanza di alimentazione fino alla fine della esecuzione del segmento critico.

I condensatori di filtro dell'alimentatore devono ovviamente essere dimensionati in modo tale da consentire l'esecuzione dell'intero segmento critico più lungo e della routine di gestione della mancanza di alimentazione.

Durante il segmento critico sono disabilitate tutte le interruzioni (hardware e GBASIC).

Libera riproduzione e libero uso

Il segmento critico si definisce con le frasi CRITSEG / ENDCRIT che non devono essere annidiate tra loro (ossia un segmento critico non deve contenere un segmento critico).

La funzione SHUTDN() consente di conoscere, durante l'esecuzione di un segmento critico, se è prossima o meno la mancanza di alimentazione ed è utile ad esempio per evitare di iniziare operazioni "lunghe".

La funzione DIAG() (vedi) ritorna un codice di errore nel caso in cui la tensione cade dopo l'inizio di un segmento critico e prima della sua fine.

In questo caso comunque si possono avere perdite di dati e la funzione DIAG() riporterà quasi certamente, alla riaccensione, anche un un errore di dati perduti.

Libera riproduzione e libero uso

# MMT

## Uso pratico

E' presente un mini multi task denominato MMT che ha principalmente la funzione essere utilizzato dai drivers. E' infatti piuttosto frequente che un driver piuttosto complesso abbia necessita' di usare un processo parallelo: ad esempio, per un driver di stampa con spooler, può essere utile realizzare un processo che ad intervalli verifica la presenza di caratteri da stampare. Il nome "multi task" è improprio poiché attualmente MMT opera a livello minimo anche se è previsto per una futura , eventuale più estesa funzionalità.

L'utilizzo di MMT può essere richiesto da particolari drivers. Riferirsi ai relativi manuali. Vedi il paragrafo 5.13 per ulteriori dettagli

All'accensione MMT e' inattivo. Per renderlo si deve:

a) attivare un oscillatore usando un CTC che non sia usato per altre cose (evitare OSC 0 che di solito e' impiegato come brg per la sio di console). Esempio:

open osc 2,"C 1 D 145"

b) specificare con l'apposita frase che l'oscillatore in questione e' usato come clock per il task switching. La frase e'

osc 2, 40 for switch

dove 403.19 MMT

# Alcuni tempi di esecuzione

### Uso pratico

Provati su hardware PDTF - clock a 3,6864 MHz

Per la misurazione, le frasi interessate sono state inserite in un loop di prova ed è stata rilevata la differenza con il loop "vuoto". L'avvio è stato controllato con una frase INPUT in modo da evitare l'overhead di avviamento che non è trascurabile.

### Loop di prova

```
FOR I=1 to 6000
NEXT
```

Tempo globale: 11.7 sec

### Programma di prova:

```
GOSUB/RETURN
FOR..
GOSUB 200
NEXT
200 RETURN
```

Tempo totale: 27.4 sec

Differenza con loop vuoto: 15.7 sec

Tempo singolo frase esaminata: 2.6 msec

### Programma di prova:

```
IF
FOR..
IF A=3 THEN LET A=A+2
NEXT
```

Tempo totale: 30.6 sec

Differenza con loop vuoto: 18.9 sec

Tempo singolo frase esaminata: 3.2 msec

**Programma di prova:**

```
LET CON DIVISIONE F.P.
FOR..
LET A=A/100
NEXT
```

Tempo totale: 26.8 sec

Differenza con loop vuoto: 15.1 sec

Tempo singolo frase esaminata: 2.5 msec

**Programma di prova:**

```
LET CON DIVISIONE DA VARIABILE
LET B=100
FOR..
LET A=A/B
NEXT
```

Tempo totale: 22.4 sec

Differenza con loop vuoto: 10.7 sec

Tempo singolo frase esaminata: 1.7 msec

**Programma di prova:**

```
LET CON VARIABILI MATRICIALI
DIM B(50)
LET C=25
LET B(25)=100
FOR..
LET A=A/B(C)
NEXT
```

Tempo totale: 30.9 sec

Differenza con loop vuoto: 19.2 sec

Tempo singolo frase esaminata: 3.2 msec

**Programma di prova:**

```
DISPLAY (lcd modello "FMD")
FOR..  DISPLAY 1,"12345678901234567890"
NEXT
```

Tempo totale: 68.5 sec

Differenza con loop vuoto: 56.8 sec

Tempo singolo frase esaminata: 95.0 msec

**Programma di prova:**

```
REM
FOR..
REM PIPPO PAPPO
NEXT
```

Tempo totale: 15.7 sec

Differenza con loop vuoto: 4.0 sec

Tempo singolo frase esaminata: 0.7 msec

**Programma di prova:**

```
INPUT LDEV 4 CON PICTURE
FOR..
INPUT PICTURE "!",LDEV 4,A$
NEXT
```

Tempo totale: 29.7 sec

Differenza con loop vuoto: 18.0 sec

Tempo singolo frase esaminata: 3.0 msec

```
LETTURA COUNTER
LET B=1
FOR..
```

```
LET A=COUNTER(B)
NEXT
```

Tempo totale: 33.3 sec

Differenza con loop vuoto: 21.6 sec

Tempo singolo frase esaminata: 3.6 msec

**Programma di prova:**

```
DOG (solo versioni custom)
FOR..
DOG
NEXT
```

Tempo totale: 13.9 sec

Differenza con loop vuoto: 2.2 sec

Tempo singolo frase esaminata: 0.4 msec

**Programma di prova:**

Influenza di interruzioni da timer (vers.6.80)

```
ON TIMER 1 GOSUB 100
LOAD TIMER 1 WITH 1 FREERUN
FOR..
LET A=A/100
NEXT
100 DISPLAY 1,I
110 RETURN TIMER 1
```

Tempo totale: 27.1 sec

Differenza con stesso esempio

senza timer 0.3 sec

Influenza su ogni ciclo di loop 0.05 msec

# Riferimento

## Contenuto

In questa sezione sono descritte in ordine alfabetico, tutte le frasi, le funzioni ed i comandi che compongono il GBASIC vers. 7.20.

## Generalità

Questa sezione del manuale è da intendersi come semplice consultazione; i concetti e le possibilità del GBASIC sono illustrati in altre parti del manuale.

## Accensione e reset

All'accensione del sistema target dove sia residente il GBASIC, viene emessa sul terminale di console una scritta di "hello" con il copyright.

Libera riproduzione e libero uso

Se il sistema non è mai stato inizializzato fin a quel momento, avviene automaticamente una inizializzazione e compare quindi la scritta "OK" che indica che il sistema è pronto ad accettare comandi.

Se il sistema è già stato inizializzato, compare una domanda

N per iniz ->

premendo "N" ( o "n") il sistema viene reinizializzato ed eventuali programmi presenti in memoria RAM sono perduti.

Se è presente l'opzione autostart e se sono verificate le condizioni per l'esecuzione dell'autostart (di solito, la posizione di un deviatore) il sistema inizia automaticamente ad eseguire il programma prefissato. L'autostart è una delle funzioni modificabili dall'utente - vedere la sezione 3 per ulteriori chiarimenti.

# Caratteristiche generali del GBASIC

Il GBASIC tratta numeri in virgola mobile con 10 cifre significative e campo dinamico compreso tra 10 alla 127 e 10 alla -128. Questo vuol dire che si possono trattare numeri grandi con 127 zeri o numeri piccoli preceduti dalla virgola e da 127 zeri.

I numeri delle linee di programma devono essere compresi tra 1 e 65535.

Sono disponibili, compatibilmente con lo spazio di memoria, le seguenti quantità e tipi di variabili:

1) 286 di tipo scalare (cioè senza indici);

2) 286 di tipo vettoriale, con un indice compreso tra 0 e 255;

3) 286 di tipo stringa, di lunghezza ciascuna compresa tra 1 e 255 caratteri.

Libera riproduzione e libero uso

La memoria disponibile limita ovviamente il numero complessivo delle variabili utilizzate ma non la loro scelta. I nomi delle variabili sono costituiti da una lettere dell'alfabeto inglese o da una lettera dell'alfabeto inglese seguita da una cifra.

**Esempi**

```
A
A1
A2
```

ecc Il nome delle variabili stringa è seguito da un carattere "$";

**Esempi**

```
A$
A1$
```

Le variabili ad indice vengono dimensionate automaticamente salvo diversa indicazione (vedi frase DIM) al valore 10. Poiché è presente anche l'elemento zero, il dimensionamento automatico dà luogo a 11 variabili.

L'utilizzazione dello spazio è ottimizzata, nel senso che è possibile utilizzare, a parità di memoria, grandi programmi e poche variabili o piccoli programmi e tante variabili. L'utente non deve procedere a stabilire le assegnazioni degli spazi che sono automatiche.

Tutte le stringhe devono essere dimensionate prima dell'impiego. La loro dimensione deve essere di uno maggiore della massima prevista.

Il GBASIC utilizza i seguenti caratteri:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
01234567890
+ - * / =
( ) " % , ; . [ ] & $ ^  < >
```

i caratteri sotto riportati possono invece essere utilizzati ma solo entro stringhe:

```
abcdefghijklmnopqrstuvwxyz
! #   ' : { }   ~ @ | \ ?
```

Le parole chiave devono essere scritte in maiuscolo.

La semplice introduzione di linee precedute dal relativo numero provoca automaticamente l'ordinamento delle stesse in senso crescente. E' opportuno prevedere numeri debitamente intervallati per facilitare successive inserzioni.

Una linea viene cancellata battendone il numero subito seguito da RETURN.

Ctrl/D - attiva o disattiva la conversione automatica minuscole-MAIUSCOLE (si/no ogni volta).

Il tasto ESC è attivo solo se non disattivato con la frase (vedi):

BREAK 0

La funzione di "backspace" viene attivata dal tasto omonimo, salvo diversa riassegnazione eseguita con la frase (vedi):

BACKSP IS <exp>

# Convenzioni e notazioni

Nel corso delle descrizioni si è preferito rinunciare ad un rigoroso formalismo a vantaggio di una migliore comprensibilità.

I comandi procedurali del GBASIC sono stati suddivisi in due categorie: frasi e comandi. Le frasi sono quelle strutture composte con comandi procedurali usati per lo più in esecuzione differita, ossia all'interno di un programma (es. PRINT A*3). I comandi sono invece usati per lo più in modo immediato (es. RUN). Alcuni comandi procedurali sono tuttavia usabili come frasi o come comandi (ad esempio, LOAD). In caso di uso improprio compare un messaggio di errore "ERR MODO DIRETTO". Non è previsto un errore di modo differito.

## Abbreviazioni

Per evitare una notazione complicata, si è preferito descrivere le parti opzionali indicando la forma di una frase o di un comando con o senza la parte opzionale. Ad esempio:

PRINT

PRINT <exp>

PRINT <string>

<exp> - sta per espressione aritmetica (es. 3.14*R).

<string> sta per stringa alfanumerica. Si noti che l'unica espressione stringa ammessa è composta da un solo elemento che può essere una costante (es. "ABC") o una delle funzioni stringa.

<var> - sta per variabile (es. A2, B$ ecc.)

<linnum> - sta per numero linea (es. 1230)

<val> - valore numerico (costante) (es. 123.2)

<explog> - espressione logica (es. A=0)

Altre abbreviazioni sono spiegate nel testo.

Libera riproduzione e libero uso

Quando una frase prevede una serie di parametri di numero non definito, si usano i puntini di sospensione; ad esempio

INPUT <var>,<var1>..

indica che dopo INPUT possono essere indicate una serie di variabili, var, var1 ecc.

Per PRINT ed INPUT, che avrebbero richiesto una descrizione della forma generale troppo complessa, si è ricorsi alla definizione a mezzo di una serie di esempi.

# Operatori

Gli operatori utilizzati da GBASIC sono i seguenti:

+ somma

- sottrazione

* moltiplicazione

/ divisione

= assegnazione (nella frase LET)

< minore

= uguale

> maggiore

=> maggiore od uguale

<= minore od uguale

L'operatore ^ è usato per identificare le frasi non interrompibili ed è descritto tra le frasi. Lo stesso simbolo è usato nell'introduzione di valori in notazione esponenziale.

# I comandi procedurali

Nelle pagine seguenti sono riportate in ordine alfabetico le descrizioni dei comandi procedurali usati per costruire le frasi GBASIC. Seguono qiuindi le funzioni ed i comandi.

I termini comandi, frasi ed istruzioni sono (anche se impropriamente) usati in modo intercambiabile.

Segue la descrizione di ogni singola frase.

## Operatore ^ (PRIVILEGED)

### Forma generale

```
<linnum> ^ <frase>
```

### Descrizione

inibisce l'accettazione delle interruzioni all'inizio della linea successiva. Normalmente le interruzioni, provenienti dai timers o da dispositivi periferici esterni, sono riconosciute all'inizio di una linea. In certe circostanze pur essere desiderabile inibire questo riconoscimento.

### Esempio

```
10 ^PRINT A
```

Le frasi FOR, CH n TO INTERRUPT, OPEN non richiedono l'operatore ^ poiché già inibiscono le interruzioni successive ed sono per questo dette frasi privilegiate.

## AUTOSTART

### Forma generale

```
AUTOSTART <exp>
```

### Descrizione

Quando si usa il modo di autostart, l'esecuzione del programma può essere interrotta con ESC o dal verificarsi di un errore. In questo caso si ritorna a livello di accettazione comandi (OK).

Se, tra l'avvio dell'esecuzione e l'interruzione, è stata eseguita una frase

```
AUTOSTART 1
```

(e non è stata eseguita una AUTOSTART 0) si avrà la ripartenza automatica dalla prima linea di programma.

Con <exp> = 0 si annulla l'effetto di AUTOSTART 1.

Vedi paragrafo 3.17.

## AUX:

### Forma generale:

```
AUX: <exp1> IS CH <exp2>
```

### Descrizione:

assegna al dispositivo logico ausiliario AUX: la periferica fisica gestita dal driver <exp2>. <Exp1> vale 0 per la funzione di uscita e 1 per quella di ingresso (0=Out 1=In).

Utilizzano il dispositivo logico AUX: :

- DISPLAY

- NGET$()

La frase AUX: viene utilizzata anche per riassegnare il dispositivo logico ausiliario ad un'altro driver.

### Esempio

```
10 OPEN CH 2, "LCD:...."
```

```
20 AUX: 0 IS CH 2
```

riassegna tutte le uscite dirette al dispositivo logico AUX: al driver LCD:

# BACKSP

### Forma generale

```
BACKSP IS <exp>
```

### Descrizione

La cancellazione dell'ultimo carattere digitato avviene di regola con il tasto "backspace" (codice ASCII 08). E' possibile riassegnare la funzione ad un tasto diverso con la frase BACKSP con <exp> uguale al codice ASCII desiderato.

### Esempio

```
10 BACKSP IS 19  'terminale Tesak
```

# BEEP

### Forma generale

```
BEEP
```

### Descrizione:

Provoca di un carattere BELL (ASCII 8) sull'attuale dispositivo di console. La sua funzionalità è quindi legata al comportamento del driver di console.

# BOOT

### Forma generale

```
BOOT
```

### Descrizione:

Provoca il riavviamento del sistema ("software reset"). Compare il messaggio iniziale di "hello".

Utile di solito per riavviare il sistema dei driver dopo avere arrestato il programma senza eseguire le frasi CLOSE per evitare l'errore ERR GIA' OPEN.

## BREAK

### Forma generale

```
BREAK <exp>
```

### Descrizione:

L'esecuzione del programma può essere arrestata in qualsiasi momento premendo il tasto ESC sull'unità di console (qualora questa ne disponga).

Il controllo del ESC può dover essere sospeso per i seguenti motivi:

1) perché si desidera impedire all'utente di uscire volontariamente o meno dal programma di gestione;

2) per accelerare la velocità di funzionamento (in particolare nella INPUT LDEV 4 (vedi)).

BREAK serve per disabilitare (se <exp> vale 0) o riabilitare (se <exp> vale 1) la funzione di ESC.

Esempio

BREAK 0

# CLEAR

**Forma generale**

```
1) CLEAR

2) CLEAR IS <string>
```

**Descrizione:**

Nella forma 1, CLEAR cancella tutte le variabili ponendo il loro valore a zero.

Nella forma 2, definisce il carattere di pulizia display da usarsi nella frase NGET$. Vedi paragrafo 5.13.

**Esempio**

```
10 LET A=10
20 CLEAR
30 PRINT A

RUN

 0
```

**Esempio:**

```
10 CLEAR IS CH "!"
```

# CLOSE

**Forma generale:**

```
CLOSE CH <exp>
```

**Descrizione:**

Chiude il driver di numero <exp>. La posizione corrispondente nel direttorio viene liberata. Il driver può essere riaperto senza generare messaggi di errore.

La CLOSE determina inoltre una chiamata alla funzione

driver #3 (CLOSE).

# CON:

### Forma generale:

```
CON: <exp1> IS CH <exp2>
```

### Descrizione:

assegna al dispositivo logico console CON: la periferica fisica gestita dal driver <exp2>. <Exp1> vale 0 per la funzione di uscita e 1 per quella di ingresso (0=Out 1=In). All'accensione, al dispositivo logico CON: viene assegnato, tanto in ingresso quanto in uscita, il driver "con:" che viene automaticamente aperto con il n.0 e deve essere fisicamente contiguo al kernel.

Utilizzano il dispositivo logico CON: :

- PRINT

- INPUT

- le segnalazioni di errore

- il tasto break (ESC) per la sola parte di ingresso

La frase CON: viene utilizzata per riassegnare il dispositivo logico di console ad un'altro driver.

### Esempio

```
10 OPEN CH 2, "LCD:...."
20 CON: 0 IS CH 2
```

riassegna tutte le uscite dirette al dispositivo logico CON: al driver LCD:

# COUNTER

**Forma generale:**

```
COUNTER <exp1>,<exp2>
```

**Descrizione:**

Abilita (se <exp2>=1) o disabilita (se <exp2>=0) la funzione di conteggio del contatore <exp1>.

**Esempio**

```
10 OPEN COUNTER 1,"C 5 D145"
20 COUNTER 1,1
30 PRINT COUNT(1)
40 GOTO 30
```

# CRITSEG

**Forma generale:**

```
CRITSEG
```

**Descrizione:**

Definisce l'inizio di un segmento critico. Vedi par. 3.18.

**Esempio**

```
1000 CRITSEG
1010 .......
1120 ENDCRIT
```

# DATA

**Forma generale**

```
DATA <val1>,<val2>,...
```

### Descrizione

La frase data crea un elenco di valori che possono successivamente essere letti dalla frase READ che li assegna a delle variabili. Possono essere usate più DATA; l'elenco dei valori viene costituito dall'insieme di tutte le frasi DATA, essendo il primo elemento dell'insieme il primo valore della prima frase DATA e l'ultimo l'ultimo della ultima frase DATA. In un programma la frase READ non può precedere la frase DATA ma peraltro le frasi DATA possono essere poste ovunque. I valori val1, val2 ecc. devono essere scritti esattamente come si introdurrebbero in risposta ad una frase INPUT (vedi).

### Esempio

```
10 DATA 12.6,7,897
20 READ A,B,C
30 PRINT A,B,C

RUN
12.6  7  897
```

### Esempio

```
10 DATA 3,6
20 DATA 6,8
30 READ A,C,C1,C2
40 PRINT A,C,C1,C2

RUN
3  6  6  8
```

## DATE$(x$)

### Forma generale

```
DATE(<string>)
```

### Descrizione

E' usata per regolare la data del dispositivo logico TIMDAT:. La stringa <string>, che deve avere il formato gg/mm/aa,

viene passata al driver in quel momento associato con TIMDAT: con una operazione di scrittura blocco sul record n.2 (il numero 1 è usato per l'ora). Non vengono eseguiti controlli sulla stringa tranne che per la sua lunghezza.

L'operazione viene eseguita mediante la funzione #0 del driver (WRITE BLOCK) sul record #2.

### Esempio

```
10 DIM A$(8)
20 OPEN CH 3, "CCAL:...","D 20"
30 TIMDAT: IS CH 3
40 LET A$=("14/05/89")
50 DATE$(A$)
```

## DIM

### Forma generale

```
DIM <var>(<exp>),<var1>(<exp1>)..
```

### Descrizione

La frase DIM serve per riservare in memoria lo spazio per una o più variabili ad indice o di tipo stringa di nome <var>, <var1> ecc. In caso di variabili numeriche la variabile <var> avrà <exp>+1 elementi, la variabile <var1> ne avrà <exp1>+1 e così via, essendo presente anche l'elemento di indice 0. Nel caso di variabili stringa, <exp> ne determina la lunghezza massima.

Variabili numeriche con indici compresi tra 0 e 10 vengono dimensionate automaticamente e non è quindi richiesto l'uso della frase DIM. Le stringhe devono sempre essere dimensionate ad un valore maggiore di una unità della lunghezza massima prevista.

### Esempio

```
10 REM NON RICHIEDE DIM
20 LET A(8)=12
```

### Esempio

```
10 DIM A(30),B(20),C$(30)
20 LET A(12)=0
30 LET A(A(12))=8
40 LET C$="VERCINGETORIGE"
```

Si noti che la frase DIM usata in modo diretto segnala ERRORE DI DIMENSION e non di MODO DIRETTO.

## DISPLAY

### Forme generali:

```
1) DISPLAY <exp>, <list>

2) DISPLAY <exp>
```

### Descrizione

provoca l'emissione del risultato delle espressioni che compongono la lista <list> verso il dispositivo logico AUX: e quindi verso il driver che in quel momento è associato ad esso. Il risultato di  viene passato al driver come numero di record e vale di solito 1 per la prima riga, 2 per la seconda ecc.  <list> è una lista di espressioni da stampare con le stesse regole della frase PRINT. Se  <list> (forma 2) viene omessa, il display viene ripulito.

### Esempio

```
10 OPEN CH 4, "LCD1:..."
20 AUX: 0 IS CH 4
30 DISPLAY 2,"ABCD"
```

## END

### Forma generale

```
END
```

### Descrizione

La frase END è stata eliminata.

## ENDCRIT

### Forma generale:

```
ENDCRIT
```

### Descrizione:

Definisce la fine di un segmento critico. Vedi par. 3.18.

### Esempio

```
1000 CRITSEG
1010 .......
1120 ENDCRIT
```

## ERR:

### Forma generale:

