BASIC 52

MANUALE D' USO

Interprete BASIC per famiglia 51 Intel

grifo®
ITALIAN TECHNOLOGY

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

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

BASIC 52 Edizione 3.0 *Rel. 13 Maggio 1996*

abaco®, **GPC®**, **grifo®**, *sono marchi registrati della ditta* **grifo®**

BASIC 52

MANUALE D' USO

BASIC 52 é un potente tool software sviluppato dalla **grifo**®, che consente la programmazione ad alto livello (BASIC), su tutte le schede basate sulla famiglia 51 Intel (**GPC® F2**, **GPC® 51**, **GPC® 552**, etc). Tale BASIC viene eseguito da eprom e genera un codice "romabile" che viene eseguito dall' eeprom parallela di bordo; si riduce così la necessità di hardware esterno (in circuit emulator, EPROM programmer, etc.) e allo stesso tempo si velocizza la fase di debug del programma applicativo. **BASIC 52** é il riferimento a pacchetti software genereci, ma per ciascuna scheda esiste una relativa versione di software implementata per gestire le differenti periferiche di bordo; perciò per ogni scheda il nome **BASIC 52** diventa BASIC e di seguito la parte finale del nome della scheda.

grifo®
ITALIAN TECHNOLOGY

Via dell' Artigiano, 8/6
40016 San Giorgio di Piano
(Bologna) ITALY
E-mail: grifo@grifo.it
http://www.grifo.it http://www.grifo.com
Tel. +39 051 892.052 (r.a.) FAX: +39 051 893.661

BASIC 52 Edizione 3.0 *Rel. 13 Maggio 1996*

abaco®, **GPC®**, **grifo®**, *sono marchi registrati della ditta* **grifo®**

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

Nessuna parte del presente manuale può essere riprodotta, trasmessa, trascritta, memorizzata in un archivio o tradotta in altre lingue, con qualunque forma o mezzo, sia esso elettronico, meccanico, magnetico ottico, chimico, manuale, senza il permesso scritto della **grifo**®.

IMPORTANTE

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

LEGENDA SIMBOLI

Nel presente manuale possono comparire i seguenti simboli:

Attenzione: Pericolo generico

Attenzione: Pericolo di alta tensione

Marchi Registrati

abaco® , **GPC**®, **grifo**® : sono marchi registrati della **grifo**®.
Altre marche o nomi di prodotti sono marchi registrati dei rispettivi proprietari.

INDICE GENERALE

INTRODUZIONE

In questo capitolo vengono descritte quali sono le operazioni da effettuare per un primo elementare utilizzo del pacchetto software. In particolare:

1) Leggere tutta la documentazione ricevuta.

2) Predisporre la scheda remota per operare (alimentazione, verifica di configurazione, ecc.).

3) Effettuare il collegamento seriale tra terminale e scheda seguendo le relative informazioni presenti sul manuale tecnico della scheda.

4) Alimentare la scheda remota.

5) Premere una o più volte la barra spaziatrice, fino a quando sul monitor del terminale compare il messaggio di presentazione del BASIC 52 seguita dal relativo prompt:

READY>

DESCRIZIONI GENERALI

Il BASIC 52 può operare in due modi: COMMAND MODE (modo diretto) o RUN MODE.
In COMMAND MODE il BASIC 52 esegue il comando appena lo si é confermato.
Un programma in basic é costituito da un insieme di istruzioni. Ogni linea di programma inizia con un numero di linea, segue poi il "corpo" dell' istruzione e termina con un Carriage Return (cr), o con i due punti (:) in caso di linee composte da più istruzioni.
Alcune istruzioni possono essere eseguite anche in COMMAND MODE, ma la maggior parte può essere eseguita solo in RUN MODE.
Ci sono tre tipi di istruzioni in BASIC 52:

A) di ASSEGNAZIONE.

B) di INPUT/OUTPUT.

C) di CONTROLLO.