```
ERR<~>: <exp1> IS CH <exp2>
```

### Descrizione:

assegna al dispositivo logico ERR: la periferica fisica gestita dal driver <exp2>. <Exp1> vale 0 per la funzione di uscita e 1 per quella di ingresso (0=Out 1=In).

Il dispositivo logico ERR: viene usato per l'emissione dei messaggi di errore.

Dopo l'inizializzazione, il dispositivo logico ERR: ha il valore 0, ossia quello del dispositivo CON:.

### Esempio

```
10 OPEN CH 2, "PTR:...."
20 ERR: 0 IS CH 2
```

Libera riproduzione e libero uso

redirige tutti i messaggi al driver PTR:

# FOR

### Forma generale

```
        1) FOR <var>=<exp1> TO <exp2>

  2) FOR <var>=<exp1> TO <exp2> STEP <exp3>
```

### Descrizione

La frase FOR viene usata per eseguire ciclicamente tutte le istruzioni che stanno fra la frase FOR medesima e la prossima frase NEXT. La frase FOR significa: esegui tutte le istruzioni che seguono fino alla NEXT per la variabile var che va dal valore <exp1> al valore <exp2>, incrementandola ogni volta del valore <exp3>. Se STEP <exp3> viene omesso, alla variabile <var> viene aggiunto ad ogni ciclo il valore 1. Tutte le espressioni possono assumere qualsiasi valore.

Limiti: la frase NEXT deve essere presente o risulterà un ERRORE DI FOR..NEXT.

### Esempio

```
10 FOR I=1 TO 24
20 PRINT TAB(I),"*"
30 NEXT
```

Spiegazione: la frase 20 viene eseguita per I che va da 1 a 24. Poiché non è specificato STEP, ad ogni ciclo viene sommato 1 alla variabile I. I vale quindi 1 al primo giro, 2 al secondo e così via.

### Esempio

```
10 FOR H1=20 TO 18.5 STEP -0.5
20 PRINT SQR(H1)
30 PRINT H1*H1
40 NEXT
```

Spiegazione: le frasi 20 e 30 vengono eseguite per H1 che va

da 20 a 18.5; all'indietro perché STEP è negativo. H1 vale 20 al primo giro, 19.5 al secondo e così via.

# GOSUB

### Forma generale

```
GOSUB <linnum>
```

### Descrizione

Quando durante l'esecuzione viene incontrata la frase GOSUB viene eseguito un salto alla linea indicata da <linnum>. Non appena poi viene incontrata una istruzione RETURN si ritorna alla linea che segue la GOSUB.

<Linnum> deve corrispondere ad un numero di linea esistente o risulterà un ERRORE DI NUMERO DI LINEA.

### Esempio

```
10 GOSUB 1000
20 PRINT H1
30 STOP
1000 LET H1=45+7
1010 RETURN
```

Spiegazione: dalla frase 10 si salta alla 1000. Ll si esegue la LET.. ed alla frase RETURN si torna alla 20.

# GOTO

### Forma generale

```
GOTO <linnum>
```

### Descrizione

la frase GOTO serve per saltare alla linea di numero <linnum>.

<Linnum> deve essere il numero di una linea esistente o si avrà una segnalazione ERRORE DI NUMERO DI LINEA.

### Esempio

```
10 GOTO 300
 ....
300 PRINT A
```

## IF

### Forma generale

```
1) IF <explog> THEN <frase>

2) IF <explog> THEN <linnum>

3) IF <explog> THEN
        blocco di istruzioni
   ENDIF

4) IF <explog> THEN
        blocco di istruzioni 1
   ELSE
        blocco di istruzioni 2
   ENDIF
```

### Descrizione

la frase IF viene usata per eseguire la <frase> se si verificano certe condizioni, se cioè l'espressione logica <explog> è vera. Se cir non avviene l'esecuzione continua dalla linea subito seguente e la parte che segue THEN viene ignorata. L'espressione esplog è cosl costruita:

<exp1> <oprel> <exp2>

dove <exp1> ed <exp2> sono due comuni espressioni aritmetiche e oprel è un operatore di relazione scelto tra quelli che seguono:

= uguale

> maggiore

< minore

<= minore od uguale

= maggiore od uguale.

<> diverso

E' ammessa l'indicazione diretta di un numero di linea (forma numero 2) sottintendendo una "GOTO".

**Esempio**

```
10 IF A=0 THEN PRINT "ERRORE"
20 ..
```

Spiegazione: se A vale 0 allora scrivi errore e passa alla 20. Se A è diverso da 0 passa subito alla 20.

**Esempio**

```
10 IF W1+W2 THEN LET A=89
20 ..
```

Spiegazione: se la somma di W1+W2 è minore della somma di E+K poni A=89 e passa alla 20. Se non passa direttamente alla 20.

**Esempio**

```
10 IF 1=2 THEN PRINT "IMPOSSIBILE"
20 ..
```

Spiegazione: poiché 1 non è diverso da 2, la frase PRINT "IMPOSSIBILE" non sarà mai eseguita.

Per le stringhe sono ammessi solo gli operatori =, <>, >, <.

Le forme 3 e 4 ("IF strutturata") consentono l'esecuzione condizionale di interi blocchi di istruzioni.

Nella forma 3 "blocco istruzioni" viene eseguito se è vera

<explog>. Nella forma 4 viene eseguito il "blocco1" se <explog> è vera e "blocco2" se <explog > è falsa.

Le IF strutturate (IF..ELSE..ENDIF) possono essere nidificate tra loro fino a 9 livelli. Tenere però presente che:

a) non è ammessa l'uscita dalla IF con un GOTO o comunque deve essere eseguita una ENDIF.

b) ai fini della velocità di esecuzione, le linee non eseguite non sono "saltate" ma sono in realtà eseguite come se fossero delle REM.

**Esempio:**

```
10 IF A$<>"SI" THEN
20 PRINT "DIVERSO"
30 LET B=5
40 ENDIF
```

# INPUT

**Forma generale**

```
1) INPUT <var>,<var1>..

2) INPUT <string>,<var>,<var1>..

3) INPUT LDEV <exp1>, <var>,<var1>..

4) INPUT NOBLANK,<var>, <var1>..

5) INPUT PICTURE <string>, <var>

6) INPUT CH <drv>, <var>, <var1>..
```

Nota: le varie opzioni sopra riportate sono variamente combinabili tra loro. Le clausole PICTURE, CH, LDEV e NOBLANK possono essere inserite in qualsiasi ordine.

**Descrizione**

La frase INPUT è utilizzata principalmente per richiedere

all'operatore di introdurre il valore di una o più variabili <var>, <var1> ecc. Opzionalmente può essere specificata una stringa <string> racchiusa tra doppi apici.

Se <string> non è presente, quando la frase INPUT viene incontrata l'elaborazione si arresta e compare un punto interrogativo. L'operatore può così introdurre i dati richiesti; se essi sono più di uno li separerà con una virgola ed infine premerà RETURN per far ripartire il programma. Se <string> è presente, invece del punto interrogativo comparirà la <string>. Punto interrogativo e stringa interrogativa sono emessi sul dispositivo logico CON: / out mentre l'ingresso avviene attraverso CON: / in.

Si noti che battendo in risposta alle interrogazioni della frase INPUT il tasto RETURN a vuoto non risulta una condizione di errore. Se le variabili in ingresso sono numeriche, assegna loro il valore 0. Alle alfanumeriche ritorna invece una stringa nulla (di lunghezza 0).

Se si batte RETURN prima di avere introdotto tutti i dati compaiono due punti interrogativi ed è possibile continuare l'introduzione.

Le clausole PICTURE, LDEV, CH e NOCLEAR modificano la funzione della frase INPUT, come sotto descritto, e possono essere inserite in qualsiasi ordine.

**ATTENZIONE - SONO STATE INTRODOTTE VARIANTI CHE RICHIEDONO LA MODIFICA DEI PROGRAMMI ESISTENTI.**

E' stato variato l'uso della parola chiave CH nelle frasi PRINT ed INPUT. Questa modifica si è resa necessaria per una doverosa riarmonizzazione logica. Infatti la parola "CH" era usata impropriamente ed era ormai limitata solo ad alcuni casi specifici. La nuova logica è la seguente:

**la parola CH** viene usata per la redirezione dell'input o dell'output su un driver. L'espressione che segue dà il numero del driver così come dichiarato nella OPEN CH..

**la vecchia parola CH viene sostituita da LDEV** e serve per

redirigere il dispositivo logico (che normalmente è CON:).

Le possibilità sono quindi adesso due:

a) redirigere temporaneamente il dispositivo logico CON:, normalmente usato da PRINT ed INPUT, su un qualunque driver;

b) redirigere temporaneamente le frasi PRINT ed INPUT su un diverso dispositivo logico.

Vedere nel seguito ulteriori spiegazioni.

### Opzione NOCLEAR

L'opzione NOCLEAR consente di non alterare le variabili qualora l'operatore risponda con un semplice CR. Es.

```
INPUT NOCLEAR,A,B,C
```

### Opzione LDEV (precedentemente CH)

L'opzione LDEV <exp> consente di utilizzare un dispositivo logico diverso al posto di CON: per le operazioni di ingresso. Sono possibili le seguenti opzioni:

### LDEV 0 - console

Seleziona CON: (default).

### LDEV 1 - input da area fisica specificata

<exp> = 1 provoca la lettura di dati da un'area che deve essere preventivamente indicata con

STORAGE ABS <exp>

dove <exp> è una espressione il cui risultato è l'indirizzo fisico dell'area stessa. Il buffer deve essere predisposto in ASCII, come mostra il seguente esempio.

### Esempio

All'indirizzo 26000:

10,11,12

21,22,23

31,32,33

(ossia in hex: 31 30 2C 31 31 2C 31 32 0D 0A 32 31 2C 32 32 2C 32 33 0D 0A ecc.)

```
10 STORAGE ABS 26000
20 FOR I=1 TO 3
30 INPUT LDEV 5,A,B,C
35 PRINT A,B,C
40 NEXT
```

In esecuzione:

```
10   11   12
21   22   23
31   32   33
```

**LDEV 4 - input "on-the-fly"**

**NOTA IMPORTANTE: Per nuovi programmi preferire l'uso della funzione BIOS (vedi).**

<exp> = 4 per eseguire l'accettazione "al volo" di caratteri dalla tastiera senza arrestare l'esecuzione del programma.

### Esempio

```
10 DIM A$(30)
20 INPUT LDEV 4, PICTURE "!", A$
30 IF A$ = "S" THEN 1000  40 IF INP(123) = 23
THEN 2000
50 GOTO 10
```

Il programma sopra riportato attende la pressione del tasto "S". Nel frattempo monitorizza la porta fisica 123. Notare l'opzione "PICTURE" che evita l'eco dei tasti premuti.

Se si preme il tasto ENTER viene ritornato il carattere '\'. In questo modo è possibile riconoscere il CR dal "nessun tasto premuto".

La condizione di nessun tasto premuto può essere rilevata anche dalla lunghezza della stringa ritornata da INPUT: se questa è zero, nessun tasto risulta premuto.

Un tasto premuto ritorna una stringa di lunghezza unitaria.

### LDEV 5 - input da storage

<exp> = 5 per eseguire l'input da uno "storage" dello pseudofile RAM in congiunzione con la frase STORAGE (vedi). Per eseguire la lettura di uno storage si deve prima posizionare il puntatore interno con la frase

STORAGE <exp>

e quindi eseguire una

INPUT LDEV 5, <var1>, <var2>...

I valori presenti nello storage (scritti in precedenza con una frase PRINT LDEV 5) vengono così rispettivamente assegnati alle variabili <var1>, <var2> ecc. E' compito del programmatore assicurare la corrispondenza di tipi e dimensioni tra la fase di scrittura e quella di lettura.

Notare che un errore "LINEA LUNGA" è dovuto spesso al ricoprimento in fase di scrittura con la PRINT LDEV5 dei vari storage uno con l'altro. In questo modo accade quanto segue:

1) lo storage n è troppo lungo e sconfina nello' n+1;

2) lo storage n+1, se scritto dopo, copre il CR di fine dello storage precedente;

3) cosl di seguito; l'ultimo CR è quello dell'ultimo storage. La INPUT tenta cosl di leggere fino al successivo CR e trova una linea esageratamente lunga.

### LDEV 6 - filtro

L'opzione LDEV 6 determina il filtraggio di tutti i caratteri non numerici esclusi il punto decimale ed il CR.

Libera riproduzione e libero uso

### LDEV 7 -redirezioni su AUX:

L'opzione LDEV 7 redirige l'ingresso dal dispositivo logico AUX:.

### Uscita da INPUT

Oltre al normale CR, altri due caratteri provocano l'uscita dalla frase INPUT. Essi sono denominati convenzionalmente NBESC e CONFIRM. La funzione EXIT(0) ritorna il codice del carattere utilizzato per l'uscita. L'uscita con un carattere o con l'altro non modifica il comportamento di INPUT.

Per default NBESC vale '$' e CONFIRM vale '*'. E' possibile modificare dette assegnazioni con l'opzione PICTURE (vedi sotto). I caratteri correntemente usati per NBESC e CONFIRM non possono ovviamente formare oggetto di normale input.

### Opzione PICTURE

La clausola opzionale PICTURE fissa il formato di ingresso.

La stringa <string> definisce il formato con le seguenti regole:

1) un "9" corrisponde ad un carattere numerico - altri caratteri sono ignorati;

2) una "A" corrisponde ad un carattere alfabetico - altri caratteri sono ignorati;

3) una "X" corrisponde a qualsiasi carattere - nessun carattere viene ignorato.

4) sono accettati solo tanti caratteri quanti ce ne sono nella stringa <string> - altri eccedenti sono ignorati;

Nella stringa di formato <string> è ammesso uno o più prefissi che possono essere:

! - non eseguire eco dei caratteri introdotti

* - esegui eco con asterischi

> - esegui un auto CR quando si arriva alla lunghezza massima della stringa di formato.

/ - fissa il carattere NBESC (vedi sopra) che deve subito seguire

& - fissa il carattere CONFIRM (vedi sopra) che deve subito seguire

La parte della stringa che descrive i caratteri di input deve essere sempre alla fine.

**Esempio**

```
10 INPUT PICTURE "999", N
```

Spiegazione: Accetta solo tre cifre

**Esempio**

```
10 INPUT PICTURE "!",LDEV 4,A$
```

Spiegazione: Accetta un carattere "al volo" senza eco

**Esempio**

```
10 INPUT NOCLEAR,PICTURE "AAAA",B$
```

Spiegazione: Accetta quattro caratteri alfabetici ed esegui auto CR dopo il quarto carattere. Se viene introdotto solo CR non modifica B$.

**Esempio**

INPUT PICTURE ">/%9"

Spiegazione: input di un solo carattere numerico con auto cr e NBESC rappresentato da %.

Esempio pregb

```
....
NBESC = _36       'codice di $
CONFIRM = _42     'codice di *
```

```
....
input " ", picture "/$&*>9", n

#-> WtStart
   let key = bios(K8279, 8)
   if key = CONFIRM then StartEx
   if key = NBESC then MainMenu
goto WtStart
```

### Redirezione dell'ingresso

L'opzione CH consente di redirigere l'ingresso della frase INPUT direttamente da un driver specificato. L'operazione avviene tramite l'assegnazione temporanea del driver richiesto al dispositivo logico AUX: ed eseguendo l'ingresso da esso. Alla fine, il dispositivo AUX: viene ripristinato alla sua assegnazione originale. Il driver deve essere di tipo "character oriented".

### Esempio

```
10 OPEN CH 1, "COM:....", "S 1 D 20 C 21 V
22"
20 PRINT  CH 1, A$
```

### Altre note su INPUT

Il tasto ESC è attivo solo se non disattivato con la frase (vedi):

BREAK 0

La funzione di "backspace" viene attivata dal tasto omonimo, salvo diversa riasse4gnazione eseguita con la frase (vedi):

BACKSP IS <exp>

INPUT usata in modo diretto provoca la segnalazione di un ERRORE DI LINGUAGGIO e non di MODO DIRETTO.

### Inserimento in notazione esponenziale

Il carattere da usarsi per l'introduzione di quantità in notazione esponenziale è "^".

### Esempio

```
123.4^7.
```

# INTERRUPT

### Forma generale

```
INTERRUPT <exp>
```

### Descrizione

Se <exp>=0, INTERRUPT disabilita l'accettazione degli interrupts ad alto livello. Le richieste di interruzione vengono comunque accumulate nel FIFO fino al limite di capacità (cui si ha un messaggio di ERR INT STK OVERFLOW).

Con <exp>=1 il riconoscimento delle interruzioni viene riabilitato e vengono servite tutte quelle eventualmente pervenute durante la disattivazione.

Con <exp>=2 le interruzioni vengono semplicemente ignorate senza procedere al loro accumulo nel FIFO.

### Esempio

```
1000 INTERRUPT 0
```

# LET

### Forma generale

```
LET var=<exp>
```

### Descrizione

la frase LET serve per assegnare ad una variabile un certo valore risultante da una espressione aritmetica. Si noti che in GBASIC la frase LET non pur essere sottintesa come nel BASIC tradizionale.

Libera riproduzione e libero uso

### Esempi

```
10 LET A=7.9

10 LET C = 3 / 8

10 LET C=3-9

10 LET D1=(X-Y)/(X+Y)

10 LET F=SIN(3.21)/COS(A)

10 LET R=1000

10 LET A$=""
```

## LOAD TIMER

### Forma generale

```
1) LOAD TIMER <exp1> WITH <exp2>

2) LOAD TIMER <exp1> WITH <exp2> FREERUN
```

NOTA - esiste anche un comando LOAD (vedi).

### Descrizione

carica il timer <exp1> con il valore <exp2>. <exp1> deve valere 1, 2 o 3 mentre <exp2>, salvo modifica (frase PRESCALE), corrisponde approssimativamente al numero di secondi che devono trascorrere prima dello scatto del timer e quindi prima della esecuzione della routine di servizio prefissata con la frase ON TIMER.. (vedi).

Indipendentemente da ciò che il processore sta eseguendo, dalla esecuzione della frase load timer si innesca un conteggio, cioè, che dopo il numero di secondi prefissati determina lo "scatto" del sottoprogramma di servizio.

Il ritardo, come già detto, è approssimativamente proporzionale al numero di secondi indicati da <exp2>. Con la frase PRESCALE (vedi) è tuttavia possibile variare il

fattore di scala in modo da allungare od accorciare il tempo corrispondente ad ogni unità di <exp2>.

Specificando l'opzione FREERUN si ha la ripartenza automatica del timer, dopo lo scatto, con l'autocaricamento di una costante di tempo <exp2> uguale alla precedente.

La sezione 3 riporta una dettagliata spiegazione della logica di funzionamento dei timers.

Vedi anche le frasi ON TIMER.., RETURN TIMER.., RESET TIMER..,OPEN TIMER..

### Esempio

```
10 ON TIMER 1 GOSUB 100
20 LOAD TIMER 1 WITH 5 FREERUN
30 PRINT "AAAAA"
40 GOTO 30
100 PRINT "----------"
120 RETURN TIMER 1
```

## LPRINT

### Forma generale

```
1) LPRINT "stringa",var,var1..

2) LPRINT "stringa",var,"stringa1",var1..
```

### Descrizione

la frase LPRINT è identica alla frase PRINT ma dirige l'uscita sul dispositivo logico LST: anzichè su CON:/out. Vedi la frase PRINT.

### Esempio

```
10 LPRINT "Raggio ",A,B,C
20 LPRINT "Lunghezza ",L," metri"
```

# NOVRAM

### Forma generale

```
NOVRAM <exp>
```

### Descrizione

Le frasi SAVE, STORAGE, PRINT LDEV 5 sono usate di solito per eseguire la scrittura di informazioni in aree di memoria non volatili.

In alcuni sistemi queste aree possono corrispondere a memoria di tipo EEPROM, caratterizzata in genere da un tempo di scrittura assai maggiore della normale RAM.

Con la frase NOVRAM è possibile aggiungere un ritardo tra la scrittura di un byte ed il successivo, tempo proporzionale ad <exp>.

All'inizializzazione, <exp> è posta a zero, eliminando il ritardo.

Il valore ottimale deve essere ottenuto per tentativi. Su sistemi a 2.5 MHz, per una normale EEPROM 8Kx8 sono usati di solito valori di circa 10.

### Esempio

10 NOVRAM 10

# NEXT

### Forma generale

```
NEXT
```

### Descrizione

indica la fine di un ciclo di FOR. Più cicli di FOR possono essere innestati uno dentro l'altro; per ogni FOR è necessaria una NEXT. L'accoppiamento avviene in modo automatico tra

Libera riproduzione e libero uso

l'ultimo FOR e la prima NEXT (vedi anche frase FOR). E' possibile, opzionalmente, indicare il nome della variabile usata dalla FOR.

Note: se viene incontrata una frase NEXT senza che questa sia stata preceduta da un FOR si ottiene un ERRORE DI FOR..NEXT. Similmente nel caso in cui una frase NEXT risulti mancante.

### Esempio

```
10 FOR I=1 TO 10
20 PRINT
30 NEXT
```

## ON..

### Forma generale

```
1) ON ERROR <linnum>

2) ON ERROR CON:  IS CH <exp>

3) ON INTERRUPT GOSUB <linnum>

4) ON TIMER GOSUB <linnum>

5) ON <exp> GOTO / GOSUB <lista di linnum>
```

La frase ON.. è una frase dichiarativa, vale a dire che non provoca nessuna immediata azione apparente ma predispone delle azioni da compiersi nel caso in cui si verifichino certi eventi.

### Forma 1:

La forma 1 determina il numero di linea cui si deve saltare nel caso in cui si verifichi un qualsiasi errore.

### Forma 2:

La forma 2 serve a riassegnare al dispositivo logico CON:/out ed in il driver di numero specificato da <exp> nel

caso in cui si verifichi un errore per evitare situazioni di stallo. I messaggi di errore sono comunque sempre inviati al dispositivo logico ERR:.

### Forme 3 e 4:

Le forme 3 e 4 stabiliscono invece l'azione da compiersi nel caso in cui avvenga lo scatto di un timer o provenga una interruzione da un dispositivo periferico.

Ricordiamo che ogni interrupt (INTERRUPT o TIMER) ha tre stati possibili

0=inattivo

1=preset

2=sotto servizio

La frase ON INTERRUPT <exp1> GOSUB <exp2> provoca il passaggio dallo stato 0 allo stato 1 (idem per TIMER).

La frase ON INTERRUPT <exp1> (cioè senza il GOSUB susseguente) riporta invece nello stato 0, disabilitando ulteriori interruzioni.

Nella frase ON TIMER, <exp1> può valere 0, 1, 2 o 3, in corrispondenza dei tre timers gestiti. Nella ON INTERRUPT, <exp1> può valere invece 0, 1, 2 o 3.

Si ricordi che il ritorno dal sottoprogramma di servizio deve avvenire con la frese RETURN TIMER <exp1> (oppure RETURN INTERRUPT <exp1>), con <exp1> uguale al numero dell'interrupt interessato, in modo da ripristinare lo stato di PRESET. Diversamente ulteriori interruzioni non verranno servite.

Un timer prima di poter essere usato deve essere aperto con la frase OPEN. Vedere la sezione 3 per dettagli.

### Esempio

```
10 OPEN TIMER 1,"CTC 2 DATA 145"
15 ON TIMER 1 GOSUB 100
```

```
20 LOAD TIMER 1 WITH 5 FREERUN
30 PRINT "AAAAA"
40 GOTO 30
100 PRINT "----------"
120 RETURN TIMER 1
```

Il precedente programma provoca l'emissione ripetuta della stringa AAAAA sullo schermo. Ogni 5 secondi circa, viene inframezzata una stringa di "-".

L'uso della frase ON INTERRUPT implica necessariamente la conoscenza dell'hardware e dei dispositivi periferici che possono eventualmente essere programmati eseguendo un breve programma in EEPROM (scrivibile byte per byte con il debug) eseguibile con la frase CONFIGURE (vedi). Vedi anche le frasi RETURN, CONFIGURE, LOAD TIMER.., RESET TIMER.., OPEN TIMER..

**Forma 5:**

Nella forma 5 si ha il salto ad una frase su più possibili. Ad esempio

10 ON N GOTO 100, 2000, 3000

provoca il salto ad una delle linee in funzione del valore di N. Per N=0 si ha il salto alla prima delle etichette elencate.

# OPEN

**Forma generale**

```
1) OPEN CH <exp>, <string1>

2) OPEN CH <exp>,<string1>, <string2>

3) OPEN TIMER <exp>,<string1>

4) OPEN OSC <exp>, <string>

5) OPEN COUNTER <exp>, <string1>
```

### Descrizione

Scopo della frase OPEN è quello di rendere edotto il kernel della presenza di un driver esterno o di una periferica autogestita. L'operazione svolta dalla frase OPEN è detta apertura.

Nelle forme 1 e 2 svolge le seguenti funzioni:

1) comanda la ricerca del driver di nome indicato con <string1> tra quelli in coda al kernel. Se non lo trova ritorna un messaggio di errore. <Exp1> deve essere compresa tra 0 e 7. La posizione 0 è riservata al driver di console ma può comunque essere usata quando la console non serva. La stringa <string1> deve essere non più lunga di 8 caratteri.

2) crea una voce di direttorio corrispondente al driver richiesto nella posizione <exp>.

3) esegue una chiamata al driver stesso con richiesta della funzione#2 (INIT), per inizializzarlo.

Nel caso della forma 2, la <string2>, detta stringa di configurazione, viene passata al driver che la sfrutta secondo propri criteri. Vedi par.3.8.

Le forme 3, 4 e 5 sono relative alle periferiche autogestite da drivers interni al kernel, rispettivamente timers, oscillatori e contatori. In questo caso <exp> deve essere compresa tra 1 e 3, tranne nel caso degli OSC che possono essere 4. Vedere il paragrafo "Periferiche autogestite" per ulteriori informazioni. Si noti che un OSC non utilizza interrupt.

Un driver può essere aperto una volta sola. Tentando la riapertura si ha un messaggio di errore ERR GIA' OPEN LN.. Un driver aperto può essere chiuso con la frase CLOSE CH (vedi) o con la reinizializzazione del sistema (BOOT, vedi) che chiude tutti i driver ad esclusione di quelli interni "OSC" per evitare di arrestare anche il baud rate generator del canale seriale di console.

Libera riproduzione e libero uso

### Esempi

```
10 OPEN CH 3,"LANDR:", "D 123 C 123 V 123"
20 OPEN OSC 3,"C 2 D 145"
30 OPEN TIMER 1, "C 3 D 147"
```

## OSC

### Forma generale

```
1) OSC <exp1>,<exp2>

2) OSC <exp1>,<exp2> FOR SWITCH
```

### Descrizione

Determina la costante di tempo <exp2> di un oscillatore <exp1>. La frequenza di oscillazione è data da

$$f = 1/ (T_{clock} \times 256 \times \text{<exp2>})$$

con $T_{clock}$ uguale al periodo del clock di sistema. OSC non può essere impartita in modo diretto.

La forma 2 è usata per avviare il Mini Multi Task, un task switcher interno usabile dai drivers.

OSC 1 è di solito utilizzato come baud rate generator per la console.

### Esempio

```
10 OPEN OSC 2,"C 1 D 145"
20 OSC 2, 40 FOR SWITCH
10 OPEN OSC 2,"C 1 D 145"
20 OSC 2, 33
```

Libera riproduzione e libero uso

# OUT

### Forma generale

```
1) OUT <exp1>, <exp2>

2) OUT <exp1>, <exp2>, <exp3>

3) OUT =<exp1>
```

### Descrizione

OUT serve per eseguire l'emissione di singoli valori su una porta di uscita. <Exp2> è il valore da far uscire ed <exp1>il numero della porta.

Se specificato, <exp3> indica che si deve eseguire l'operazione di uscita relativa ad un solo bit, indicato appunto da <exp3>. In questo caso <exp2> potrà valere solo 0 od 1. Questa opzione di uscita di bit singoli può avvenire solo su porte PIO o in modo bufferizzato come sotto illustrato.

<Exp3>=0 indica il bit meno significativo, <exp3>=7 il più significativo. Gli indirizzi delle porte dipendono dalla struttura e dalla implementazione dello specifico sistema target.

Limiti: Sia <exp1> che <exp2> devono essere compresi tra 0 e 255 o ne risulterà un ERR DI LIMITE.

Si noti che sotto GBASIC l'uso della porta di indirizzo 00 è inibito. Scrivere sulla porta 00 ha infatti un significato particolare che varia in funzione del parametro <exp2> come segue:

<exp2> = 0 - normale funzionamento delle porte di uscita.

<exp2> = 1 - le porte di uscita sono delle PIO

<exp2> = 2 - usa il modo bufferizzato. In questo modo è possibile l'out per bit a patto di avere assegnato un buffer di 256 bytes per detto uso. Il buffer si assegna con la frase

OUT= specificando un indirizzo assoluto. Un buffer di memoria può essere facilmente ottenuto come mostrato nell'esempio sottostante.

**Esempi:**

```
10 OUT 128,124
20 OUT 1,40,6


10 REM ricava dim. di un elemento di matrice
20 LET A = INFO(18)

30 REM crea un buffer di 256 bytes
40 REM allocando una matrice
50 DIM B(INT(255/A+1))

60 REM usa buffer mode per le porte uscita
70 OUT 0,2
80 REM indica al basic dove sta il buffer
80 OUT =ADDROF(B(0))-A
```

# OVERPRINT

**Forma generale**

```
OVERPRINT <stg1>,<exp>,<stg2>
```

**Descrizione**

Sovrascrive la stringa <stg1> con la stringa <stg2> a partire dalla posizione <exp>.

# POP

**Forma generale**

```
POP <stringvar>
```

**Descrizione**

Preleva il prossimo dato dal FIFO in cui vengono inseriti i

messaggi provenienti dai drivers. Se questo è vuoto si avrà un errore ERR I STK UF (Interrupt Stack UnderFlow). Vedi i paragrafi 3.7 e 3.10 per ulteriori informazioni..

<Stringvar> è il nome di una variabile stringa che deve essere definita e di dimensione adeguata a contenere il messaggio.

### Esempi

POP A$

## POKE

### Forma generale

```
1) POKE <exp>  [<exp1]

2) POKE  <exp> [<exp1>, <exp2>...]
```

### Descrizione

la frase POKE è utilizzata per scrivere un dato in una cella di memoria ad un certo indirizzo assoluto e nei successivi. <Exp> dà l'indirizzo mentre <exp1>, <exp2> ecc. danno i relativi contenuti; <exp1> viene registrato all'indirizzo <exp>, <exp2> all'indirizzo <exp>+1e cosl via.

Limiti: <exp> deve essere compreso tra 16384 e 65535 (infatti la memoria sotto 16384 è di tipo in sola lettura e non vi possono quindi essere inseriti dei valori). <Exp-n> deve essere compresa tra 0 e 255.

### Esempi

```
10 POKE 20000[0]

10 POKE 20000[0,9,89,234,90,76]
```

POKE eseguiti sull'area EEPROM devono essere seguiti da un loop di ritardo della durata di almeno 10 msec prima di un ulteriore POKE. La durata di tale loop è funzione del clock del sistema.

# PRESCALE

### Forma generale

```
PRESCALE <exp>
```

### Descrizione

Con il clock del processore a 3.7 MHz, il parametro di tempo prefissabile con la frase LOAD TIMER corrisponde approssimativamente ad un secondo per ogni unità. In realtà le interruzioni che il processore riceve sono assai più frequenti che non una per ogni secondo; esse vengono divise da un programma chiamato prescaler che assume un fattore di divisione pari a 56 (cioè una interruzione ogni 56 arriva effettivamente al GBASIC).

Con la frase PRESCALE è possibile assegnare un nuovo fattore di divisione da indicarsi come variare l'unità di misura del tempo. Il valore dell'unità di tempo è determinabile con la formula:

<exp> * 65536 * T

con T=periodo del clock di sistema (infatti con <exp>=56, f<MJ8>clock <DJ0>= 3.6864 MHz (T= 271 ns) si ha che ogni unità corrisponde a 0.9955.. secondi).

### Esempio

```
10 PRESCALE 78
```

# PRINT

### Forma generale

```
1) PRINT <lista di espress. numeriche o
stringa>

2) PRINT LDEV <exp>, <lista di esp. num o
stringa>
```

```
3) PRINT CH <drv>, <lista di espr. num o
stringa> <P1EX>
```

### Descrizione

La frase PRINT serve per determinare l'emissione verso il dispositivo logico CON: di quantità numeriche od alfanumeriche. Le espressioni sono valutate prima di essere emesse.

La seconda forma è utilizzata per scrivere nello pseudofile RAM.

**ATTENZIONE - SONO STATE INTRODOTTE VARIANTI CHE RICHIEDONO LA MODIFICA DEI PROGRAMMI ESISTENTI.**

E' stato variato l'uso della parola chiave CH nelle frasi PRINT ed INPUT. Questa modifica si è resa necessaria per una doverosa riarmonizzazione logica. Infatti la parola "CH" era usata impropriamente ed era ormai limitata solo ad alcuni casi specifici. La nuova logica è la seguente:

**la parola CH** viene usata per la redirezione dell'input o dell'output su un driver. L'espressione che segue dà il numero del driver così come dichiarato nella OPEN CH..

**la vecchia parola CH viene sostituita da LDEV** e serve per redirigere il dispositivo logico (che normalmente è CON:).

Le possibilità sono quindi adesso due:

a) redirigere temporaneamente il dispositivo logico CON:, normalmente usato da PRINT ed INPUT, su un qualunque driver;

b) redirigere temporaneamente le frasi PRINT ed INPUT su un diverso dispositivo logico.

Vedere nel seguito ulteriori spiegazioni.

### Esempio di PRINT

```
5 LET A6=3.678
```

```
10 PRINT "Il risultato è ",A6/3
```

### Separatori

La stessa frase PRINT può essere impiegata per stampare il risultato di più espressioni separando queste ultime con uno dei seguenti separatori:

, (virgola) - salta alla prossima tabulazione (8 caratteri)

; (punto e virgola) - non inserire spazi;

! (esclamativo) - esegui un ritorno a capo / interlinea

### Esempio

```
10 PRINT A,A/4,A-8,B$,CONCAT$(A$,Z$)
20 PRINT "Peso=";K1;" chilogrammi"
30 PRINT "AAAAA" ! "BBBBB"
```

### Funzione TAB(<exp>)

La frase PRINT può utilizzare la funzione TAB() (vedi) per inserire spazi.

### Esempio

```
10 PRINT TAB(30),"VALORE= ",V1
```

(per scrivere VALORE= alla colonna 30)

La funzione TAB() è utile anche per generare diagrammi.

### Esempio

```
10 FOR I=1 TO 24
20 PRINT TAB(I),"*"
30 NEXT
```

Il GBASIC conserva una sola locazione di memoria per memorizzare la posizione corrente di stampa. Per questo motivo, se si commuta il dispositivo logico di uscita, è opportuno eseguire un ritorno alla posizione di stampa 0 prima di eseguire una TAB. Controllo del formato di uscita

Libera riproduzione e libero uso

La frase PRINT consente anche di variare il formato dei dati in uscita. Per fare ciò si usano dei codici preceduti e seguiti dall'operatore %.

### Formattazione dell'uscita

Dalla versione 7.35 è stato integralmente rivisto il modo di formattare l'uscita aggiungendo una notevole flessibilità.

Per formattare l'uscita si deve inserire nella frase print, come un qualsiasi elemento della lista, uno **specificatore di formato** costituito dal carattere % e da una costante stringa; ad esempio

%"I6"

I caratteri dello specificatore di formato sono i seguenti:

**Dn** - fissa la stampa di un numero fisso n di decimali anche se nulli.

**Nn** - fissa la stampa di un numero n di decimali solo se non nulli.

**In** - fissa la stampa di un numero n di cifre della parte intera. Se le cifre più significative sono nulle, al loro posto vengono stampati degli zeri o qualsiasi altro carattere fissabile con lo specificatore F.

**Fc** - fissa il carattere c di riempimento per le cifre a sinistra non significative. Di solito si usa lo spazio, lo zero o l'asterisco

**E** - determina l'uscita in formato esponenziale.

L'effetto degli specificatori è cumulativo e rimane in effetto fino a modifica.

La variazione del formato di uscita della PRINT ha effetto anche su PRINTS e STR$.

Lo specificatore %"I0F0" ripristina il comportamento standard.

### Esempio

```
10 PRINT %"I6D2F*",.....
```

dispone la stampa di 6 cifre intere e 2 decimali con riempimento di asterischi.

### Opzione LDEV (precedentemente CH)

Con l'opzione LDEV è possibile variare il dispositivo logico attraverso cui vengono emessi i dati dalla frase PRINT. Sono ammessi i seguenti valori:

LDEV 0 - console

Seleziona CON: (default).

LDEV 2 - lista

Seleziona LST: (è più semplice usare LPRINT).ù

LDEV 3 - lista

Seleziona ERR:.

LDEV 5 - scrittura negli storages (pseudofile RAM)

La frase PRINT ammette l'opzione LDEV 5 per scrivere dati in uno storage precedentemente selezionato con la frase STORAGE <exp>. Vedere le frasi INPUT e STORAGE per ulteriori informazioni.

La velocità di scrittura è regolabile per consentire l'impiego di EEPROM. Vedi frase NOVRAM.

### Esempio

```
PRINT LDEV 5,A,B,C$ LDEV 7 - ausiliario
```

Seleziona AUX: (meglio usare DISPLAY).

### Redirezione dell'uscita

L'opzione CH consente di redirigere l'uscita della frase

PRINT direttamente su un driver specificato. L'operazione avviene tramite l'assegnazione temporanea del driver richiesto al dispositivo logico AUX: ed eseguendo l'uscita su di esso. Alla fine, il dispositivo AUX: viene ripristinato alla sua assegnazione originale. Il driver deve essere di tipo "character oriented".

### Esempio

```
10 OPEN CH 1, "COM:....", "s 1 D 20 C 21 V
22"
20 PRINT CH 1, "ABCDEF"
```

## PRINTS

### Forma generale

```
PRINTS <stringvar>, <list>
```

### Descrizione

La frase PRINTS è del tutto identica alla PRINT ma l'uscita è diretta nella stringa di nome <stringvar> che deve essere definita e di dimensione adeguata. La <list> deve essere terminata con un punto e virgola per evitare l'emissione di CR/LF.

### Esempio

```
10 PRINTS A$,A,B,C;
```

## READ

### Forma generale

```
1) READ <var1>,<var2>...

2) READ CH <exp1>,<exp2>,<string>

3) READ  CH <exp1>,<exp2>,=<var>
```

### Descrizione

### Forma 1

La frase READ assegna alla variabile var1 il primo valore letto dall'insieme delle frasi DATA (vedi), alla seconda il secondo e cosl via. Alla fine il puntatore di lettura viene lasciato posizionato dopo l'ultimo elemento letto. La successiva frase READ partirà quindi da esso per le successive assegnazioni. La frase READ è utile per eseguire lunghi caricamenti di variabili. Desiderando riposizionare all'inizio dell'insieme DATA il puntatore, si userà la frase RESTORE (vedi).

Se si tenta di leggere variabili quando non ci sono più DATA disponibili si ottiene un ERR READ.

### Esempio

```
10 DATA 12.4,45,8
20 READ A,B,C
```

### Forma 2

READ con opzione CH è usata per ricevere blocchi di dati da un driver. <Exp1> è il numero del driver di origine, che deve essere aperto e deve prevedere la comunicazione di tipo BLOCK. I dati vengono ricevuti nella stringa <string>. Il parametro <exp2> viene passato senza modifiche al driver ed il suo uso differisce a seconda dei casi. Frequentemente è usato, quando abbia senso, come numero del record. L'operazione richiesta al driver è la #1 (BLOCK READ).

### Esempio

```
10 OPEN CH 3,"CARD:...","D 100"
20 READ CH 3, 0, Q
```

### Forma 3

Prima della v.7.4x, tutte le scritture della WRITE avvenivano solo come stringhe: pur accettando la write anche una espressione numerica, essa veniva convertita in stringa e passata al driver come tale (in pratica, come una PRINT con

uscita sul driver o file).

Adesso è possibile specificare che un valore numerico deve essere convertito in intero unsigned a 16 bit e passato al driver senza altra conversione. Lo si ottiene preponendo un "=" all'espressione da scrivere o alla variabile da leggere.

**Esempio:**

```
10 WRITE CH 1, 3, =A1
10 READ CH 1, 3, =A2
```

Il numero del record è adesso gestito a 16 bit.

# REM

**Forma generale**

```
REM testo
```

**Descrizione**

la frase REM serve per aggiungere commenti all'interno di un programma. Il testo viene completamente ignorato.

Le frasi REM accrescono il tempo di esecuzione di un programma e aumentano la sua occupazione di memoria. Poiché in genere viene utilizzato il precompilatore, che non produce codice oggetto per i commenti, la frase REM è di solito di impiego limitato.

**Esempio**

```
10 REM PROGRAMMA CONTROLLO POMPE ARIA
20 REM VERSIONE 1.0 DEL 12/7/89
```

# RESET

**Forma generale**

```
1) RESET OSC <exp>
```

```
2) RESET TIMER <exp>

3) RESET COUNTER <exp>

4) RESET INTERRUPT

5) RESET INTERRUPT CLEAR
```

### Descrizione

### Forme 1, 2 e 3

La frase RESET è usata per arrestare il funzionamento di un timer o di un oscillatore o per azzerare il valore di un contatore. . <Exp> indica il numero dell'unità interessata. RESET OSC non può essere impartita in modo diretto.

Vedi anche ON TIMER, LOAD TIMER, PRESCALE, RETURN TIMER, OPEN, COUNTER, OSC, COUNT()

### Esempi

```
100 RESET OSC 1
110 RESET TIMER 2
120 RESET COUNTER 1
```

### Forma 4 e 5

Nella forma 4 azzera il contenuto dei FIFO dell'interrupt (FIFO delle richieste e FIFO dei messaggi). Tutti gli interrupts pendenti sono cancellati.

Nella forma 5, provoca il reset di tutto il sistema di interruzione. Ogni inizializzazione deve essere ripetuta dopo questa operazione.

Leggere attentamente il paragrafo 3.10.

## RESTORE

### Forma generale

```
RESTORE
```

### Descrizione

la frase RESTORE serve per riposizionare il puntatore della frase READ all'inizio dell'insieme DATA. Vedi anche frase DATA.

### Esempio

```
10 DATA 12,34,45,67
20 FOR I=1 TO 4
30 READ A(I)
40 NEXT
50 RESTORE
60 READ X,Y,Z,W
70 PRINT A(1),A(2),A(3),A(4),X,Y,Z,W

RUN  12  34  45  67  12  34  45  67
```

## RETURN

### Forma generale

```
1) RETURN

2) RETURN TIMER <exp>

3) RETURN INTERRUPT <exp>
```

### Descrizione

serve per ritornare alla frase successiva all'ultimo GOSUB (vedi) eseguito.

Il ritorno da un sottoprogramma eseguito in seguito ad una frase ON TIMER.. o ON INTERRUPT (vedi) deve avvenire con la frase RETURN TIMER <exp> (oppure RETURN INTERRUPT ), con <exp> uguale al numero dell'interrupt interessato, in modo da ripristinare lo stato di preset. Viceversa ulteriori interruzioni di quel codice non saranno servite.

Se nessuna frase GOSUB era stata precedentemente eseguita si ha un messaggio ERR DI FOR..NEXT.

### Esempio

```
10 GOSUB 1000
20 PRINT A
30 STOP
1000 A=787878
1010 RETURN
```

# SIOSET

### Forma generale

```
1) SIOSET <ctlp>, <dbits>,<par>,<stopbits>

2) SIOSET CH <exp>, <dbits>,<par>,<stopbits>
```

### Descrizione

Con SIOSET è possibile variare i parametri di comunicazione di una SIO. Nella forma 19 si indica l'indirizzo fisico della porta controllo della SIO. Nella forma 2 si indica il numero del canale. Questa seconda forma è ammessa solo nel caso in cui il driver accetti di rispondere alla nuova funzione driver #7 (vedi). DIversamente si ha un DRV ERR.

I parametri sono espressioni aritmetiche con il seguente significato:

<ctlp> - numero della porta controllo

<exp> - numero del canale associato al driver

<dbits> - numero dei bit di dati (7 od 8)

<par> - parità - 0 = no; 1 = dispari; 2 = pari

<stopbits> - numero dei bit di stop (1 o 2)

I registri della SIO non sono leggibili. Per questo motivo, SIOSET può alterare alcuni parametri programmati dal driver o dall'utente e precisamente:

DTR - viene sempre azzerato

RTS - viene sempre settato

AUTOENABLE - viene sempre disabilitato

In funzione di ciò, l'utente valuterà la convenienza di usare la SIOSET, eventualmente riprogrammando i parametri che devono assumere diversi valori con la frase OUT.

Quando la generazione del clock di comunicazione della SIO venga fatta con un CTC, il baud rate può essere variato con la frase OSC.

### Esempio