Il BASIC 52 segue le seguenti regole e perciò anche il programmatore deve tenerne conto:
- Ogni linea di programma deve avere un numero linea che sia compreso tra 0 e 65535 inclusi.
- Il BASIC 52 utilizza il numero linea per ordinare in modo sequenziale il programma.
- In un programma uno stesso numero linea non può identificare più linee ossia deve comparire solo una volta.
- Il BASIC 52 automaticamente ordina in modo crescente le linee di programma.
- Una linea di programma può contenere al massimo 79 caratteri.
- Il BASIC 52 ignora gli spazi e durante il LIST automaticamente ne aggiunge.
- Su una stessa linea si possono inserire più istruzioni e bisogna separarle con i due punti ":".

COSA SUCCEDE DOPO UN RESET

Dopo un reset il BASIC 52 effettua le seguenti operazioni:
1) Reset di tutta la ram interna del micro-processore.
2) Inizializzazione dei puntatori e dei registri interni.
3) Test, cancellazione e determinazione della quantità di memoria esterna.
4) Assegnazione alla variabile di sistema MTOP di un valore che esprime la quantità di memoria esterna.
5) Assegnazione alla variabile di sistema XTAL di un valore che esprime la frequenza di default del quarzo (11.0592 Mhz) la quale verrà utilizzata in tutte quelle funzioni che gestiscono il real time clock.
6) Test della locazione 08000H nella memoria esterna per verificare se le informazioni a riguardo del baud-rate erano già state salvate. Se sono già presenti il BASIC 52 salta la procedura di auto-determinazione del baud-rate e procede, altrimenti viene interrogata la porta seriale e si attende di ricevere il valore 020H (space).

FORMATO DEI NUMERI

Il range di numeri che il BASIC 52 può rappresentare è: ± 1E-127 to ± .99999999E +127.
Il BASIC 52 gestisce una precisione di 8 cifre e perciò internamente i numeri sono "arrotondati" di conseguenza. I numeri possono essere inseriti o visualizzati in 4 formati: intero, decimale, esadecimale ed esponenziale (es. 129, 34.98, 0A6EH, 1.23456E+3).

I NUMERI INTERI

Gli interi in BASIC 52 sono numeri compresi nel range 0÷65535. Tutti i numeri interi possono essere inseriti sia nel formato decimale che in quello esadecimale. Quando viene utilizzato un operatore tipo .AND., il BASIC 52 tronca la parte frazionaria del risultato e quindi il risultato diventa un INTEGER.

LE COSTANTI

Una costante é un numero reale compreso nel range ±1 E-127 ÷ ±99999999 E+127.
Una costante naturalmente può essere un intero.

GLI OPERATORI

Un operatore, effettua una operazione predefinita su variabili e/o costanti.
Gli operatori possono richiedere uno o due operandi. Operatori tipici che richiedono due operandi sono: +, -, *, /. Gli operatori unari, sono quelli che richiedono un solo operando tipo: SIN, COS e ABS.

ORDINE DI VALUTAZIONE DEGLI OPERATORI

L' operazione 7+3*2 dà come risultato 20 o 13 ? Dipende da quale operando é valutato per primo. Per ovviare a tale problema, questo BASIC come molti altri linguaggi, presenta una lista di precedenza. Gli operatori che hanno una precedenza maggiore, vengono valutati per primi, in caso si debbano valutarne due aventi lo stesso "grado" di precedenza, si procede da sinistra a destra. Gli operatori nelle parentesi vengono valutati prima di quelli fuori. La tabella seguente, mostra una lista che esprime il grado di precedenza degli operandi, perciò più si scende nella lettura e più il relativo operatore ha un grado minore di precedenza:

1) OPERATORI CHE USANO LE PARENTESI ()

2) ESPONENZIALE (**)

3) NEGAZIONE (-)

4) MOLTIPLICAZIONE (*) e DIVISIONE (/)

5) ADDIZIONE (+) e SOTTRAZIONE (-)

6) ESPRESSIONI RELAZIONALI (=, <>, >, >=, <, <=)

7) AND LOGICO (.AND.)

8) OR LOGICO (.OR.)

9) XOR LOGICO (.XOR.)