```
SIOSET CH 1,7,2,1
```

(7 bit - parità pari - 1 stop)

## STOP

### Forma generale

```
1) STOP

2) STOP ERROR
```

### Descrizione

quando in un programma viene incontrata la frase STOP l'esecuzione si arresta e compare il messaggio:

BRK LN <linnum>

dove <linnum> è il numero della linea dove si trova lo STOP. Il controllo ritorna all'operatore.

La seconda forma resetta anche i dispositivi logici al valore di default. Questo evita di trovarsi con la tastiera disabilitata al momento del break.

### Esempio

```
450 STOP
```

# STORAGE

### Forma generale

```
1) STORAGE =<exp>

2) STORAGE <exp>

3) STORAGE ABS <exp>

4) STORAGE <exp>,> <lista var>

5) STORAGE <exp>,< <lista var>

6) STORAGE SYS
```

### Logica degli storages.

Una area riservata e non azzerata al reset, denominata SA (Save Area), rende possibile registrare informazioni in modo abbastanza simile a quello usato nel BASIC standard per i files. In GBASIC è disponibile un unico pseudo file dove i dati, anche non omogenei, sono registrati in records detti storages. Uno storage può contenere qualsiasi tipo di dato, alfanumerico o numerico. E' compito del programmatore provvedere alla oculata gestione degli storages e dei loro contenuti.

In considerazione che la memoria usata per registrare lo pseudo file potrebbe essere di tipo EEPROM,la velocità di scrittura è variabile con la frase NOVRAM (vedi).

Le informazioni sono registrate e rilette dagli storages secondo due possibili formati:

**formato ASCII** - usando le frasi PRINT LDEV 5 ed INPUT LDEV 5

**formato fast** - usando la frase STORAGE nelle forme 4 e 5. Questo formato è disponibile solo per le variabili numeriche.

### Uso del formato ASCII

Le informazioni sono registrate con la frase PRINT LDEV 5

(vedi).

### Esempio

```
PRINT LDEV 5,A,B,C$
```

registra nella SA il valore delle variabili A, B e C$ (al posto dei nomi di variabili possono essere usate delle espressioni). La rilettura avviene poi con una frase INPUT LDEV 5 purché ovviamente si rispetti l'ordine di rilettura, anche usando variabili di nome diverso. Per ogni quantità numerica registrata si dovrà usare una variabile numerica per la rilettura e cosl per quelle alfanumeriche. Ad esempio

```
10 PRINT LDEV 5,1.23,4567,"ABCDEF"
20 INPUT LDEV 5,J,K,F$
```

Il separatore usato nella frase PRINT LDEV 5 deve necessariamente essere la virgola (,). Sono attivi i controlli di formato come per la frase PRINT (vedi).

Come accennato, a differenza di quanto avviene in molti linguaggi per i records, la struttura dello storage non è rigida: in ogni storage possono essere immagazzinate quantità eterogenee, purché ovviamente si tenga conto al momento della rilettura della loro tipologia.

L'accesso al singolo storage è diretto: basta far precedere la frase PRINT o INPUT dalla frase

STORAGE <exp>

per posizionarsi sullo storage di numero pari alla espressione <exp>.

Come default gli storage hanno lunghezza di 32 bytes, ma accettano informazioni di lunghezza anche superiore; in questo caso ovviamente viene utilizzato anche lo storage (o gli storages) successivi e deve essere cura del programmatore evitare letture da questi ultimi.

### Dimensione degli storages

La dimensione degli storages (di tutti quanti) pur essere

modificata con la frase

STORAGE=<exp>

dove <exp> fornisce il numero da assumere come lunghezza (notare il segno di uguale in più rispetto alla STORAGE <exp>). Il suo valore massimo è 256.

Il numero massimo di storages disponibili dipende dalla loro lunghezza e dal sistema target (vedi sezione 3) ma non può in ogni caso essere maggiore di 65536. E' ammesso lo storage di numero 0.

Qualora si tenti di accedere ad uno storage che vada oltre i limiti della memoria disponibile si otterrà un errore di ERR FUORI MEMORIA.

Le operazioni di scrittura e lettura non alterano il puntatore interno, per cui non è necessario provvedere ad un nuovo posizionamento con STORAGE quando si desidera riscrivere un dato appena letto.

### Esempio (ASCII)

```
10 STORAGE = 60
15 DIM A$(60)
20 STORAGE 0
30 PRINT LDEV 5,"MARIO ROSSI - VIA VERDI 128
BOLOGNA"
40 STORAGE 1
50 PRINT LDEV 5,"GIUSEPPE BIANCHI -
VIA......."
 .... ecc.
1000 INPUT "Codice? ",N
1010 STORAGE N
1020 INPUT LDEV 5,A$
1030 PRINT A$
```

### Uso del formato fast

Le frasi PRINT ed INPUT hanno delle notevoli capacità operative (valutano espressioni, controllano i formati ecc.) ma proprio per questo risultano piuttosto lente. Per conseguire una maggiore velocità operativa ed una migliore

compattezza di codice, è stato introdotto il modo "fast" (veloce) che consente di registrare o rileggere una serie di variabili numeriche con una sola frase. I formati 4 e 5 ne danno la forma generale.

Anche variabili stringa possono essere utilizzate come sorgente o destinazione dei dati, anche più di una per storage.

Ogni variabile numerica registrata nel formato fast occupa 7 bytes. Più variabili possono essere registrate nello stesso storage.

La posizione della freccia ricorda il senso del trasferimento: "<" verso lo storage, ">" verso le variabili.

### Esempio

```
10 STORAGE = 60
20 INPUT N,M
30 STORAGE N,<M 'REGISTRA IL VALORE M NELLO
ST.N
 .... ecc.  1000 INPUT "Codice? ",N
1010 STORAGE N,>M 'RILEGGE IL VALORE M
1020 PRINT M
```

### STORAGE ABS

La forma STORAGE ABS serve per posizionare il puntatore di lettura delle frasi INPUT/PRINT LDEV 1 (vedi) ad una specifica locazione di memoria. In questo modo è possibile leggere dati da una EPROM o scriverli in una RAM.

### Lo storage di configurazione (STORAGE SYS)

Dalla v.7.4x è stata aggiunta la possibilità di creare uno speciale storage di configurazione per salvare parametri di sistema.

In esso al momento è possibile registrare due parametri: la dimensione di memoria da assegnare agli storages ed il baud rate generator per la console nei sistemi muniti di batteria tampone.

Libera riproduzione e libero uso

Si accede a questo storage, denominato "storage sys", con le consuete frasi. Il posizionamento avviene con STORAGE SYS mentre la scrittura e la lettura avvengono con WRITE e READ LDEV 5, nel solito modo.

Lo storage sys viene allocato in una area apposita e non ruba quindi spazio ai normali storages. Il sistema verifica quando necessario il contenuto dello storage sys; se vi trova un contenuto valido lo utilizza, viceversa lo ignora. In questo modo è possibile conservare il modo di funzionamento standard anche nei sistemi senza batteria.

Si osservi però che registrando dei parametri errati potrebbe diventare impossibile la loro modifica se non staccando la batteria.

Lo storage sys si ritiene valido quando contiene all'inizio la stringa "SYS:".

I parametri utilizzabili sono al momento:

**STOR:xxK** - dove xx è un numero, obbligatoriamente di 2 cifre, che indica il numero di k bytes da riservare per gli storages. Questo parametro è considerato solo al momento della inizializzazione ed è quindi influente solo dal successivo BOOT o reset.

**BRG:n** - dove n è un numero (compreso tra 1 e 255) che esprime la costante da programmare nel CTC. Con clock a 4.916 MHz si ha 1=19200 baud, 2=9600, 4=4800 ecc. Questo parametro è considerato solo al momento della inizializzazione ed è quindi influente solo dal successivo BOOT o reset. Non è indispensabile indicare entrambi i parametri, ma se indicati devono essere riportati nell'ordine.

**Esempio:**

```
10 STORAGE SYS
20 PRINT LDEV 5,"SYS: STOR05K BRG:2"
```

Libera riproduzione e libero uso

# TIMDAT:

**Forma generale:**

```
TIMDAT: <exp1> IS CH <exp2>
```

**Descrizione:**

assegna al dispositivo logico TIMDAT: la periferica fisica gestita dal driver <exp2>. TIMDAT: è utilizzato per le frasi e funzioni TIME$ e DATE$.

La frase AUX: viene utilizzata anche per riassegnare il dispositivo logico TIMDAT: ad un'altro driver.

Esempio

```
10 OPEN CH 2, "TIMDAT:","D 45"
20 TIMDAT: IS CH 2
30 PRINT DATE$()," ",TIME$()
```

# TIMER

**Forma generale**

```
TIMER <exp1>,<exp2>
```

**Descrizione**

attende un tempo prefissato pari ad <exp2>, espresso in 10 msec /unità. Prima di usarla deve essere stato aperto un timer di numero <exp1>.

**Esempio:**

```
10 OPEN TIMER 1, "C 120"
.....
100 TIMER 1,100
```

# TIME$(x$)

### Forma generale

```
TIME$(<string>)
```

### Descrizione

regola l'ora dell'orologio (dispositivo logico TIMDAT:) sul valore di <string> che deve avere il formato oo:mm:ss. Non vengono eseguiti controlli sulla stringa tranne che per la sua lunghezza. I caratteri ":" usati come separatori possono essere sostituiti con altri caratteri. Non confondere con la funzione TIME$().

L'operazione viene eseguita mediante la funzione #0 del driver (WRITE BLOCK) sul record #1.

### Esempio

TIME$("18:55:00")

# WRITE

### Forma generale

```
1) WRITE CH <exp1>,<exp2>,<string>

2) WRITE CH <exp1>,<exp2>,=<var>
```

### Descrizione

### Forma 1:

WRITE è usata per trasmettere blocchi di dati ad un driver. <Exp1> è il numero del driver di destinazione, che deve essere aperto e deve prevedere la comunicazione di tipo BLOCK. I dati vengono da passare sono contenuti nella stringa <string>. Il parametro <exp2> viene passato senza modifiche al driver ed il suo uso differisce a seconda dei casi. Frequentemente è usato, quando abbia senso, come numero del record. L'operazione richiesta al driver è la #0 (BLOCK

WRITE).

### Esempio

```
10 OPEN CH 3,"CARD:...","D 100"
20 WRITE CH 3, 0, Q
```

### Forma 2

Vedi frase READ.

## WRITE

### Forma generale

```
WRITEC <exp1>,<exp2>
```

### Descrizione

Invia il carattere <exp2> al driver <exp1> chiamando direttamente la funzione driver n.4 WRITEC.

# Funzioni

## &H(x$)

### Forma generale

```
&H(<string>)
```

### Descrizione

ritorna un numero convertendo la stringa esadecimale <string> (che deve essere composta solo da cifre esadecimali).

### Esempio

```
10 PRINT &H("3A")
```

## ADDROF(n)

### Forma generale

```
ADDROF(<var>)
```

### Descrizione

ritorna l'indirizzo fisico della variabile numerica di nome <var>. Non opera su variabili stringa.

### Esempio

```
10 LET N1=2.45
20 PRINT ADDROF(N1)
```

## AND(m,n)

### Forma generale

```
AND (<exp1>,<exp2>)
```

### Descrizione

ritorna l'AND logico a 16 bit dei valori risultanti da <exp1> ed <exp2>.

### Esempio

```
10 PRINT AND(10,20)
```

## ARG(x)

### Forma generale

```
ARG(<exp>)
```

### Descrizione

ARG() serve per predisporre il passaggio di un parametro (argomento della funzione) ad una routine dell'utente in linguaggio macchina.

Il valore passato verrà ritrovato nella coppia di registri BC. La chiamata della routine avviene con la funzione CALL (vedi).

### Esempio

```
LET A=ARG(45)
```

## ASC(x$)

### Forma generale

```
ASC(<string>)
```

### Descrizione

Ritorna il codice ASCII del primo carattere della stringa <string>.

### Esempio

```
10 PRINT ASC(A$)
```

## ABS(x)

### Forma generale

```
ABS(<exp>)
```

### Descrizione

ABS() ritorna il valore assoluto della espressione , cioè se <exp> è >=0 ritorna 1, viceversa -1.

### Esempi

```
PRINT ABS(9876.8)

 9876.8

PRINT ABS(-9876.8)

 9876.8
```

# BIOS(x)

### Forma generale

```
BIOS(<drv>,<functcode>)
```

### Descrizione

La funzione BIOS() consente di richiedere ad un driver <drv> di eseguire una funzione <n> e di restituire il valore da esso ritornato.

Questa funzione risulta particolarmente utile per accedere a drivers specifici che possono eseguire numerose funzioni.

Non tutte le funzioni possono essere richiamate con BIOS ma solo quelle che non ritornano parametri o che ritornano un parametro nel registro E. La chiamata deve inoltre avere un significato logico.

Con il driver standard di console, hanno particolare utilità le seguenti funzioni:

**8 (GETCH)** - ritorna prossimo carattere dal buffer, se presente, oppure 0

**9 (FLUSH)** - azzera il buffer dei caratteri ricevuti

**10 (CHCNT)** - ritorna il numero dei caratteri presenti nel buffer

L'uso della funzione GETCH è raccomandata in sostituzione della INPUT LDEV 4. Vedi esempio seguente.

### Esempi

```
10 LET N=BIOS(0,8)
20 IF N=0 THEN 10
20 LET A$=CHR$(n)
```

invece di

```
10 INPUT LDEV 4, A$
20 IF LEN(A$)=0 THEN 10
```

## CALL(x)

### Forma generale

CALL(<exp>)

### Descrizione

Chiamata ad una subroutine dell'utente in linguaggio macchina. <Exp>dà l'indirizzo della routine.

E' consigliabile, all'ingresso della subroutine, salvare lo SP.

La routine dell'utente potrà ritornare un valore, che sarà passato poi al GBASIC dal valore della funzione, lasciandolo nel registro HL.

### Esempi

```
10 PRINT CALL(25000)

10 LET A=CALL(21987)

10 ARG 200       'passa il val. 200 in BC
20 LET A=CALL(25000)   'chiama la sub a 25000
30 PRINT A             'A contiene 1330



          ORG  25000

INGRESSO: LD   (TEMP),SP
          LD   SP,SPUSER ;BC CONTIENE 200
```

```
....
LD   HL,(XYZ)  ;XYZ CONTIENE 1330
LD   SP,(TEMP)
RET
```

Vedi anche frase ARG.

# CHR$(x)

### Forma generale

```
1) CHR$(<exp>)

2) CHR$($<exp1>)
```

### Descrizione

Nella forma 1 ritorna la stringa di un solo carattere corrispondente al codice ASCII del valore <exp>. Nella forma 2 opera come nella forma 1, con la differenza che i caratteri di controllo vengono esplicitati con il loro nome simbolico: es. ETX, STX ecc.

### Esempio

```
10 LET G$=CHR$(65)
20 PRINT G
```

provoca la stampa di una lettera "A".

# CKDAT(x$)

### Forma generale

```
CKDAT(<string>)
```

### Descrizione

CKDAT() verifica che la stringa <string>sia una data valida. In caso positivo ritorna 1, diversamente 0.

<String> deve avere il seguente formato:

data: GG/MM/AA

e devono essere indicati anche gli zeri (es. 04 e non 4).

Il giorno 29/02 viene considerato valido indipendentemente dall'anno.

## CKSUM(x$)

### Forma generale

```
CKSUM(<string>)
```

### Descrizione

CKSUM() esegue lo XOR ad 8 bit dei caratteri che compongono la stringa <string> e ne ritorna il risultato.

### Esempio

```
10 PRINT CKSUM(A$)
```

## CKTIM(x$)

### Forma generale

```
CKTIM(<string>
```

### Descrizione

CKTIM() verifica che <string> sia una ora valida. In caso positivo ritorna 1, diversamente 0.

<String> deve avere il seguente formato:

data: OO:MM:SS

e devono essere indicati anche gli zeri (es. 04 e non 4).

# CONCAT$(x$,y$)

### Forma generale

```
CONCAT$(<string1>,<string2>)
```

### Descrizione

concatena due stringhe <string1> e <string2> per formarne una nuova.

### Esempio

LET A$=CONCAT$(B$,"PIPPO")

Nominalmente accetta come argomenti solo costanti e/o variabili; tuttavia è stata provata con esito positivo in varie circostanze anche in combinazioni più complesse. Si consiglia di provare caso per caso. Gli esempi sotto riportati sono stati collaudati con esito positivo:

```
10 PRINT SUBSTR$("123456789",3,4)
20 PRINT CONCAT$("ABC","123")

10 DIM A$(20),B$(20)
20 LET A$="ABC"
30 LET B$="123"
40 PRINT CONCAT$(A$,B$)
50 PRINT CONCAT$(A$,"ZZZ")
60 PRINT HEX$(100)
70 PRINT CONCAT$(A$,HEX$(100))
80 LET B$=CONCAT$(A$,HEX$(100))
85 PRINT B$
90 LET B$=CONCAT$(B$,B$)
100 PRINT B$
105 LET A$=CONCAT$(B$,SUBSTR$(B$,2,3))
106 PRINT A$
110 PRINT SUBSTR$("1234567890",3+3,5)
```

Libera riproduzione e libero uso

# COUNT(x)

### Forma generale

```
COUNT(<exp>)
```

### Descrizione

Ritorna il valore attuale del contatore <exp>. <Exp> deve essere compreso tra 1 e 3.

Il valore del conteggio accumulato non viene inizializzato all'accensione, in modo da consentire la sua lettura anche dopo una perdita di alimentazione (in caso di RAM non volatile). E' quindi necessario eseguire sempre una RESET COUNTER prima dell'uso.

### Esempio

```
10 OPEN COUNTER 1,"C 1 D 120"
20 COUNTER 1,1
30 PRINT COUNT(1)
40 GOTO 30
```

# COS(x)

### Forma generale

```
COS(<exp>)
```

### Descrizione

COS() ritorna il coseno della espressione . <Exp> deve essere in radianti.

### Esempi

```
10 D=COS(3*Y)
10 D=COS(A/B)
```

# DATE$()

### Forma generale

```
DATE$()
```

### Descrizione

ritorna una stringa del formato GG:MM:AA contenente la data del momento. Essa viene determinata da un driver esterno che deve essere precedentemente aperto con la frase OPEN (vedi). Il driver deve essere dichiarato come quello che svolge la funzione di data/ora con la frase TIMDA:. (vedi).

La funzione viene ottenuta mediante una chiamata al driver con codice #1 (BLOCK READ) e numero di record uguale a 2.

### Esempio

```
10 OPEN CH 4,"CCAL:...","D 20"
20 TIMDAT: IS CH 4
30 PRINT DATE$()
```

# DIAG(n)

### Forma generale

```
DIAG(<exp>)
```

### Descrizione

**Con argomento 0**, ritorna l'esito del controllo iniziale eseguito al momento del reset generale. Il codice ritornato può essere:

1 - nel caso in cui il controllo statistico della RAM abbia rilevato una differenza di contenuti rispetto all'ultimo spegnimento.

2 - nel caso di alimentazione mancata durante un segmento

critico (par.3.18).

4 - nel caso di una locazione di RAM difettosa

Questa funzione ha senso solo se è implementato un sistema di gestione della mancanza di alimentazione.

Eseguendo un BOOT senza avere compiuto lo spegnimento (e quindi senza avere eseguito l'operazione statistica di controllo) si ha normalmente un a condizione di DIAG(0)=1. Nel caso che gli errori di cui sopra si verifichino contemporaneamente il codice di errore è pari alla loro somma (1+2=3).

**Con argomento 1**, ritorna 0 in caso di power-up o un numero maggiore di zero in caso di reset manuale (pressione del tasto reset). Il numero ritornato indica quanti reset manuali sono stati eseguiti dall'ultimo power-up fino ad un massimo di 254.

Questa funzione è molto utile per non eseguire routines diagnostiche al reset manuale.

### Esempio pregb

```
#-> DiagSys
  if diag(1) > 0 then return
  .....
```

## DRVST(x),DRVST(x,y)

### Forma generale

```
1) DRVST(<exp>)

2) DRVST(<exp1>,<exp2>)
```

### Descrizione

Ritorna lo stato del driver <exp1> mediante chiamata della funzione #6 (STATUS). Il parametro <exp2>, opzionale, viene passato integralmente al driver.

### Esempio

```
10 PRINT DRVST(2)
```

## EXIT(0)

### Forma generale

```
EXIT(0)
```

### Descrizione

Ritorna il codice del carattere utilizzato per uscire dall'ultima frase INPUT eseguita. Vedi frase INPUT "Uscita da INPUT"

### Esempio

Vedi INPUT.

## HEX$(x)

### Forma generale

```
HEX$(<EXP>)
```

### Descrizione

ritorna una stringa contenente il valore di <exp> espresso in esadecimale.

### Esempio

```
10 PRINT HEX$(120)
```

## INFO(x)

### Forma generale

```
INFO(<exp>)
```

### Descrizione

ritorna alcuni parametri di sistema. La selezione del parametro avviene in base al valore di <exp> come dalla sottostante tabella:

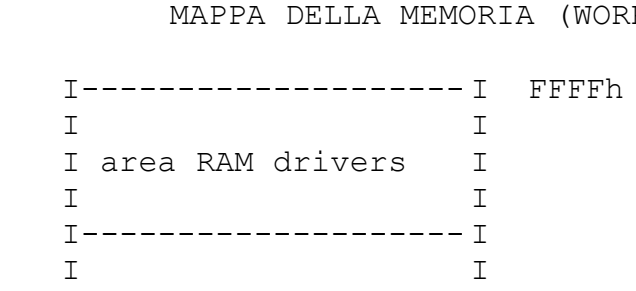
| <exp> | mnemonico | descrizione |
|---|---|---|
| 0 | BOFA | prima locazione programma |
| 1 | EOFA | ultima locazione programma (dopo iniziano matrici/str) |
| 2 | MATA | ultima locazione matrici/stringhe |
| 3 | STB | |
| 4 | MEMTOP | massima locazione in uso (v. anche BLANK) |
| 5 | VER | versione corrente |
| 6 | IBUF | prima locazione buffer ingresso tastiera |
| 8 | NOVST | prima locazione area RAM non modificata al rest |
| 9 | | dimensione complessiva dell'area degli storages |
| 10 | | numero di interrupts pendenti |
| 15 | SAPOINT | SAPOINT - puntatore all'area usata da PRINT ed INPUT LDEV 5. |
| 16 | FREEWS | memoria libera (workspace, ossia area per programma o variabili |
| 18 | FPSIZE | dimensione di una variabile numerica |
| | | |

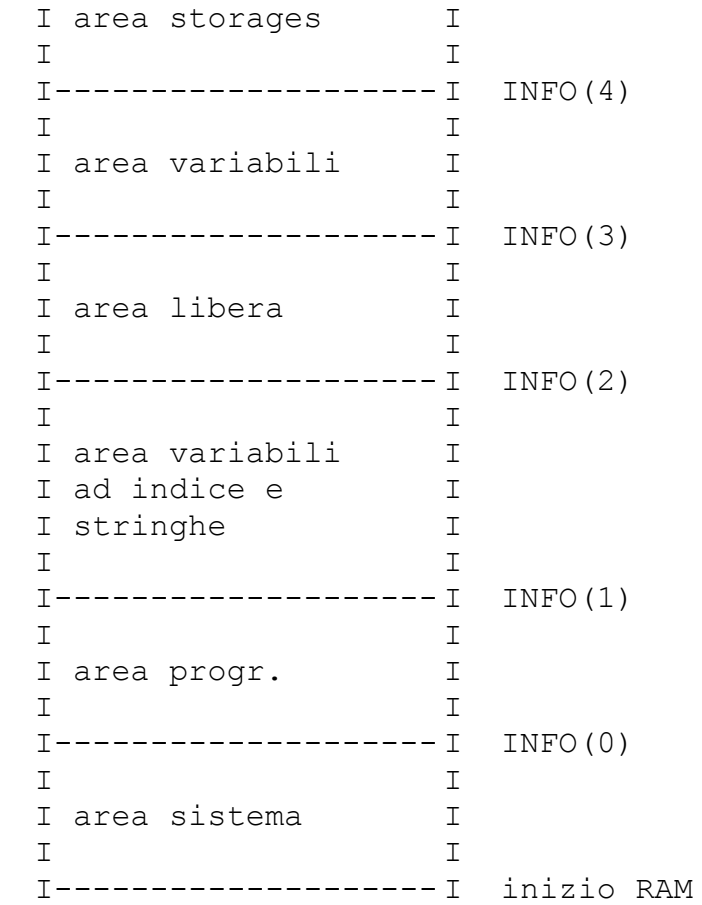
A pagina seguente è riportata la struttura della mappa della memoria relativamente al segmento della WS principale.

Si noti che

PRINT INFO(3) - INFO(2)

fornisce indicazione sulla quantità di memoria ancora disponibile.






# INP(x) INP(x,y)

### Forma generale

```
1) INP(<exp1>

2) INP(<exp1>,<exp2>)
```

### Descrizione

INP(<exp1>) ritorna il valore del dato in ingresso alla porta<exp1>.

Se specificato, <exp2> indica che si deve eseguire l'operazione di ingresso da un solo bit, indicato appunto da . In questo caso il valore ritornato potrà valere solo 0 od 1. <exp2>=0 indica il bit meno significativo, =7 il più significativo.

Gli indirizzi delle porte dipendono dalla struttura e dalla implementazione dello specifico sistema target.

<exp> deve corrispondere ad una porta fisica realmente implementata.

### Esempio

```
10 PRINT INP(241)
```

## INT(x)

### Forma generale

```
INT(<exp>)
```

### Descrizione

INT() ritorna la parte intera di <exp>, cioè la parte a monte del punto decimale.

### Esempi

```
PRINT INT(1)

 1

PRINT INT(1.2)

 1

PRINT INT(0.2)

 0
```

## LEN(x$)

### Forma generale

```
LEN(<string>)
```

### Descrizione

ritorna la lunghezza della stringa <string>. Si faccia attenzione alla distinzione tra lo spazio riservato per una stringa e la sua lunghezza.

### Esempio

```
10 DIM A$(100)
20 LET A$="ABC"
30 PRINT LEN(A$)
```

dà per risultato 3 (cioè lo spazio di A$ è 100 e la lunghezza, dopo l'esecuzione della frase 20, è di 3).

## NGET$(x$)

### Forma generale

```
NGET$(<string>)
```

### Descrizione

con questa funzione è possibile editare stringhe alfanumeriche con una tastierina solo numerica. La stringa <string>viene emessa sul dispositivo logico AUX: (di solito un display LCD). Con i tasti S e D si muove il cursore a sinistra e a destra. Con + e - si incrementa o si decrementa secondo la sequenza ASCII il carattere dove è posizionato il cursore. (ad esempio: incremento C, D, E..; decremento Z, Y, X ecc.) Con CR si accettano le modifiche effettuate e si ritorna il valore della stringa modificata.

I tasti di comando sono acquisiti dal dispositivo logico CON:/in..

### Esempio

```
10 LET B$=NGET$(B$)
```

# NIBBLES(x$)

### Forma generale

```
NIBBLES(<string>)
```

### Descrizione

Ritorna un valore (compreso tra 0 e 255) calcolato sulla base dei primi due caratteri della stringa <string> che devono avere codice ASCII compreso tra 30h e 3Fh. Il calcolo viene effettuato come segue:

risultato = (car1 and 0Fh) x 16 + (car2 and 0Fh).

### Esempi

```
10 PRINT NIBBLES ("74")
```

Risultato: 116

```
10 PRINT NIBBLES ("7?")
```

(il codice di "?" è 3Fh) Risultato: 127

# NOT(x,y)

### Forma generale

```
NOT(<exp>)
```

### Descrizione

ritorna il complemento, a 16 bit, del valore risultante da <exp> convertito in intero.

# OR(x,y)

### Forma generale

```
OR(<exp1>,<exp2>)
```

### Descrizione

ritorna l'OR logico a 16 bit dei valori risultanti da <exp1> ed <exp2> convertiti in interi.

## PEEK(x)

### Forma generale

```
PEEK(<exp>)
```

### Descrizione

PEEK() ritorna il contenuto della cella di memoria all'indirizzo <exp>.

<Exp> deve essere compreso tra 0 e 65535 e deve ovviamente corrispondere a memoria realmente installata.

### Esempio

```
10 IF PEEK(21234)=0 THEN A=89
```

## RND(x)

### Forma generale

```
RND(<exp>)
```

### Descrizione

RND() ritorna un valore casuale compreso tra 0 ed 1.

### Esempi

```
PRINT RND(0)

  .098723

 PRINT RND(0)

  .075670
```

# SECS(x)

### Forma generale

```
SECS(<exp>)
```

### Descrizione

Ritorna il numero di secondi trascorsi dall'avvio del timer <exp1> (compreso tra 1 e 3).Il valore continua ad incrementarsi anche dopo lo scatto del timer se è prevista l'opzione FREERUN e fino ad una RESET TIMER.

### Esempi

```
PRINT SECS(1)
```

# SGN(x)

### Forma generale

```
SGN(<exp>)
```

### Descrizione

SGN() ritorna la funzione segno di <exp>, cioè 0 se <exp>=0, 1 se <exp>>0, -1 se <exp><0.

### Esempi

```
PRINT SGN(12)

  1

PRINT SGN(-12)

  -1

PRINT SGN(0)

  0
```

# SHUTDN(0)

### Forma generale

```
SHUTDN(0)
```

### Descrizione

SHUTDN(0) deve essere usata solo all'interno di un segmento critico e ritorna 0 quando non è scattato l'allarme per mancanza di alimentazione e viceversa. Vedi par. 3.18.

### Esempio

```
10 CRITSEG
 .......
70 IF SHUTDN(0) = 1 THEN 130
 .......
80 ENCRIT
```

# SIN(x)

### Forma generale

```
SIN(<exp>)
```

### Descrizione

SIN() ritorna il seno di <exp> . <Exp> deve essere espressa in radianti.

### Esempio

```
10 LET A=123/SIN(X)
```

# SQR(x)

### Forma generale

```
SQR(<exp>)
```

### Descrizione

SQR() ritorna la radice quadrata di <exp>. <Exp> deve essere maggiore od uguale a zero.

### Esempio

```
PRINT SQR(25)

  5
```

## STR$(x)

### Forma generale

```
STR$(<exp>)
```

### Descrizione

STR$() ritorna come stringa il valore numerico di <exp>.

STR$ opera con le stesse routines di PRINT, per cui è possibile variare il formato di conversione mediante l'uso dei comandi di formato "%".

### Esempio

```
10 LET A$=STR$(100)
```

## SUBSTR$(x$,x,y)

### Forma generale

```
SUBSTR$(<string>,<exp1>,<exp2>)
```

### Descrizione

ritorna la sottostringa di <string> che inizia al carattere <exp1> di lunghezza <exp2>. <Exp1> = 1 corrisponde al primo carattere.

E' ammesso il costrutto

```
LET A$=SUBSTR$(A$,m,n)
```

# TAN(x)

### Forma generale

```
TAN(<exp>)
```

### Descrizione

TAN() ritorna la tangente di <exp>. <Exp> deve essere espressa in radianti.

### Esempio

```
10 LET S=TAN(X)
```

# TIME$()

### Forma generale

```
TIME$()
```

### Descrizione

ritorna una stringa del formato oo:mm:ss contenente il tempo attuale. Leggere quanto indicato per la funzione DATE$().

La funzione viene ottenuta mediante una chiamata al driver con codice #1 (BLOCK READ) e numero di record uguale a 1.

### Esempio

```
10 LET A$=TIME$()
20 PRINT A$
```

# TIMSEC(x)

### Forma generale

```
TIMSEC(<exp>)
```

### Descrizione

Ritorna il valore di conteggio del timer <exp>. Il valore viene inizializzato al valore massimo con la LOAD TIMER e si decrementa.Ad ogni scatto del timer, se è prevista l'opzione FREERUN, si ricarica il valore massimo.

### Esempio

```
10 PRINT TIMSEC(1)
```

# VAL(x$)

### Forma generale

```
VAL(<string>)
```

### Descrizione

converte in numero il valore della variabile stringa <string> in modo da poterlo usare in calcoli. <String>deve contenere solo caratteri numerici.

### Esempio

```
10 LET C$="6"
20 LET A=VAL(C$)/2
30 PRINT A
```

Risultato: 3

# XOR(m,n)

### Forma generale

```
XOR(<exp1>,<exp2>)
```

### Descrizione

ritorna lo XOR logico a 16 bit dei valori risultanti da <exp1> ed <exp2>.

# Comandi

## ERASE

### Forma generale

```
ERASE
```

### Descrizione

cancella il programma in memoria e tutte le variabili.

### Esempio

LIST

10 FOR I=1 TO 100

20 PRINT I

30 NEXT

40 END

ERASE

LIST

(nessun effetto)

## LIST

### Forma generale

```
1) LIST

2) LIST <linnum>

3) LIST <linnum1>,<linnum2>
```

### Descrizione

LIST serve per ottenere la lista del programma attualmente in memoria verso il dispositivo logico CON:/out. La lista può iniziare da una linea differente dalla prima specificando <linnum>. La lista si arresta dopo 24 linee o, se specificato <linnum2>, dopo <linnum> linee. Se <linnum> = 0 viene listato tutto il programma che si trova in memoria.

### Esempio

LIST 1000

1000 PRINT "Valore finale = ";T2

1010 END

## LLIST

### Forma generale

```
LLIST
```

(stesse opzioni di LIST)

Descrizione come LIST ma con uscita verso LST:.

## LOAD

### Forma generale

```
LOAD ABS <exp>
```

NOTA - esiste anche una frase LOAD (vedi).

### Descrizione

provoca il caricamento nell'area di lavoro RAM di un programma registrato in memoria all'indirizzo fisico <exp>. Tipicamente il programma viene generato dal procompilatore e registrato in eprom.

Il programma generato dal precompilatore (file con suffisso .GBA) non contiene indirizzi assoluti e può quindi essere ubicato in qualsiasi area libera.

Se presente della memoria scrivibile e non volatile, il programma può venirvi registrato con la frase SAVE ABS (vedi) e quindi riletto con LOAD ABS.

Il programma, una volta caricato con LOAD, può essere editato come se lo si fosse digitato da tastiera.

Si ricordi comunque che non è indispensabile caricare un programma in RAM per eseguirlo. Il comando RUN <exp> (vedi) consente di eseguire il programma direttamente in eprom.

Il comando LOAD ABS non carica direttamente dall'indirizzo indicato in argomento ma usa la funzione 90H del driver di console. In questa maniera è possibile customizzare il caricamento da una eventuale pagina estesa di rom.

Ad esempio, alcune CPU usano una 27512 divisa in due metà: la metà alta contiene il GBASIC e quella bassa il programma utente. L'utente carica il programma con LOAD ABS 0

# RUN

### Forma generale

```
1) RUN

2) RUN <exp>
```

### Descrizione

RUN serve per avviare l'esecuzione di un programma a partire dalla linea di numero più basso. Tutte le variabili sono azzerate.

L'esecuzione del programma può essere arrestata battendo

ESC a patto che l'arresto non sia stato inibito con la frase BREAK 0 (vedi).

RUN <exp> avvia l'esecuzione del programma presente in memoria alla locazione fisica specificata da <exp>. Dopo RUN <exp> viene disattivata l'area di lavoro RAM. Eventuali tentativi di modificare il programma danno luogo ad un messaggio di errore. L'area RAM torna attiva introducendo il comando RUN senza operando.

### Esempio

RUN 20000

(l'esecuzione inizia..)

## SAVE

### Forma generale

```
SAVE ABS <exp>
```

### Descrizione

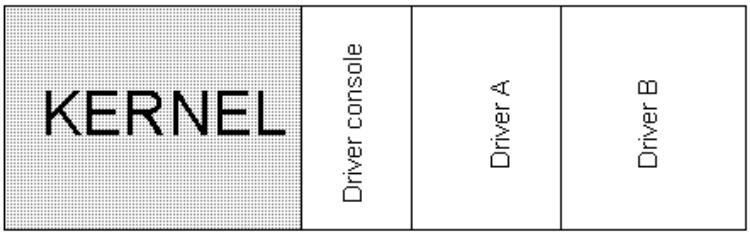
Provoca la registrazione del programma attualmente in memoria (RAM) all'indirizzo fisico specificato da <exp>. L'area interessata deve ovviamente essere disponibile e di adeguata capacità

Vedi comando LOAD ABS.

# Come scrivere un driver

## Generalità

La possibilità di poter scrivere un proprio driver rappresenta uno dei maggiori vantaggi dell'utente del GBASIC v.7.3. Scrivere un driver è assai meno semplice dello scrivere un programma GBASIC, in quanto è indispensabile una buona conoscenza della programmazione assembler e dello specifico dispositivo da interfacciare. Un driver però, una volta scritto e provato, diventa un patrimonio acquisito e può essere sfruttato molto velocemente anche da parte di coloro che ignorano la programmazione assembler.




Con queste premesse, scrivere un driver per GBASIC è relativamente semplice, come vedremo nel seguito, a prescindere ovviamente dai problemi tipici che una certa periferica può presentare.

Nel kit GBASIC sono forniti alcuni drivers esemplificativi, il più importante dei quali è certo quello della console. Questa sua importanza discende da due fatti: il primo è che il driver di console è indispensabile per il funzionamento di base del sistema, ed il secondo è che la sua relativa semplicità gli consente di costituire un valido esempio per chi si appresta a scrivere un nuovo driver.

Nella sezione 2 è stato spiegato come generalmente non sia necessario intervenire sul driver di console, potendosi agire semplicemente sulla stringa di configurazione per procedere all'implementazione del GBASIC su un nuovo hardware.

I motivi di studio e modifica del driver di console possono essere i seguenti (in ordine decrescente di probabilità):

a) necessità di modificare la funzione di autostart, scrivendo le poche linee di codice necessarie per l'acquisizione del segnale esterno e per fissare l'indirizzo di partenza dell'esecuzione;

c) la necessità di modificare certi parametri come parità, numero di bit ecc.

c) la necessità di utilizzare un hardware diverso da quello previsto come standard (ossia una UART diversa dalla SIO);

d) la necessità di utilizzare un modo diverso di comunicazione (ad esempio sincrono);

e) la necessità di eseguire particolari inizializzazioni di uno specifico hardware al momento del reset;

f) l'apprendimento dei fondamenti basilari per accingersi alla scrittura di un nuovo driver.

Una personalizzazione opzionale del kernel è prevista per la funzione di gestione della mancanza di alimentazione, come descritto nel seguito di questa sezione.

**NOTA: i listati riportati nelle pagine seguenti sono**

Libera riproduzione e libero uso

**esemplificativi e possono risultare non aggiornati. Riferirsi sempre alle versioni aggiornate presenti sul dischetto di distribuzione.**

## Drivers block o char oriented

I drivers sono classificabili in due categorie principali in funzione del fatto che i trasferimenti dati da e verso di essi avvenga per singoli caratteri o per interi blocchi.

Il driver di console, ad esempio, opera su singoli caratteri: ogni carattere proveniente dal la tastiera del terminale deve essere reinviato al video del medesimo di volta in volta.

Il driver di una stampante opera invece di solito in modo block: le linee sono inviate una alla volta via via che sono disponibili.

Un driver può prevedere un modo di operazione multiplo, ossia consentire il trasferimento tanto per singoli caratteri quanto per blocchi.

## Posizione fisica dei drivers

I drivers risiedono fisicamente in memoria nelle locazioni subito conseguenti al kernel come illustrato dalla figura sottostante:

Come si vede il driver di console è immediatamente contiguo al kernel e, a loro volta, ulteriori drivers sono "attaccati" al driver di console senza spazi infrapposti.

# Struttura fisica di un driver

I drivers hanno tutti una struttura fisica identica. All'inizio di ogni driver è presente un blocco di dati, detto header, che contiene alcune informazioni elementari tra cui la lunghezza fisica del driver.

Questa è la costituzione dell'header:

```
Carattere fisso: 1Bh
Lunghezza del driver (due bytes)
Tipo (un byte)
Nome del driver (8bytes)
Entry point del driver
```

Il "tipo" vale 0 per character, 1 per block e 2 per misto.

Come si può osservare, l'header contiene le informazioni che possono essere usate dal kernel per "saltare" da un driver all'altro per ricercarne uno in particolare (infatti il kernel conosce sempre la propria lunghezza).

In questo modo non sono necessari legami di alcun tipo tra kernel e drivers e non è necessario quindi procedeere al link complessivo. E' sufficiente, per l'utente assicurarsi della contiguità dei vari drivers.

La prima posizione dopo l'ultimo driver NON deve contenere il carattere 1Bh. Da questo infatti il kernel si rende conto che non ci sono più drivers.

Può essere conveniente per l'utente modificare il driver di console per adattarlo al proprio assemblatore in modo da linkare, con il proprio linker, i tutti driver assieme.

# Uso della RAM da parte dei drivers

Spesso un driver potrà necessitare l'uso di RAM.L'area a disposizione inizia dalla locazione F800h. Ogni driver occuperà l'area immediatamente seguente a quella utilizzata dal driver precedente.

# Funzioni eseguite dal driver

Il kernel richiama il driver, dopo avere identificato l'indirizzo fisico del suo entry point al momento dell'operazione di OPEN, richiedendo l'esecuzione di una certa funzione a mezzo di un codice. Le funzioni eseguibili dal driver sono descritte nelle pagine seguenti.

## Funz. driver #0 Block Write

### Descrizione

Invia un record di dati al driver passandogli l'indirizzo del blocco ed il numero del record. Il blocco è terminato dal carattere NULL. Viene chiamata alla esecuzione delle frasi WRITE, DISPLAY, TIME$() e DATE$().

### Parametri in ingresso

```
B = codice funzione (0)
HL = indirizzo del blocco
DE = numero del record
```

### Parametri in uscita

```
ACC = codice esito:
        0 = OK
        1 = ERRORE
```

# Funz. driver #1 Block Read

### Descrizione

Legge un record di dati dal driver passandogli l'indirizzo dell'area di destinazione ed il numero del record. Il blocco deve essere terminato dal carattere CR. Viene chiamata alla esecuzione della frase READ e dalle funzioni TIME$() e DATE$().

### Parametri in ingresso

```
B = codice funzione (1)
HL = indirizzo di destinazione
DE = numero del record
```

### Parametri in uscita

```
ACC = codice esito:
        0 = OK
        1 = ERRORE
```

# Funz. driver #2 Init

### Descrizione

Esegue l'inizializzazione del driver. Viene chiamata al momento della esecuzione della frase OPEN CH. Tra gli altri parametri viene passato anche il numero di canale assegnato dall'utente nella frase OPEN.

### Parametri in ingresso

```
B = codice funzione (2)
C = numero del canale usato nella OPEN CH..
E = flag di presenza stringa di conf.
   0 = no
   1 = si
HL = indirizzo della stringa di
configurazione (se presente)
```

### Parametri in uscita

```
ACC = codice esito:
        0 = OK
        1 = ERRORE
```

## Funz. driver #3 Close

### Descrizione

Esegue la disattivazione del driver. Viene chiamata al momento della esecuzione della frase CLOSE CH.

### Parametri in ingresso

```
B = codice funzione (3)
```

### Parametri in uscita

```
ACC = codice esito:
        0 = OK
        1 = ERRORE
```

## Funz. driver #4 Write Char

### Descrizione

Invia un carattere al driver. Chiamata dall'eco di console, dalle frasi PRINT, LPRINT, INPUT ecc.

### Parametri in ingresso

```
B = codice funzione (4)
E = carattere per uscita
```

### Parametri in uscita

```
ACC = codice esito:
        0 = OK
        1 = ERRORE
```

# Funz. driver #5 Read Char

### Descrizione

Invia un carattere al driver. Chiamata dall'input di console, dalle frasi INPUT ecc.

### Parametri in ingresso

```
B = codice funzione (5)
```

### Parametri in uscita

```
ACC = codice esito:
        0 = OK
        1 = ERRORE  E = carattere ritornato
```

# Funz. driver #6 Status

### Descrizione

Ritorna un codice corrispondente allo stato del driver quando viene richiamata la funzione DRVST() (vedi).

Il driver di console usa questa funzione anche in altre circostanze (ad esempio per riconoscere la pressione del ESC). Vedi sezione 5 ed appendice B.

La funzione DRVST() può avere opzionalmente due parametri. In questo caso il secondo parametro può assumere qualsiasi valore compreso tra 0 e 65535.

### Parametri in ingresso