LE VARIABILI

In BASIC 52 le variabili possono essere definite tramite lettere e numeri, e possono avere una lunghezza massima di 8 caratteri. Di seguito vengono riportati esempi di definizione corretta di variabili: FRED VOLTAGE1 I_I1 ARRAY(ELE_1)
Si ricorda che l' utente non può inserire nel nome della variabile le parole chiavi altrimenti verrà generato un BAD SYNTAX ERROR; per esempio nomi variabili tipo TABLE e DIET non possono essere utilizzati in quanto TAB e IE sono parole riservate.
Il BASIC 52 alloca le variabili in modo statico, e ciò significa che ogni volta che una variabile viene utilizzata, esso alloca una porzione di memoria (8 bytes) specificatamente per quella variabile. Purtroppo non esiste la possibilità di rendere libere solo alcune delle porzioni di memoria utilizzate per l' allocamento delle variabili, e perciò l' unico modo per rendere libera tale parte di memoria é quello di utilizzare l' istruzione CLEAR che libera "tutta" la memoria.
Si consiglia per ottenere una velocità di esecuzione maggiore, di utilizzare in gran parte le variabili scalari, gli array devono essere utilizzati solo se effettivamente servono.

LE STRINGHE

Una stringa rappresenta un carattere o una serie di caratteri salvati in memoria. La gestione delle stringhe é utile in quanto permette al programmatore di lavorare con parole e non con numeri e ciò é molto significativo ai fini della buona leggibilità e manutenzione del programma. Il BASIC 52 é in grado di gestire 255 variabili di tipo stringa e la sintassi per definire una stringa é la seguente: $([expr]) dove [expr] può assumere i valori compresi nel range 0÷254. Inizialmente per le stringhe non viene allocata memoria, infatti é una operazione che viene eseguita dall' istruzione STRING [expr],[expr]. In BASIC 52 le stringhe possono essere definite in due modi, con l' istruzione LET o con l' istruzione INPUT, esempio:

```
>10 STRING 100,20
>20 $(1)="THIS IS A STRING, "
>30 $(1)="WHAT' S YOUR NAME ? - ",$(2)
>40 PRINT $(1), $(2)
>RUN

WHAT' S YOUR NAME ? - FRED

THIS IS A STRING, FRED
```

LE ESPRESSIONI

Una espressione é una formula matematica che gestisce operatori, variabili e costanti.
Le espressioni possono essere semplici o complesse:
12*EXP(A)/100, H(1)+55, (SIN(A)*SIN(A)+COS(A)*COS(A))/2.
Le espressioni relazionali sono quelle espressioni che contengono i seguenti operatori:

UGUALE	=
DIVERSO	<>
MAGGIORE DI	>
MINORE DI	<
MAGGIORE O UGUALE A	>=
MINORE O UGUALE A	<=

Essi sono usati nelle istruzioni di controllo e richiedono sempre due operandi.

VALORI DI CONTROLLO SISTEMA

Tali funzioni servono all' utente per essere a conoscenza delle risorse sistema che si stanno utilizzando. In dettaglio:

LEN -->	Contiene la lunghezza del programma utente.
FREE -->	Contiene il numero di bytes di RAM ancora non utilizzati.
MTOP -->	Contiene l' indirizzo dell' ultima locazione di memoria utilizzabile dal basic.

L' EDITOR

Il BASIC 52 è dotato di un editor che permette però una operatività limitata.
Una volta che é stata inserita, una linea non può essere modificata se non reinserendola completamente. E' possibile cancellare un carattere mentre si sta inserendo la linea tramite il carattere RUBOUT o DELETE. Premendo CONTROL S o CONTROL Q durante un LIST o un PRINT, si darà uno stop o uno start alla visualizzazione dei caratteri.

DISABILITAZIONE DEL CONTROL-C

In molte applicazioni é desiderabile od anche richiesto che una esecuzione di programma non venga interrotta accidentalmente. Con una operazione “normale” può essere fermata l’ esecuzione di qualsiasi programma, semplicemente premendo sulla tastiera control-C. Comunque, in BASIC 52 esiste la possibilità di disabilitare la funzione di break del control-C. Questa operazione é compiuta settando il bit 48 (30H) a 1. Tale bit risiede nella memoria interna alla locazione 38.0 (26.0H). Il bit può essere settato in un programma BASIC 52 tramite la seguente istruzione:

DBY(38)=DBY(38).OR.01H

Una volta settato il bit a 1, la funzione di break del control-C é disabilitata per le operazioni di LIST e di RUN. L’ utente ha la possibilità di creare un carattere o una stringa di caratteri per il break da utilizzare con l’ operatore GET. Nel seguente esempio si può vedere come può essere compiuta questa operazione:

```
>50 REM inizializzazione delle stringhe
>100 STRING 110,10
>200 D=1
>250 REM “FERMATA” e` la parola chiave
>300 $(1)= “FERMATA”
>350 REM disabilitazione del control-C
>400 DBY(38)=DBY(38).OR.01H
>450 REM ciclo di attesa
>500 FOR X=1 TO 500
>600 W=COS(X)
>700 T=GET
>800 IF T<>0 THEN 1000 ELSE NEXT X
>900 END
>1000 IF T=ASC($(1),D) THEN D=D+1 ELSE D=1
>1050 REM controllo della parola chiave
>1100 IF D=1 THEN NEXT X
>1200 IF D=7 THEN 1400 ELSE NEXT X
>1300 END
>1400 PRINT “FERMATA”
>1450 REM abilitazione del control-C
>1500 DBY(38)=DBY(38).AND.0FEH
```

In questo esempio, digitando la parola FERMATA si ferma l’ esecuzione del programma, quindi FERMATA é la parola chiave.

ANOMALIE

Le anomalie sono qualcosa di sbagliato all' interno di un programma. Come tutti i programmi, anche il BASIC 52 contiene alcune anomalie. Lo scopo di menzionare le anomalie conosciute é quello di aiutare molto spesso il programmatore, in quanto durante l' esecuzione potrebbero succedere cose strane. Le anomalie conosciute che interessano principalmente il BASIC 52 fanno parte anche del programma BASIC. Le anomalie conosciute e la loro prevenzione sono riportate qui sotto:

A) Quando usando una variabile H dopo il numero di linea, bisogna essere sicuri di inserire uno spazio tra il numero di linea ed H, altrimenti il BASIC 52 assume il numero di linea come un numero esadecimale.

Esempi:

SBAGLIATO	CORRETTO
>40H=5	>40 H=5
>LIST	>LIST
32 =5	40 H=5

B) Quando utilizzando la variabile I prima di un' istruzione ELSE, bisogna essere sicuri di porre uno spazio tra tra I e l' istruzione ELSE, altrimenti il BASIC assume la porzione IE di IELSE come l' operatore di funzione speciale IE.

Esempi:

SBAGLIATO	CORRETTO
>10 IF I<2 THEN PRINT IELSE 30	>10 IF I<2 THEN PRINT I ELSE 30
> LIST	> LIST
10 IF I<2 THEN PRINT IELSE 30	10 IF<2 THEN PRINT I ELSE 30

C) Un carattere spazio non può essere posto all' interno di un operatore ASC(). In altre parole, un' istruzione come PRINT ASC() provoca un BAD SYNTAX ERROR. Comunque gli spazi posti internamente ad una stringa, come in un' istruzione LET $(1)="CIAO, COME STAI?", non provocano alcun genere di anomalia. La ragione per la quale ASC() produce un errore é perché il BASIC 52 elimina tutti gli spazi nella linea che sta eseguendo, così ASC( ) sarà immagazzinato come ASC() e il BASIC 52 lo interpreta come un errore.

BAUD

Istruzione: **BAUD [espress.]**

Modo: Comando interno o esterno al programma.

Tipo: Controllo

Descrizione:

L'istruzione BAUD [espress.] é usata per settare il baud rate per il port della stampante di linea presente nel dispositivo **BASIC 52**. Affinché questa istruzione determini correttamente il baud rate prescelto, il quarzo (operatore di funzione speciale XTAL) deve essere inizializzato correttamente (XTAL=9000000). Il BASIC 52 assume un valore di quarzo di 11.0592 MHz se non éstato fissato il valore di XTAL. Il port della stampante di linea é P1.7 sul dispositivo CPU. Lo scopo principale del port della stampante é di fornire all' utente una "hard copy" del listato del programma e/o dei dati. Il comando LIST# e l' istruzione PRINT# dirigono le uscite al port della stampante. Se non é stata eseguita un' istruzione come BAUD [espress.] prima di un' istruzione di comando come LIST# o PRINT#, l' uscita del port della stampante di linea sarà a circa 1 BAUD e quindi impiegherà molto tempo per fare uscire qualcosa. Si potrebbe pensare che il BASIC abbia distrutto il listato, ma non é così. Egli sta usando un rate molto lento. Bisogna quindi essere sicuri di assegnare il BAUD Rate al port di stampa, prima di usare LIST# o PRINT#. Il BAUD rate massimo che può essere assegnato dall' istruzione di BAUD dipende dal quarzo. In generale, 4800 é il baud rate massimo, comunque l' utente potrebbe volere sperimentarlo con diversi rate. Il software seriale trasmette 8 bits di dati, 1 bit di start e 2 bits di stop. La trasmissione non comprende controlli di parità.