```
B = codice funzione (6)
DE = parametro opzionale (se utilizzato)
```

### Parametri in uscita

```
ACC = codice esito:
        0 = OK
        1 = ERRORE
```

# Funz. driver #7 Info

### Descrizione

Richiesta di informazioni, selezionate in funzione di un indice dato dal registro E. Nella versione corrente le informazioni possibili sono le seguenti:

E=0 porta di controllo della principale SIO usata dal driver

E=1 idem, porta dati

E=2 idem, porta vettore

Nella corrente versione, solo la frase SIOSET (vedi) usa questa funzione.

### Parametri in ingresso

```
B = codice funzione (7)
E = indice richiesta funzione
```

### Parametri in uscita

```
ACC = codice esito:
        0 = OK
        1 = ERRORE
E = valore ritornato
```

# Funz. driver #8 Getch

### Descrizione

Ritorna il prossimo carattere dal buffer di ingresso di un driver character oriented. Se non ci sono caratteri ritorna 0.

### Parametri in ingresso

```
B = codice funzione (8)
```

### Parametri in uscita

```
ACC = codice esito:
```

```
        0 = OK
        1 = ERRORE
E = carattere letto o 0.
```

# Funz. driver #9 Flush

### Descrizione

Azzera il buffer di ingresso del driver specificato.

### Parametri in ingresso

```
B = codice funzione (9)
```

### Parametri in uscita

```
ACC = codice esito:
        0 = OK
        1 = ERRORE
```

# Funz. driver #10 Chnum

### Descrizione

Ritorna il numero dei caratteri presenti nel buffer di ingresso di un driver.

### Parametri in ingresso

```
B = codice funzione (10)
```

### Parametri in uscita

```
ACC = codice esito:
        0 = OK
        1 = ERRORE
E = num. dei caratteri presenti nel buffer.
```

## Funz. driver #88h Autostart

### Descrizione

Funzione speciale usata solo dal driver di console. Viene chiamata al momento del reset generale per stabilire se entrare nel normale modo operativo di console o se avviare automaticamente l'esecuzione di un programma.

In caso di autostart, il driver deve ritornare anche l'indirizzo di una stringa che contiene il comando da eseguire, tipicamente un "RUN <addr>" che deve essere terminata da CR (0Dh). <Addr> è l'indirizzo fisico, espresso in decimale, della prima locazione ove è ubicato il programma in formato .GBA.

### Esempio

```
AUTOCOM: DB   'RUN 20000',0dh
```

### Parametri in ingresso

```
B = codice funzione (88h)
```

### Parametri in uscita

```
ACC = codice esito:
       0 = OK
       1 = ERRORE
E = flag richiesta autostart
       0 = si
       1 = no
HL = indirizzo stringa con comando
```

# Il "System"

Esistono varie operazioni che non potrebbero essere eseguite direttamente dal driver senza l'ausilio o la supervisione del kernel ed altre ancora che, per la loro ripetitività, non conviene duplicare in ogni driver.

Per questo motivo il kernel offre la possibilità, con una procedura detta "system" di eseguire particolari funzioni richieste dall'utente.

Il system è una innovazione introdotta con la versione 7.xx del GBASIC ed il numero e la varietà delle funzioni offerte tende a crescere. Riferirsi eventualmente a note tecniche di aggiornamento.

La tecnica di chiamata di una funzione system è simile a quella delle funzioni driver: un parametro viene passato per richiedere una delle varie funzioni, numerate progressivamente.

Il punto di ingresso del system è alla locazione di indirizzo assoluto 0005.

# Funz. system #0 PSH_USTK

### Significato mnemonico

PuSH a record into the User STacK (fifo)

### Descrizione

Introduce un blocco di dati nel fifo dati. I dati introdotti possono poi essere riletti dal programma GBASIC mediante la frase POP. Il blocco non deve avere particolari caratteri di terminazione.

### Parametri in ingresso

```
ACC = codice funzione (0)
HL = indirizzo del blocco
BC = numero dei caratteri che compongono il
blocco
```

### Parametri in uscita

nessuno

# Funz. system #1 ASK_MY_LEV

### Significato mnemonico

ASK MY interrupt LEVel

### Descrizione

Ritorna il livello di interrupt associato al presente driver indicando il proprio numero di canale (che viene comunicato al driver dalla funzione driver INIT, vedi). L'utente associa il livello al driver con la frase

CH <m> TO INTERRUPT <n>

### Parametri in ingresso

```
ACC = codice funzione (1)
E = numero di canale
```

### Parametri in uscita

```
ACC = numero del livello di interrupt
```

# Funz. system #2 HIRQ

### Significato mnemonico

High Level Interrupt Request

### Descrizione

Inserisce nel FIFO degli interrupt una richiesta di interruzione al GBASIC. Deve essere chiamata indicando il livello richiesto, che deve corrispondere con quello assegnato al driver (vedi funzione precedente).

Nelle precedenti versioni questa funzione era denominata ISRX2.

### Parametri in ingresso

```
ACC = codice funzione (2)
E = livello di interrupt
```

### Parametri in uscita

nessuno

## Funz. system #3 INSTVEC

### Significato mnemonico

INSTall a VECtor into the low level interrupt table

### Descrizione

Installa uno o più vettori nella tabella degli interrupts a basso livello.Il vettore (HL) è l'indirizzo fisico della routine cui si deve saltare al momento dell'interruzione. La tabella dispone di spazio per 3 SIO (B da 0 a 5), per 3 CTC (B da 0 a 11) e per 3 PIO (B da 0 a 5). La posizione 0 della tabella SIO è normalmente occupata dal driver di console. Il tipo dei vettori e la posizione sono ricavati dalla stringa di configurazione. Vedi esempi e par. 3.7.

### Parametri in ingresso

```
ACC = codice funzione (3)
HL = tavola dei vettori da installare
B = numero  di posizione dove installare i
vett.
E = tipo dei vettori da installare
   0 = SIO (quattro vettori da installare)
   1 = CTC (un vettore)
   2 = PIO (un vettore)
```

### Parametri in uscita

```
ACC = esito 0 = OK / 1 = errore
C = 8 bit meno significativi del vettore di
```

```
interrupt (da programmare nella porta
relativa)
```

# Funz. system #4 PARSEP

### Significato mnemonico

PARSE a pair (keyletter+value) in the config string

### Descrizione

Aiuta a decodificare una stringa di configurazione. Viene chiamata di solito nella funzione driver INIT (vedi). La chiamata alla INIT lascia già posizionato il puntatore alla stringa (TXA_CS) rendendo possibile l'immediata chiamata a questa funzione. I valori ritornati sono i contenuti della prossima coppia della stringa.

### Esempio

Se la stringa e'

"DATA 45"

ritorna E='D' e B=45d

Vedere anche PARSEP2 e TRIPLEX.

### Parametri in ingresso

```
ACC = codice funzione (4)
puntatore (TXA_CS) alla stringa di
configurazione (è posizionato automaticamente
dal sistema alla chiamata della funzione
driver INIT)
```

### Parametri in uscita

```
zero flag = esito  set = OK / reset = non
trovato
E = lettera chiave
B = valore
```

# Funz. system #5 PARSEP2

### Significato mnemonico

PARSE a pair (keyletter+value) in the config string (type 2)

### Descrizione

Simile alla precedente PARSEP, consente di determinare in precedenza la lettera chiave che ci aspetta. Se questa non corrisponde a quella effettivamente presente nella stringa, si ha un errore GBASIC "ERR CS".

### Esempio

Se la stringa e'

"DATA 45"

ritorna B=45d

Vedere anche PARSEP e TRIPLEX.

### Parametri in ingresso

```
ACC = codice funzione (5)
E = lettera chiave prevista

puntatore (TXA_CS) alla stringa di
configurazione (è posizionato automaticamente
dal sistema alla chiamata della funzione
driver INIT)
```

### Parametri in uscita

```
zero flag = esito  set = OK / reset = non
trovato
B = valore
```

# Funz. system #6 CSERR

### Significato mnemonico

Configuration String ERRor

### Descrizione

Genera un ERR CS (errore di stringa di configurazione)

### Parametri in ingresso

```
ACC = codice funzione (6)
B = 1 (non usato al momento ma da prevedere
per futura compatibilità)
```

### Parametri in uscita

```
nessuno - la funzione non ritorna ma il
GBASIC passa a livello comandi.
```

# Funz. system #7 TRIPLEX

### Significato mnemonico

parse a common config string (SIO, PIO, CTC)

### Descrizione

Analizza una stringa di configurazione di tipo comune (SIO, PIO o CTC). La PIO non è supportata dalla corrente versione. E' la routine più usata per decodificare stringhe di configurazione per la sua maggiore semplictà di uso. Il numero dei parametri richiesti nella stringa varia in funzione del tipo: per la SIO sono richiesti 3 indirizzi (dati, controllo e vettore) mentre per il CTC solo uno (dati), potendo il kernel ricavare gli altri in modo automatico.

### Esempio

Se la stringa e'

"SIO 2 DATA 145 CONT 146 VECT 146"

ritorna B=2, D=145, C=146, E=146

### Parametri in ingresso

```
ACC = codice funzione (7)
D = lettera chiave prevista
   S = SIO
   C = CTC
   P = PIO
E = 1 se stringa presente; E = 0 viceversa

puntatore (TXA_CS) alla stringa di
configurazione (è posizionato automaticamente
dal sistema alla chiamata della funzione
driver INIT)
```

### Parametri in uscita

```
zero flag = esito  set = OK / reset = non
trovato
 D = porta dati
 C = porta controllo
 E = porta vettore
 B = numero della posizione in tabella
```

## Funz. system #8 REPLDRV

### Significato mnemonico

REPlace a DRiVer with a new one

### Descrizione

Il GBASIC conserva gli indirizzi dei drivers in una tabella interna chiamata DIRTAB. La funzione REPLDRV rimpiazza l'indirizzo presente in tabella con uno nuovo fornito dal driver. L'unico controllo fatto è che il canale sia stato effettivamente aperto. Si consiglia di usare solo a ragion veduta.

### Parametri in ingresso

```
ACC = codice funzione (8)
E = numero del canale
HL = indirizzo fisico del nuovo driver
```

### Parametri in uscita

```
nessuno
```

## Funz. system #9 RPLBKSP

### Significato mnemonico

RePLace BacKSPace key

### Descrizione

Normalmente la funzione "backspace" (cancellazione ultimo carattere) è affidata al codice 08. Con questa funzione essa può essere riassegnata ad un codice diverso.

### Parametri in ingresso

```
ACC = codice funzione (9)
E = codice del nuovo tasto backspace
```

### Parametri in uscita

```
nessuno
```

## Funz. system #10 NEWTASK

### Significato mnemonico

Install NEW TASK

### Descrizione

Installa un nuovo task nel MMT (Mini Multi Task). La

versione attuale supporta solo in parte MMT.Vedi nei paragrafo 5.8.

### Parametri in ingresso

```
ACC = codice funzione (10)
HL = indirizzo routine da installare
B = fattore di prescala (*)
C = livello di priorità (*)
D = massimo tempo di CPU (in "tics")(*)
```

(*) non supportati nella corrente versione

### Parametri in uscita

```
nessuno
```

## Funz. system #11 STOPALL

Questa funzione è stata eliminata.

## Funz. system #12 DOCKSUM

### Significato mnemonico

DO ram ChecKSUM

### Descrizione

Esegui il calcolo statistico del checksum della RAM ed archivialo nell'apposita variabile. Usata durante power down.

### Parametri in ingresso

```
ACC = codice funzione (12)
```

### Parametri in uscita

```
nessuno
```

# Funz. system #13 CRITSEG

### Significato mnemonico

CRITical SEGment

### Descrizione

Definisce l'inizio di un segmento critico (vedi par.3.18).

### Parametri in ingresso

```
ACC = codice funzione (13)
```

### Parametri in uscita

```
nessuno
```

# Funz. system #14 ENDCRIT

### Significato mnemonico

END of a CRITical segment

### Descrizione

Definisce la fine di un segmento critico (vedi par.3.18).

### Parametri in ingresso

```
ACC = codice funzione (14)
```

### Parametri in uscita

```
nessuno
```

# Funz. system #15 &BOF

### Significato mnemonico

address of Beginning Of File pointer

Libera riproduzione e libero uso

### Descrizione

Ritorna l'indirizzo della locazione ove è contenuto l'indirizzo della prima locazione del programma.

### Parametri in ingresso

```
ACC = codice funzione (15)
```

### Parametri in uscita

```
HL = indirizzo della locazione BOF
```

## Funz. system #16 &EOF

### Significato mnemonico

address of End Of File pointer

### Descrizione

Ritorna l'indirizzo della locazione ove è contenuto l'indirizzo della ultima locazione del programma.

### Parametri in ingresso

```
ACC = codice funzione (16)
```

### Parametri in uscita

```
HL = indirizzo della locazione EOF
```

## Funz. system #18 CNS

### Significato mnemonico

CoNvert integer to String

### Descrizione

Converte un intero in una stringa.

### Parametri in ingresso

```
ACC = codice funzione (18)
HL - valore da convertire
DE - punta al buffer dove porre la stringa
```

### Parametri in uscita

```
DE - punta alla stringa convertita
```

## Funz. system #19 INTMON

### Significato mnemonico

INTerrupt MONitor

### Descrizione

Dalla v.7.4x è stata aggiunta la funzione INTMON per aumentare l'arco di riconoscibilità delle interruzione GBASIC. In precedenza le interruzioni erano riconosciute solo all'inizio dell'esecuzione di ogni frase. In certi casi questa situazione risultava molto penalizzante. In particolare la frase INPUT da operatore bloccava il servizio degli interrupt (con un inevitabile overflow dello stack delle richieste).

La nuova chiamata consente di monitorizzare lo stato degli interrupts, e, nel caso, di accettarli e servirli. Questa chiamata viene inserita nei loop di attesa (come ad esempio quello che aspetta un carattere da tastiera).

Si osservi che INTMON, quando serve un interrupt, preserva tutti i registri (tranne acc.) e TXA (il puntatore interno al programma in esecuzione) e che i programmi di interrupt non possono richiamare ancora INTMON.

Ad esempio una routine di interrupt servita durante una INPUT da tastiera, non può contenere un'altra INPUT da tastiera. Viceversa si ottiene un errore di segmento critico.

In sintesi, grazie ad INTMON, la INPUT da operatore non

blocca più il servizio degli interrupts. Il driver di console (CONS.DRV) è già stato modificato per sfruttare questa nuova possibilità.

### Parametri in ingresso

```
ACC = codice funzione (19)
```

### Parametri in uscita

```
nessuno
```

## Funz. system #20 QRYMMT

### Significato mnemonico

QuerY MMT

### Descrizione

Ritorna il valore utilizzato dall'utente per programmare il timer dell'MMT o zero se MMT non è attivo (non usare 0 come costante del CTC).

### Parametri in ingresso

```
ACC = codice funzione (20)
```

### Parametri in uscita

```
ACC = valore della costante di MMT
```

## Funz. system #21 QRYFREQ

### Significato mnemonico

QueRY Frequency

### Descrizione

Ritorna la frequenza del clock di sistema divisa per 10000 o 0 se il dato non è disponibile. La frequenza viene calcolata solo

se è presente il driver CCAL: ed il relativo hardware.

**Parametri in ingresso**

```
ACC = codice funzione (21)
```

**Parametri in uscita**

```
ACC - valore richiesto
```

## Funz. system #22 DRADDR

**Significato mnemonico**

DRiver ADDress

**Descrizione**

Ritorna l'indirizzo fisico del driver richiesto che deve essere aperto.

**Parametri in ingresso**

```
ACC = codice funzione (22)
```

**Parametri in uscita**

```
DE ed IY = indirizzo del driver richiesto
```

## Funz. system #23 POPTASK

**Significato mnemonico**

POP TASK

**Descrizione**

Elimina l'ultimo driver installato nell'MMT.

**Parametri in ingresso**

```
ACC = codice funzione (23)
```

### Parametri in uscita

```
nessuno
```

# Mini Multi Task (MMT)

MMT e' un semplice esecutore multitask che ha principalmente la funzione essere utilizzato dai drivers. E' infatti piuttosto frequente che un driver piuttosto complesso abbia necessita' di usare un processo parallelo: ad esempio, per un driver di stampa con spooler, può essere utile realizzare un processo che ad intervalli verifica la presenza di caratteri da stampare. Per evitare che ogni driver operi a modo proprio, e' stato realizzato questo MMT. Al momento opera a livello minimo ma e' gia' previsto per una più estesa funzionalità.

All'accensione MMT e' inattivo. Per renderlo si deve:

a) attivare un oscillatore usando un CTC che non sia usato per altro (evitare OSC 0 che di solito e' impiegato come baud rate generator per la SIO di console).

### Esempio

```
open osc 2,"C 1 D 145"
```

b) specificare con l'apposita frase che l'oscillatore in questione e' usato come clock per il task switching. La frase e

```
osc 2, 40 for switch
```

dove 40 e' la costante che verrà programmata nel CTC, tenuto presente da parte di quest'ultimo il massimo fattore di prescaler (256). Con 40, a 3.6864 MHz si avrebbe quindi:

270 ns x 256 x 40 = 2.77 msec

Ad ogni "tic" (nell'esempio precedente ogni 2.77 msec) l'esecuzione del GBASIC viene interrotta ed inizia quella del prossimo lavoro in coda di attesa che termina con una semplice RET.

La routine chiamata può alterare ogni registro tranne IX, IY ed SP.

La coda di attesa opera al momento secondo una semplice logica "round robin". Il concetto non e' tanto quello di un alternarsi di task diversi, quanto quello di interrompere periodicamente l'esecuzione del GBASIC per eseguire un task secondario. Il GBASIC non e' quindi inserito nella coda di attesa e si presuppone pertanto che il tempo di esecuzione dei singoli task sia assai inferiore al periodo di "tic" (questi concetti sono destinati pero' ad evolversi in futuro).

Un driver può installare un task nella coda di attesa (una volta per tutte) usando la funzione system #10 (vedi).

## Power down

Questa sezione è variata con l'introduzione dei segmenti critici. Si noti in particolare che la routine di esecuzione del power down inizia dalla locazione 069H e non più 066H.

Il power down restava fino alla presente versione una caratteristica del kernel del tutto al di fuori della portata dell'utente. Adesso il kernel e' stato modificato per consentire una semplice customizzazione. In primo luogo sono state aggiunte due nuove funzioni di system: la #11 e la #12 precedentemente descritte per arrestare i CTC e per valutare il checksum di RAM.

Il kernel e' stato poi rivisto in modo da concentrare le routines interessate in aree ben definite. Si hanno due indirizzi riservati:

63H - salto alla routine di inizializzazione del sistema di

power down che deve terminare con un RET;

69H - routine di servizio per la mancanza di alimentazione (si suppone sempre che l'hardware preveda un NMI per il power down).

Lo spazio da 66h a BFh e' libero per inserire le routines dell'utente. Si riportano delle routine esemplificative (i riferimenti all'hardware specifico sono ovviamente da rivedere caso per caso).

```
;-------------------------------------------------------------------------
;     63H ABSOLUTE - JP TO POWER FAULT INIT ROUTINE
;-------------------------------------------------------------------------
      ORG   63H
INIPOW:     JP    INIPOW1
;-------------------------------------------------------------------------
;     69H ABSOLUTE - POWER FAULT EXECUTE ROUTINE
;-------------------------------------------------------------------------
      ORG   69H
;-------------------------------------------------------------------------
;     CUSTOM FMD ROUTINE TO EXECUTE POWER DOWN
;-------------------------------------------------------------------------
KILPORT     EQU   60H         ;/KILL PORT
KILBIT      EQU   4           ;/KILL BIT

NMI:  DI
      LD    SP,STACK1   ;USA LO STACK RISERVATO ALLA LOC 8010H
                        ;NON SFRUTTARE PIU' DI 4 LIVELLI
;(queste due istr. sono sempre presenti e possono essere lasciate  intatte)
      XOR   A
      OUT   (70H),A
      OUT   (0EH),A

      LD    A,12        ;CALCOLA E SALVA CHECKSUM RAM PER
      CALL  SYS         ;CONTROLLARLO ALLA RIPARTENZA

      XOR   A
      OUT   (KILPORT),A ;AUTOKILL

      LD    B,10        ;ATTENDI MANCANZA ALIMENTAZIONE
      LD    DE,0
SIP2: DEC   DE
      LD    A,D
      OR    E
      JR    NZ,SIP2
      DJNZ  SIP2

      JP    0           ;EVENTUALMENTE RIPARTI DA 0000H
;-------------------------------------------------------------------------
;     CUSTOM ROUTINE TO INIT POWER DOWN SYSTEM
;-------------------------------------------------------------------------
```

```
;     LO STACK E' GIA' INIZIALIZZATO - NON USARE PIU' DI 3 LIVELLI

INIPOW1:XOR A
;     RES   KILBIT,A    ;SUPERFLUA  OUT   (KILPORT),A ;AUTOKILL

      OUT   (70H),A
      CPL
      OUT   (0EH),A           ;SPEGNE LEDS

      LD    HL,0
LUPSTA: DEC HL          ;RITARDO
      LD    A,H
      OR    L
      JR    NZ,LUPSTA

      XOR   A
      SET   KILBIT,A
      OUT   (KILPORT),A ;NO AUTOKILL
      RET

;     E' GARANTITA LA DISPONIBILITA' DELL'AREA CHE SEGUE FINO A 0BFH
;     (ANCHE NELLE VERSIONI FUTURE) PER LE ROUTINES UTENTE

      ORG    0C0H
```

# Il driver di console

Nella appendice B è riportato il listato del driver di console fornito come standard. Si tratta, come accennato, di un driver di tipo character che governa una SIO. Il listato è abbondantemente commentato e ci soffermeremo pertanto solo sui punti più critici.

Solo nella inizializzazione il driver CONS: esegue delle particolari operazioni che possono non risultare ben chiare.

## *Inizializzazione porte*

Le operazioni svolte sono le seguenti:

1) resetta con 18h il ctlp (channel reset = disabilita tutto interrupts compresi per un singolo canale)

2) resetta con 18h il vecp

3) programma il registro 1 di vecp con 4 (solo "status affects vector" - condizione normale per usare la tabella dei vettori invece del vettore singolo). NOTA BENE che questo bit e' attivo solo sul canale B ossia sulla porta vettore.

4) programma da tabella il ctlp (compreso un 10h = reset ext status interrups ed un 1Ch in reg 1 che significa status affects vector e int on all rx chars parity does NOT affect vector).

La necessita' di compiere queste macchinose operazioni deriva dal bisogno di avere certamente programmato lo "status affects vector" nel registro 1 della porta vettore.

Questo pero' ancora non spiega come mai si debba resettare con 18h la porta controllo; questo avviene solo perche' la medesima operazione non è compresa nella tabella di programmazione standard.

In linea generale (ossia per altri drivers) le operazioni da fare non possono essere quelle sopra descritte poiché in questo modo si rischia di cancellare una programmazione gia' esistente per un'altra porta.

Il modo corretto è risultare il seguente:

a) resettare la porta controllo con 18h - questo sicuramente non da' inconvenienti;

b) programmare il valore desiderato nel registro 1 della porta vettore. Questo valore potrebbe essere sempre 14h. Se questo fosse gia' scritto, una riscrittura non dovrebbe fare danno.

c) programmare la tabella nella porta controllo;

d) programmare il vettore nella porta vettore. Questo, in ogni caso, ossia per porte A o B, dovrebbe avere un identico valore e non dovrebbe dar luogo a problemi.

## *Autostart da RAM*

E' possibile che l'autostart avvii il programma in RAM alle seguenti condizioni:

a) la RAM deve prevedere caratteristiche di non volatilità;

b) la stringa di avviamento (funzione #88) deve essere semplicemente "RUN";

c) la prima frase del programma deve essere una REM.

## Un driver block oriented

In appendice C è riportato, sempre a titolo di esempio, il listato di un driver block oriented. Si tratta di un driver per il clock / calendario realizzato con un chip Mariner M3003.

## Drivers per dispositivo logico AUX:

Il dispositivo logico AUX: richiede un driver di tipo character oriented (usare quindi DISPLAY o PRINT e non WRITE). Per poter eseguire correttamente la frase NGET$, il driver riconoscere i caratteri di controllo descritti nella tabella seguente:

| Mnemo | Dec | Funzione |
|---|---|---|
| SEL1 | 1 | Seleziona display 1 |
| SEL2 | 2 | Seleziona display 2 |
| CBACK | 8 | Passo indietro cursore |
| CHOME | 11 | Cursore ad inizio riga |
| CLRD | 12 | Azzeramento display |
| CSTEP | 14 | Passo avanti cursore |
| COFF | 15 | Spegnimento cursore |

# Richiesta interrupt ad alto livello

La serie di esempi si conclude con un breve listato che illustra come un driver possa richiedere un interrupt al GBASIC. Tipicamente, una routine di servizio di un interrupt a basso livello, quando decide di richiedere un interrupt all'alto livello, deve solo eseguire le operazioni sottodescritte.

```
;     ESEMPIO DI GENERAZIONE INTERRUPT DA UN DRIVER
;     LE ISTRUZIONI CHE SEGUONO SONO ESEGUITE IN UNA ROUTINE DI
;     SERVIZIO INTERRUPT A BASSO LIVELLO

;     ESEGUI IL PUSH NEL FIFO DEL RECORD DA PASSARE AL GBASIC

INTGB:      LD    A,13        ;SEGMENTO CRITICO (SOLO SE
      CALL  SYS          ;HARDWARE PREVEDE POWER FAULT)
      LD    B,0          ;PREDISPONI IN BC IL NUMERO DEI CARATTERI
      LD    A,(COUNT)          ;DA PUSHARE NEL FIFO
      LD    C,A
      LD    HL,RBNMSG          ;INDIRIZZO DEI DATI DA PUSHARE
      XOR   A            ;CODICE FUNZIONE SYSTEM #0 PSH_USTK
      CALL  SYSTEM             ;ESEGUI IL PUSH

      LD    A,(CHNUM)          ;NUMERO DEL NOSTRO CANALE - ERA STATO     LD
      E,A         ;SALVATO AL MOMENTO DELLA INIT
      LD    A,2          ;CODICE FUNZIONE SYSTEM #2 - ASKMYLEV
      CALL  SYSTEM             ;RICHIEDI IL NOSTRO LIVELLO

      LD A,14                  ;FINE SEGMENTO CRITICO (SOLO SE POW FAULT)
      CALL SYS

      LD    E,A          ;E CONTIENE ORA IL NOSTRO LIVELLO
      LD    A,1          ;CODICE FUNZIONE SYSTEM #1 - HIRQ
      CALL  SYSTEM             ;RICHIEDI INTERRUPT A BASIC

      EXX
      EX    AF,AF'
      EI
      RETI
```

# Drivers associati al dispositivo AUX:

Il dispositivo logico AUX: è utilizzato dalle frasi DISPLAY e dalla funzione NGET$.

I drivers ad esso associati devono essere di tipo orientati al carattere. Per operare correttamente con la NGET$() deve eseguire particolari funzioni quando riceve i seguenti caratteri di controllo:

| Dec | Funzione svolta |
|---|---|
| 1 | Seleziona il display n.1 |
| 2 | Seleziona il display n.2 |
| 8 | Passo indietro cursore |
| 11 | Riporta il cursore a inizio linea |
| 12 | Ripulisce il display |
| 14 | Passo avanti cursore |
| 15 | Disattiva il cursore |

Libera riproduzione e libero uso

# Messaggi di errore

## Messaggi di errore

Sono elencati nel seguito i messaggi di errore emessi dal GBASIC. I messaggi di errore del precompilatore PREGB sono descritti nella sezione relativa.

I messaggi di errore sono emessi nella forma:

ERR <tipo errore> LN <linnum>

dove <tipo errore è una sigla che identifica il tipo di errore e LN è il numero della linea dove si è verificato l'errore.

### SINT

Errore di sintassi. L'interprete non comprende un comando o la costruzione di una frase.

### VAL

Parametro con valore errato.

## F/NXT/RET 1 (o F/NXT/RET 2)

Errore di FOR..NEXT o RETURN. Probabilmente FOR senza NEXT o viceversa.

## FOR/NXT

Errore di FOR..NEXT.

## RET

Return senza GOSUB. CTL STK OF

Overflow dello stack di controllo. Dovuto di solito ad un eccessivo numero di subroutine o a frasi FOR..NEXT annindiate una dentro l'altra.

## IDX

Un indice di una variabile ha superato il proprio valore massimo ammesso. Può derivare anche da un indice scorretto in una ON..GOTO od ON..GOSUB.

## DIM

Variabile ad indice non dimensionata o fine dello spazio riservato alle matrici.

## MEM

Superamento della capacità di memoria. Si verifica aggiungendo linee al programma o scrivendo su storages oltre la massima capacità dello pseudofile.

Libera riproduzione e libero uso

## INT NON PREDIS

E' pervenuta una richiesta di interruzione senza che sia stata eseguita la frase ON INTERRUPT.

## INT#

E' pervenuta una richiesta di interruzione ad un livello non ammesso, ossia minore di 0 o maggiore di 3.

## STG

Errore in una variabile o costante stringa. Stringa prevista e non trovata. Stringa non prevista.

## READ

Errore nella esecuzione della frase READ/DATA per tentativo di leggere oltre la fine dei DATA.

## INPUT

Errore nella esecuzione di una frase INPUT.

## STK

Superamento della capacità dello stack aritmetico interno. Espressione numerica troppo complessa.

## RAD NEG

Tentativo di eseguire una SQR() con argomento <0.

## NUM LIN

Numero di linea inesistente.

## MODO

Si è tentato di eseguirte in modo diretto un comando ammesso solo per il modo differito.

## LN LUNGA

Una frase input ha ricevuto un numero eccessivo di caratteri (vedi nota alla frase INPUT LDEV 5)

## CH

Errore nel canale specificato nella frase INPUT CH o PRINT CH nella vecchia forma. Da non confondere con i canali dei drivers.

## ARITM

Superamento capacità aritmetica.

## DEV INES

Dispositivo fisico inesistente.Es. DISPLAY 3, CLOSE 9 ecc.

## DIV PER 0

Tentativo di eseguire una divisione per 0.

## GIA' OPEN

Tentativo di aprire un driver già aperto

## NON OPEN

Tentativo di utilizzare un driver non aperto.

## NOME

Nome del driver maggiore di 8 caratteri. DRV

Errore correlato ad un driver.

## VETT INT

La funzione dos INSTVEC è stata chiamata con un parametro E 2 o <0 (ossia è stato richiesto di installare un vettore che non appartiene nè ad una SIO, nè ad una PIO. nè ad un CTC).

## BUG

Errore interno. Segnalare.

## C STG

Errore in una stringa di configurazione.

## MMT

Errore del Mini Multi Task. Si è tentato di installare più task del numero massimo previsto (4 sulla versione attuale).

# I STK OVFL

Un FIFO degli interrupts (dati o richieste) ha superato la massima capacità ammessa. Può accadere se gli interrupt arrivano a velocità assai maggiore di quella massima con cui possono essere processati o se l'interrupt è rimasto disabilitato (vedi frase INTERRUPT). Notare che la versione corrente, a differenza delle precedenti, si inizializza con l'interrupt disabilitato e che pertanto esso deve essere riabilitato da programma. con

INTERRUPT 1

# Listato di CONS.DRV

Si riporta a titolo di esempio il listato di un possibile driver per la console. Detto listato coincide solo a grandi linee con quello effettivamente fornito con il GBASIC.

```
;=============================================================================
;
;     DRIVER PER CONSOLLE SERIALE - GBASIC COMPATIBILE
;
;     VERS. 1.20 18/10/90 - COPYRIGHT ING. G.BECATTINI - FIRENZE
;
;=============================================================================


      CSEG

SYSTEM      EQU   5     ;GBASIC SYSTEM ENTRY POINT
BUFLEN      EQU   20    ;BUFFER DI RICEZIONE

CR    EQU   0DH
LF    EQU   0AH
XON   EQU   11H
XOFF  EQU   13H
ESC   EQU   27

      INCLUDE     B7.CNF

;-----------------------------------------------------------------------------
;                SEZIONE DI PUBBLICO ACCESSO
```

```
;-------------------------------------------------------------------------

;-------------------------------------------------------------------------
;     HEADER
;-------------------------------------------------------------------------
HEADER:     DB    1BH         ;INDICA DRIVER PRESENTE

      DW    DRVEND-HEADER

      DB    0           ;DEFINIZIONE DEL TIPO - VALORI AMMESSI
                        ;     0 - CHAR ORIENTED
                        ;     1 - BLOCK ORIENTED
                        ;     2 - SPECIAL DEVICE

      DB    'CON:....'  ;NOME DEL DRIVER - SEMPRE 8 CARATTERI

;-------------------------------------------------------------------------
;     INIZIO PARTE ESEGUIBILE
;-------------------------------------------------------------------------
XCONS:
DRIVER:
      LD    A,B         ;DECIDI QUALE OPERAZIONE ESEGUIRE
      CP    2
      JP    Z,INISIO
      CP    4
      JP    Z,SIOTX
      CP    3
      JP    Z,DRV_CLO
      CP    5
      JP    Z,RDCHAR
      CP    6
      JP    Z,STATUS
      CP    7
      JP    Z,INFO
      CP    8
      JP    Z,GETCH
      CP    9
      JP    Z,FLUSH
      CP    10
      JP    Z,CHCNT
      CP    88H
      JP    Z,AUTO

      LD    A,1         ;ERRORE - CODICE RICHIESTA NON AMMESSO
      RET

;-------------------------------------------------------------------------
;     FUNZIONE N.2 - INIT - INIZIALIZZA IL DRIVER
;-------------------------------------------------------------------------

INISIO:     DI

      LD    A,7         ;IL REG E E' GIA' SETTATO DA CHI HA FATTO
                        ;CALL DRIVER
      LD    D,'S'       ;ASPETTATI UNA SIO
      CALL  SYSTEM            ;"TRIPLEX" - ANALIZZA LA STRINGA DI CONFIG
```

```
                        ;CHE DEVE ESSERE PRESENTE E RITORNA
                        ;D=DATA C=CONTROLLO V=VETTORE B=TABELLA
        LD      HL,CTLP         ;CONTROL PORT     LD    (HL),C
        INC     HL
        LD      (HL),D          ;DATA PORT
        INC     HL
        LD      (HL),E          ;VECTOR PORT
                        ;B=N.TABELLA

        LD      A,3             ;CHIAMA INSTVEC
        LD      HL,TABVECS      ;VETTORI DA INSTALLARE
        LD      E,0             ;SIO
        CALL    SYSTEM
        PUSH    BC              ;VETTORE

;       ADESSO RESETTO ENTRAMBI I CANALI CTLP E VECP -
;       QUESTO E' CONSENTITO IN QUANTO
;       IL CANALE DI CONSOLE E' SEMPRE IL PRIMO AD ESSERE PROGRAMMATO
;       DALLO STESSO BASIC DURANTE L'INIZIALIZZAZIONE - SE IL REGISTRO
;       VETTORE FOSSE LO STESSO DI CTLP NON CI SAREBBERO PROBLEMI IN QUANTO
;       IL RESET E' FATTO ANCHE DALLA PROGRAMMAZIONE OTIR (VEDI TESTO)


        LD      A,(CTLP)
        LD      C,A

        LD      A,18H           ;RESETTA CANALE CTLP
        OUT     (C),A

        LD      A,(VECP)
        LD      C,A

        LD      A,18H           ;RESETTA CANALE VECP
        OUT     (C),A

        LD      A,1             ;PROGRAMMA REGISTRO 1
        OUT     (C),A

        LD      A,4             ;CON SOLO "STATUS AFFECTS VECTOR"
        OUT     (C),A

        CALL    FLUSH

        LD      HL,TABSIOA
        LD      A,(CTLP)
        LD      C,A
        LD      B,9
        OTIR

        LD      A,(VECP)
        LD      C,A

        LD      A,2             ;SELEZIONA REGISTRO VETTORE
        OUT     (C),A
        POP     DE              ;VETTORE - PUSHATO COME BC
        LD      A,E             ;PROGRAMMA VETTORE
```

```
      OUT   (C),A

      LD    E,XON
      CALL  SIOTX

      XOR   A
      LD    (XOFSNT),A
      EI
      RET

;-------------------------------------------------------------------------
;     FUNZIONE #3 - CLOSE
;-------------------------------------------------------------------------
DRV_CLO:    XOR   A
      RET

;-------------------------------------------------------------------------
;     FUNZIONE N.4 - WRITE CHAR - TRASMISSIONE DI UN CARATTERE SULLA LINEA
;-------------------------------------------------------------------------
SIOTX:      LD    A,E
      PUSH        AF
      LD    A,(CTLP)
      LD    C,A
SIOT1:      IN    A,(C)
      BIT   2,A         ;TX VUOTO?
      JR    Z,SIOT1
      LD    A,(DTAP)
      LD    C,A
            POP         AF
      OUT   (C),A
      XOR   A           ;TUTTO OK
      RET

;-------------------------------------------------------------------------
;     FUNZIONE N.5 READ CHAR - LETTURA CAR. DA BUFFER DI RICEZ. O ATTESA
;-------------------------------------------------------------------------

RDCHAR:     PUSH  HL
      PUSH  BC
      LD    HL,CHR_CNT        ;INDIRIZZO DEL CONTACARATTERI
      LD    A,6
      CP    (HL)
      JR    C,SIOANF          ;PIU' DI 6 CARATTERI NEL BUFFER, SALTA

      LD    A,(XOFSNT)        ;MENO DI 6 CARATTERI NEL BUFFER
      OR    A           ;E' STATO SPEDITO PER ULTIMO UNO XOFF?
      JR    Z,SIOANF          ;NO, SALTA

      LD    E,XON       ;INVITA A TRASMETTERE   CALL  SIOTX

      XOR   A           ;ATTIVA PROSSIMA TRASMISSIONE DI XOFF
      LD    (XOFSNT),A        ;QUANDO NECESSARIA

SIOANF:     XOR   A
SAWAIT:     CP    (HL)        ;BUFFER VUOTO (CONTATORE=0)?
      JR    Z,SAWAIT          ;ATTENDI UN CARATTERE
```

```
RDCH1:      DEC   (HL)        ;CARATTERE ARRIVATO - DECREMENTA CONTATORE
      LD    HL,(RD_PO)        ;PRENDI PUNTATORE DI LETTURA
      LD    B,(HL)            ;CARATTERE IN B
      CALL  INCPOIN           ;INCREMENTA POINTER
      LD    (RD_PO),HL
RDCH2:      LD    A,B         ;CARATTERE IN ACC
      AND   7FH         ;VIA BIT 7
      POP   BC
      POP   HL
      LD    E,A         ;METTI CARATTERE IN E
      XOR   A           ;TUTTO OK
      RET

;-------------------------------------------------------------------------
;     FUNZIONE N.6 - RITORNA LO STATO DELLA PERIFERICA
;-------------------------------------------------------------------------
;     RITORNA E=0 SE NON CI SONO CARATTERI IN ATTESA
;     RITORNA E=CARATTERE SE CI SONO CARATTERI IN ATTESA
;     NON MODIFICA IL BUFFER

STATUS:     LD    A,(CHR_CNT)
      LD    E,0
      OR    A           ;CARATTERI NEL BUFFER
      RET   Z           ;NO,  RET CON E=00
      LD    HL,(RD_PO)
      LD    E,(HL)            ;SI,RET CON E=CARATTERE
      XOR   A
      RET

;-------------------------------------------------------------------------
;     FUNZIONE N.7 - INFO
;-------------------------------------------------------------------------

;     QUESTA E' UNA FUNZIONE OPZIONALE - IN QUESTO CASO SERVE PER RITORNARE
;     IL NUMERO DELLE PORTE FISICHE UTILIZZATE DALLA MAIN SIO DI QUESTO DRV
;           0=CTL 1=DTA 2=VEC - CHIAMATA DA SIOSET

INFO: LD    D,0         ;USA E COME INDICE ALLE TRE LOC DATI
      LD    HL,CTLP           ;TANTO SAPPIAMO CHE SONO CONTIGUE
      ADD   HL,DE
      LD    A,(HL)
      LD    E,A         ;RITORNA IL VALORE IN E
      XOR   A
      RET
;-------------------------------------------------------------------------
;     FUNZIONE N.8 - GETCH - RITORNA 0 OD IL CAR. PRELEVANDOLO DA BUFF.
;-------------------------------------------------------------------------
GETCH:      PUSH  HL
      PUSH  BC
      LD    HL,CHR_CNT        ;INDIRIZZO DEL CONTACARATTERI
      LD    A,6
      CP    (HL)
      JR    C,SIOANF1         ;PIU' DI 6 CARATTERI NEL BUFFER, SALTA

      LD    A,(XOFSNT)        ;MENO DI 6 CARATTERI NEL BUFFER
      OR    A           ;E' STATO SPEDITO PER ULTIMO UNO XOFF?
```

```
      JR    Z,SIOANF1         ;NO, SALTA

      LD    E,XON       ;INVITA A TRASMETTERE
      CALL  SIOTX

      XOR   A           ;ATTIVA PROSSIMA TRASMISSIONE DI XOFF
      LD    (XOFSNT),A        ;QUANDO NECESSARIA

SIOANF1:XOR A
      LD    B,A
      CP    (HL)        ;BUFFER VUOTO (CONTATORE=0)?
      JR    Z,RDCH2           ;RITORNA 0
      JR    RDCH1       ;O IL CARATTERE SE C'E'

;-------------------------------------------------------------------------
;     FUNZIONE N.9 - FLUSH - AZZERA BUFFER DI RICEZIONE E SIO
;-------------------------------------------------------------------------
FLUSH:      XOR   A
      LD    (CHR_CNT),A
      LD    HL,RX_BUF
      LD    (RD_PO),HL
      LD    (RX_PO),HL
      RET               ;ACC GIA' = 0


;-------------------------------------------------------------------------
;     FUNZIONE N.10 - CHCNT - RITORNA N.CARATTERI NEL BUFFER
;-------------------------------------------------------------------------
CHCNT:      LD    A,(CHR_CNT)
      LD    E,A
      XOR   A
      RET

;-------------------------------------------------------------------------
;     FUNZIONE N.88H - AUTOSTART SISTEMA
;-------------------------------------------------------------------------
AUTO:
;     ESEGUIRE QUI LA LETTURA DELLO SWITCH E SETTARE FLAG
;     OGNI HW LO FA A MODO SUO
SIOBC EQU   0A3H        ;FMD USA BIT CTS DELLA SIO B
      IN    A,(SIOBC)
      BIT   5,A         ;(CTS) SW1-2 ON = AUTOSTART
      LD    A,10H       ;QUANDO SI LEGGE DA BIT SIO POI SI DEVE
      OUT   (SIOBC),A         ;IMPARTIRE UN 'RESET EXTERNAL'
      JR    Z,AUTOST          ;READ DATA=0 = SW ON
NOAUTO:     XOR   A
      JR    AUTANY
AUTOST:     LD    HL,STUPSTG        ;STARTUP STRING
      LD    A,1
AUTANY:     LD    E,A
      XOR   A
      RET

;-------------------------------------------------------------------------
;     SERVIZIO INTERRUPT RICEZIONE CARATTERE
;-------------------------------------------------------------------------
```

```
SIORXA:     EX    AF,AF'            ;SCAMBIA I REGISTRI
      EXX

      LD    A,(DTAP)          ;LEGGI IL DATO
      LD    C,A
      IN    A,(C)

      AND   7FH         ;VIA MSB
      CP    ESC         ;PREMUTO ESC?
      JR    NZ,NOESC
      CALL  FLUSH       ;AZZERA BUFFER
      LD    A,ESC       ;E INSERISCICI ESC

NOESC:      CP    XON         ;IGNORA XON/XOFF
      JR    Z,QSIORX

      CP    XOFF
      JR    Z,QSIORX

      LD    B,A         ;SALVA CARATTERE RICVUTO

      LD    A,(CHR_CNT) ;INCREMENTA CONTA CARATTERI
      INC   A           ;MA SENZA RISALVARLO
      LD    D,A         ;SALVALO INVECE IN D

      CP    BUFLEN-10         ;PIU' DI BUFLEN-10 CARATTERI? (OSSIA 9)?
      JR    C,SANTFLL         ;NO, SALTA A SIO A NOT FULL

                        ;IL BUFFER CONTIENE PIU' DI BUFLEN-10 CARATT.
      CP    BUFLEN-1          ;SE IL TERMINALE INSISTE MALGRADO GLI XOFF
      JP    Z,QSIORX          ;A BUFFER PIENO IGNORA TUTTO CIO' CHE ARRIVA

      LD    E,XOFF            ;INVIA XOFF (SE DOVESSERO ARRIVARE ALTRI
      CALL  SIOTX       ;CARATTERI SI RINVIA ANCORA XOFF)

      LD    A,1         ;SETTA FLAG CHE DICE CHE HAI SPEDITO XOFF
      LD    (XOFSNT),A

SANTFLL:    LD    A,D         ;RIPRENDI D       LD    (CHR_CNT),A ;ED
AGGIORNA CONTACARATTERI

      LD    HL,(RX_PO)        ;PRENDI IL PUNTATORE
      LD    (HL),B            ;SALVA CARATTERE

      CALL  INCPOIN           ;INCREMENTA PUNTATORE
      LD    (RX_PO),HL        ;E SALVALO

QSIORX:     EXX
      EX    AF,AF'
      EI
      RETI

;-------------------------------------------------------------------------
;     RICEZIONE CARATTERE IN CASO DI ERRORE (SERVIZIO INTERRUPT)
;-------------------------------------------------------------------------
```

```
SIOSPA:     EX    AF,AF'            ;IN CASO DI ERRORE
      LD    A,(CTLP)
      LD    C,A
      LD    A,30H       ;ERROR RESET
      OUT   (C),A
      EX    AF,AF'
      EI
      RETI

;---------------------------------------------------------------------------
;     VETTORI SIO NON UTILIZZATI
;---------------------------------------------------------------------------
SIOTXA:
SIOSTA:     EI
      RETI


;---------------------------------------------------------------------------
;     ROUTINES AUSILIARIE
;---------------------------------------------------------------------------

INCPOIN:INC HL          ;ROUTINE PER INCREMENTARE PUNTATORE
      LD    A,H         ;TEST MSBYTE
      CP    .HIGH.ENDBUF      ;FINE BUFFER?
      RET   NZ
      LD    A,L
      CP    .LOW.ENDBUF
      RET   NZ
      LD    HL,RX_BUF         ;RICOMINCIA CIRCOLARMENTE
      RET

;---------------------------------------------------------------------------
;     TABELLE E VETTORI
;---------------------------------------------------------------------------

TABSIOA:
      DB    4,44H       ;CLOCK X 16 - NO SYNC - 1 STOP BIT
                        ;NO PARITY  DB    3,0C1H            ;RX ENABLE - 8
BIT CHAR
                        ;
      DB    5,6AH       ;8 BIT CHAR - TX ENABLE - RTS
                        ;
      DB    10H         ;RESET EXT/STATUS INT
      DB    1,1CH       ;INT ON ALL RX CHARS - PARITY DOES NOT
                        ;AFFECT VECTOR - NO INT ON TX

TABVECS:    DW    SIOTXA
      DW    SIOSTA
      DW    SIORXA
      DW    SIOSPA


STUPSTG:DB  'RUN 19000',CR

DRVEND:
```

```
;-------------------------------------------------------------------------
;     AREA RAM
;-------------------------------------------------------------------------
      DSEG

RD_PO:      DS    2           ;PUNTATORE DI LETTURA BUFFER
RX_PO:      DS    2           ;PUNTATORE DI SCRITTURA BUFFER

;     ATTENZIONE - LE TRE LOC CHE SEGUONO DEVONO STARE CONTIGUE

CTLP: DS    1           ;SIO PORTA CONTROLLO
DTAP: DS    1           ;SIO PORTA DATI
VECP: DS    1           ;SIO PORTA VETTORE

RX_BUF:     DS    BUFLEN
ENDBUF:                 ;FINE BUFFER
CHR_CNT:    DS    1           ;CONTA CARATTERI RICEVUTI
XOFSNT:     DS    1           ;FLAG DI XOFF GIA' SPEDITO (SE=1)
```

# Un programma esempio

Si riporta a titolo di esempio un listato parziale di un programma GBASIC.

Si tratta di un data logger per accumulare i dati provenienti da una centralina anemometrica. Il sistema reale dispone di un display LCD 2x40 e di 6 pulsanti come interfaccia utente (per gentile concessione della Cite spa).

Come si può notare analizzando il listato, molte funzionalità di basso livello, come la misurazione del periodo del segnale proveniente dall'anemometro, somo svolte da drivers.

```
#DECLCONST
'-------------------------------------------------------------------------
'      costanti varie
'-------------------------------------------------------------------------
       VERSIONE $ "STAMP"      'STAMP e' parola riservata per ora e data
       SI            = "1"
       NO            = "0"
       FOROUT        = "0"
       FORINP        = "1"

       STORPTR       = _15        'costanti usabili in info()
       FREEWS        = _16
       FREESTOR      = _9

       MAXSAMPLE     = _10000     'numero massimo di campioni accumulabili

' -----       funzioni BIOS
       RDCHAR = _5
       GETCH  = _8
       INKEY         = _8
       FLUSH         = _9
       CHCNT         = _10

       SETTIMER      = _4

       REC1          = _20
       REC2          = _40

'-------------------------------------------------------------------------
'      drivers
'-------------------------------------------------------------------------
       CONSOLE       = _0        'canale console
       LCD           = _1         'lcd
       CCAL          = _2           'clock tempo reale
       PWMES         = _3         'misuratore periodo
       KEYPAD = _4          'tastierina
'-------------------------------------------------------------------------
'       ldev's
'-------------------------------------------------------------------------
       RAMFILE  = "5"          'canale storages

'-------------------------------------------------------------------------
'      porte
'-------------------------------------------------------------------------
       KEYB   = _176        'porta tastiera
       WIND   = _168        'porta impulsi da anemometro
       WINDBIT = _0         'bit impulsi da anemometro

'-------------------------------------------------------------------------
'      tasti
'-------------------------------------------------------------------------
       ANYKEY        = _0
       NOKEY         = _255

'      tasti
       F1            = _83
       F2            = _68
       F3            = _69
       F4            = _46
```

```
       F5            = _48
       F6            = _13

       X1     = "10.9"
       X2     = "98.8"
       Y1     = "4.67"
       Y2     = "32.06"
#END

#DECLVAR
       n,i,j,k
       a,b           'coefficienti retta
       wind          'uso temp. per vento
       dir           'uso temp. direzione
                     'uso temp. temperatura
       wnd           'periodo da anemometro in msec
       med           'valore medio di Twnd
       freq          'frequenza anemometro in Hz
       Wind          'velocità del vento

       Key           'numero del tasto ritornato da GetKey

       NextRec       'prossimo record utilizzabile per la registrazione
       Logging       'se non zero siamo in registrazione

       TimerSet      'stringa per comando timer
#END

#CODE
       rem
       goto Main
'------------------------------------------------------------------------
'      routines di interrupt
'------------------------------------------------------------------------
#-> Second
       return timer 1
'------------------------------------------------------------------------
'      programma principale
'------------------------------------------------------------------------
#-> Main
       gosub IniSys

       if Logging = 1 then gosub LogMore
#-> MainMenu
       display 1, "      Eolo v.1.0 - "; %"I4D0F "; NextRec; %"I3D0F "; "(";
       int(NextRec*100/MAXSAMPLE); "%)"
       display 2, "    |LOG |MEAS|ESAM|DUMP|CLEAR|SETUP|   "
       gosub WaitNoKey
#-> Loop
       et key = bios (KEYPAD, INKEY)

       if key = F1 then gosub Log : goto MainMenu
       if key = F2 then gosub Measure : goto MainMenu
       if key = F3 then gosub ExamMem : goto MainMenu
       if key = F4 then gosub Dump : goto MainMenu
       if key = F5 then gosub ClearMem : goto MainMenu
       if key = F6 then gosub Setup : goto MainMenu
       goto Loop

'------ registrazione dati
#-> Log
       if NextRec = MAXSAMPLE then MemFull
       if NextRec <> 0 then
```

```
              display 1,"Ok to resume data logging?"
       else
              display 1,"Ok to start data logging?"
       endif
       gosub Confirm

       if key <> F5 then return

       if NextRec = 0 then
              storage MAXSAMPLE + REC1
              print ldev RAMFILE, date$(); " "; time$()
       endif

#-> LogMore

       display 1, "Measuring..."
       display 2

       gosub WindSpeed
       storage MAXSAMPLE + 1, > NextRec
       let Logging = 1

       display 1, "Recording sample # ";%"I4D0F0"; NextRec
       display 2, "Wind speed: "; %"I3D1F "; Wind;"         |STOP LOG|"

       storage NextRec
       let j=info(STORPTR)
       poke j [Tmed]

       if NextRec = MAXSAMPLE then MemFull

       let NextRec = NextRec + 1
       storage MAXSAMPLE + 1, < NextRec, Logging

       for i=1 to 100
              gosub AnyKey
              if key = ANYKEY then QLog
       next

       display 1, "Switching off..."
       display 2

       storage MAXSAMPLE + REC2, >TimerSet$
       write ch CCAL, SETTIMER, TimerSet$
#-> Self
       goto Self

#-> MemFull
       display 1, "Memory full"
       gosub WaitKey
       return
#-> QLog
       let Logging = 0
       storage MAXSAMPLE + 1, < NextRec, Logging
       return
.............
.............
.............

'------       stampa memoria
#-> Dump
       if NextRec = 0 then return
       storage MAXSAMPLE+REC1
```

```
       input ldev RAMFILE, a$
       print
       print "D";a$
       for i=0 to NextRec - 1
              storage i
              let j = info(STORPTR)
              let Tmed = peek(j)
              gosub Convert
              print "S"; %"I4D0F0"; i; %"I3D1F "; Wind
       next
       return

'------      esame della memoria
#-> ExamMem
       if NextRec = 0 then return 'memoria vuota
       storage MAXSAMPLE + REC1
       input ldev RAMFILE, a$
       display 1, "Logging started on ";a$
       gosub WaitKey

       let i=0
       gosub WaitNoKey
       display 2, "    | -  | +  | ++ |1.st|LAST |DONE |   " #-> ExamMem1
       storage i
       let j = info(STORPTR)
       let Tmed = peek(j) '       let dir = peek(j+1) '      let t = peek(j+2)
       gosub Convert '
       display 1,  "Sample:"; %"I4D0F0"; i; %"I3D1F ";
& Wind; "/"; dir; "/"; t; "        "
       display 1,  "Sample:"; %"I4D0F0"; i; %"I3D1F "; Wind; "m/sec"
#-> ExamMem2
       let key = bios (KEYPAD, INKEY)
       if key = F1 then if i > 0 then let i = i - 1 : goto ExamMem1
       if key = F2 then if i < NextRec-1 then let i = i + 1 : goto ExamMem1
       if key = F3 then if i < (NextRec-10) then let i = i + 10 : goto
& ExamMem1
       if key = F4 then let i= 0 : goto ExamMem1
       if key = F5 then let i = NextRec-1 : goto ExamMem1
       if key = F6 then return
       goto ExamMem2
...................
...................
...................

'------ setup macchina
#-> Setup
       display 1
       display 2, "    |TIME|DATE|INTV|    |ABOUT|DONE |   "
       gosub WaitNoKey
#-> Loop1
       let key = bios (KEYPAD, INKEY)
       if key = F1 then gosub SetTime : goto Setup
       if key = F2 then gosub SetDate : goto Setup
       if key = F3 then gosub SetInterval : goto Setup
       if key = F5 then gosub About : goto Setup
       if key = F6 then return
       goto Loop1
#-> About
       display 1, "Cite spa - Eolo v.1.0 - GBI v.7.4"
       display 2, "Compiled on ";VERSIONE
       gosub WaitKey1
       return
#-> SetInterval
```

```
       display 1, "Set sampling interval to:"
       display 2, "    |1min|5min|15mi|30 m|45min|60min|   "
       gosub WaitNoKey #-> Loop2
       let key = bios (KEYPAD, INKEY)
       if key = F1 then let TimerSet$="23:59:00"  : goto SetInt1
       if key = F2 then let TimerSet$="23:55:00"  : goto SetInt1
       if key = F3 then let TimerSet$="23:45:00"  : goto SetInt1
       if key = F4 then let TimerSet$="23:30:00"  : goto SetInt1
       if key = F5 then let TimerSet$="23:15:00"  : goto SetInt1
       if key = F6 then let TimerSet$="23:00:00"  : goto SetInt1
       goto Loop2
#-> SetInt1
       storage MAXSAMPLE + REC2, <TimerSet$
       return
'-------------------------------------------------------------------------
'      WaitKey - attende un tasto e lo lascia in Key
'-------------------------------------------------------------------------
#-> WaitKey
       display 2, "Press any key to continue"
#-> WaitKey1
       gosub WaitNoKey #-> WaitKey2
       let key = bios (KEYPAD, INKEY)
       if key = 0 then WaitKey2
       return
 '-------------------------------------------------------------------------
'      WaitNoKey - attende che nessun tasto sia premuto '--------------------------
-----------------------------------------
#-> WaitNoKey
       if or(inp(KEYB),128) <> NOKEY then WaitNoKey
       return

'-------------------------------------------------------------------------
'      AnyKey - ritorna in key ANYKEY o NOKEY
'-------------------------------------------------------------------------
#-> AnyKey
       let key = or(inp(KEYB),128)
       if key <> NOKEY then let key = ANYKEY
       return

'-------------------------------------------------------------------------
'      Confirm - attende un tasto e lo lascia in Key
'-------------------------------------------------------------------------
#-> Confirm
       display 2, "Confirm?               | YES | NO  |   "
       gosub WaitKey1
       return

'-------------------------------------------------------------------------
'      WindSpeed - misura e ritorna la velocità del vento '-------------------------
-----------------------------------------
#-> WindSpeed
..................
..................
..................

'-------------------------------------------------------------------------
'      rimessa orologio
'-------------------------------------------------------------------------
#-> SetTime
       display 1, "Ora attuale: ";time$()
       display 2, "    |    |    |    |RFSH|EDIT |DONE |   " #-> SetTime1
       let key = bios(KEYPAD, INKEY)
       if key = F6 then return
```

```
       if key = F4 then SetTime
       if key <> F5 then SetTime1
       display 2, "    | <- | -> | +  | -  |     |DONE |   "
       display 1
       let a$ = time$()
       let a$ = nget$(a$)
       if cktim(a$) <> 0 then time$(a$)
       gosub WaitNoKey
goto SetTime

'-------------------------------------------------------------------------
'      rimessa data
'-------------------------------------------------------------------------
#-> SetDate
       display 1, "Data attuale: ";date$()
       display 2, "    |    |    |    |RFSH|EDIT |DONE |   " #-> SetDate1
       let key = bios(KEYPAD, INKEY)
       if key = F6 then return
       if key = F4 then SetDate
       if key <> F5 then SetDate1
       display 2, "    | <- | -> | +  | -  |     |DONE |   "
       display 1
       let a$ = date$()
       let a$ = nget$(a$)
       if ckdat(a$) <> 0 then date$(a$)
       gosub WaitNoKey
       goto SetTime
...................
...................
...................

'-------------------------------------------------------------------------
'    inizializzazione
'-------------------------------------------------------------------------
#-> IniSys
       reset interrupt clear

       dim a$(50), TimerSet$(20)

       open ch KEYPAD, "KPAD:"
       con: FORINP is ch KEYPAD
       open ch LCD,"LCD1:"               'apre canale display lcd
       aux: 0 is ch LCD
       open ch CCAL,"CCAL:","D 188"       'apre canale rtc
       timdat: is ch CCAL
'      open timer 1,"C 2 D 130"
'      prescale 75                        'interrupt ogni secondo
'      on timer 1 gosub Second '  load timer 1 with 1 freerun
       open ch PWMES, "PWMES:", "C 2 D 130"
       write ch PWMES, 0, =256+9
       storage = 1                'dimensione di ogni campione accum.
       storage MAXSAMPLE + 1, > NextRec, Logging
       let a = (Y1-Y2) / (X1-X2)
       let b = Y1 - a * X1
       interrupt SI
       return
#END
```

# Indice

Introduzione ..... 5
Descrizione generale ..... 5
Sistema tipo ..... 7
Ciclo di sviluppo ..... 7
Lo Z-80 ..... 8
I drivers ..... 8
Limiti ..... 8
Precompilatore PREGB ..... 10
Generalità ..... 10
Altri files generati ..... 11
Contenuto del dischetto ..... 12
Installazione ed uso ..... 12
La sintassi estesa ..... 13
File sorgente ..... 14
Identificatori ..... 14
Campo di validità ..... 15
Struttura di un programma .GBE ..... 16
Procedure ..... 17
Dichiarazione delle variabili ..... 18
Dichiarazione delle costanti ..... 19
Costanti numeriche nuovo tipo ..... 20
STAMP ..... 21
Il programma ..... 21
Compilazione condizionale ..... 22
Inclusione di files ..... 22
Time-stamping ..... 23
Diagnostica ..... 23
Conclusione ..... 24
Guida dell'utente ..... 25
Generalita' ..... 25
Le stringhe di configurazione ..... 26

Libera riproduzione e libero uso

Stringa di configurazione della SIO....26
Frase let....29
Vettori e stringhe....30
Editing....31
L'ingresso / uscita (I-O)....32
I dispositivi logici....34
Preliminari sull'interrupt....37
Le stringhe di configurazione....39
Periferiche autogestite....41
Interruzione ad alto livello....43
Timers e contatori....47
Oscillatori....49
Complemento sui drivers....50
Organizzazione della memoria....51
Autostart....52
Segmenti critici....54
MMT....56
Alcuni tempi di esecuzione....57
Riferimento....61
Contenuto....61
Generalità....61
Accensione e reset....61
Caratteristiche generali del GBASIC....62
Convenzioni e notazioni....64
Abbreviazioni....65
Operatori....66
I comandi procedurali....67
Operatore ^ (PRIVILEGED)....67
AUTOSTART....67
AUX:....68
BACKSP....69
BEEP....69
BOOT....69
BREAK....70
CLEAR....71
CLOSE....71
CON:....72
COUNTER....73
CRITSEG....73
DATA....73
DATE$(x$)....74
DIM....75
DISPLAY....76
END....76
ENDCRIT....77
ERR:....77
FOR....78
GOSUB....79
GOTO....79
IF....80
INPUT....82
INTERRUPT....90
LET....90
LOAD TIMER....91
LPRINT....92

NOVRAM....93
NEXT....93
ON....94
OPEN....96
OSC....98
OUT....99
OVERPRINT....100
POP....100
POKE....101
PRESCALE....102
PRINT....102
PRINTS....107
READ....107
REM....109
RESET....109
RESTORE....110
RETURN....111
SIOSET....112
STOP....113
STORAGE....114
TIMDAT:....119
TIMER....119
TIME$(x$)....120
WRITE....120
WRITE....121
Funzioni....121
&H(x$)....121
ADDROF(n)....122
AND(m,n)....122
ARG(x)....122
ASC(x$)....123
ABS(x)....123
BIOS(x)....124
CALL(x)....125
CHR$(x)....126
CKDAT(x$)....126
CKSUM(x$)....127
CKTIM(x$)....127
CONCAT$(x$,y$)....128
COUNT(x)....129
COS(x)....129
DATE$()....130
DIAG(n)....130
DRVST(x),DRVST(x,y)....131
EXIT(0)....132
HEX$(x)....132
INFO(x)....132
INP(x) INP(x,y)....134
INT(x)....135
LEN(x$)....135
NGET$(x$)....136
NIBBLES(x$)....137
NOT(x,y)....137
OR(x,y)....137
PEEK(x)....138

RND(x) ....138
SECS(x) ....139
SGN(x) ....139
SHUTDN(0) ....140
SIN(x) ....140
SQR(x) ....140
STR$(x) ....141
SUBSTR$(x$,x,y) ....141
TAN(x) ....142
TIME$() ....142
TIMSEC(x) ....143
VAL(x$) ....143
XOR(m,n) ....144
Comandi ....145
ERASE ....145
LIST ....145
LLIST ....146
LOAD ....146
RUN ....147
SAVE ....148
Come scrivere un driver ....149
Generalità ....149
Drivers block o char oriented ....151
Posizione fisica dei drivers ....151
Struttura fisica di un driver ....152
Uso della RAM da parte dei drivers ....153
Funzioni eseguite dal driver ....153
Funz. driver #0 Block Write ....153
Funz. driver #1 Block Read ....154
Funz. driver #2 Init ....154
Funz. driver #3 Close ....155
Funz. driver #4 Write Char ....155
Funz. driver #5 Read Char ....156
Funz. driver #6 Status ....156
Funz. driver #7 Info ....157
Funz. driver #8 Getch ....157
Funz. driver #9 Flush ....158
Funz. driver #10 Chnum ....158
Funz. driver #88h Autostart ....159
Il "System" ....159
Funz. system #0 PSH_USTK ....160
Funz. system #1 ASK_MY_LEV ....161
Funz. system #2 HIRQ ....161
Funz. system #3 INSTVEC ....162
Funz. system #4 PARSEP ....163
Funz. system #5 PARSEP2 ....164
Funz. system #6 CSERR ....165
Funz. system #7 TRIPLEX ....165
Funz. system #8 REPLDRV ....166
Funz. system #9 RPLBKSP ....167
Funz. system #10 NEWTASK ....167
Funz. system #11 STOPALL ....168
Funz. system #12 DOCKSUM ....168
Funz. system #13 CRITSEG ....169
Funz. system #14 ENDCRIT ....169

Funz. system #15 &BOF....169
Funz. system #16 &EOF....170
Funz. system #18 CNS....170
Funz. system #19 INTMON....171
Funz. system #20 QRYMMT....172
Funz. system #21 QRYFREQ....172
Funz. system #22 DRADDR....173
Funz. system #23 POPTASK....173
Mini Multi Task (MMT)....174
Power down....175
Il driver di console....177
Inizializzazione porte....177
Autostart da RAM....178
Un driver block oriented....179
Drivers per dispositivo logico AUX:....179
Richiesta interrupt ad alto livello....180
Drivers associati al dispositivo AUX:....180
Messaggi di errore....182
Messaggi di errore....182
SINT....182
VAL....182
F/NXT/RET 1 (o F/NXT/RET 2)....183
FOR/NXT....183
RET....183
IDX....183
DIM....183
MEM....183
INT NON PREDIS....184
INT#....184
STG....184
READ....184
INPUT....184
STK....184
RAD NEG....184
NUM LIN....185
MODO....185
LN LUNGA....185
CH....185
ARITM....185
DEV INES....185
DIV PER 0....185
GIA' OPEN....186
NON OPEN....186
NOME....186
VETT INT....186
BUG....186
C STG....186
MMT....186
I STK OVFL....187
Listato di CONS.DRV....188
Un programma esempio....197