Esempio:

```
BAUD 2400
```

Causerà un' uscita di dati dal port della stampante di linea a 2400 BAUD.

CALL

Istruzione: **CALL [intero]**

Modo: Comando interno o esterno al programma.

Tipo: Controllo.

Descrizione:

L' istruzione CALL [intero] é usata per chiamare un programma in linguaggio ASSEMBLY. L' intero che segue CALL é l' indirizzo presso il quale l' utente deve riporre la routine di linguaggio ASSEMBLY. Per ritornare al BASIC l' utente deve eseguire un' istruzione di RET in linguaggio ASSEMBLY.
Se il numero intero che segue l' istruzione di CALL, cioé il numero fornito dall' utente, é compreso tra 0 e 127 (7FH), il BASIC 52 moltiplicherà per 2 tale numero, poi gli addizionerà 4100H e, dopo tali operazioni, si otterrà l' indirizzo della locazione in cui sarà posto il primo codice operativo della procedura. Questo significa che CALL 0 chiamerà la locazione 4100H, CALL 1 chiamerà 4102H, CALL 2 chiamerà 4104H e così via. Questo permette all' utente di generare una semplice tabella di routine in linguaggio assembly senza dovere inserire un intero di 4 cifre esadecimali dopo l' istruzione CALL.

Esempio:

```
CALL 700H
```

Farà sì che la CPU esegua un programma di linguaggio ASSEMBLY che inizia alla locazione 700H (il program counter sarà caricato con 700H).

CLEAR

Istruzione: **CLEAR**

Modo: Comando interno o esterno al programma.

Tipo: Controllo.

Descrizione:

L' istruzione di CLEAR setta tutte le variabili uguali a 0 e resetta tutti gli stacks e gli interrupts del BASIC. Questo significa che, dopo che l' istruzione CLEAR é stata eseguita, un' istruzione ONEX1 o ONTIME deve essere eseguita prima che il BASIC 52 acquisisca gli interrupts. L' errore dell' istruzione ONERR non si verificherà fino a che l' istruzione di ONERR [intero] é esecutiva. L'istruzione CLEAR non influenza il Real Time Clock, che é stato abilitato dall' istruzione CLOCK1. CLEAR non resetta anche la memoria che é stata riservata per le stringhe, così non é necessario inserire l' istruzione STRING [espress.] [espress.] per riallocare la memoria per stringhe dopo che é stata eseguita l' istruzione CLEAR.
In generale, CLEAR e' semplicemente usato per "cancellare" tutte le variabili.

Esempio:

```
>10 FOR I=1 TO 10
>20 A(I)=I
>30 NEXT I
>40 CLEAR
>50 FOR I=1 TO 10
>60 PRINT A(I)
>70 NEXT I
>RUN
0
0
0
0
0
0
0
0
0
0
```

Nel caso che mancasse la linea 40, i valori stampati dopo l'esecuzione del programma sarebbero: 1,2,3,4,5,6,7,8,9,10, tutti su un' unica colonna.

CLEARI

Istruzione: **CLEARI** (clear interrupts)

Modo: Comando interno o esterno al programma

Tipo: Controllo

Descrizione:

Cancella tutti gli INTERRUPTS e gli stack richiamati dal BASIC. Specificamente, gli interrupts ONTIME e ONEX1 sono disabilitate dopo che é stata eseguita l' istruzione CLEARI. Questo é stato effettuato dai bit di cancellazione 2 e 3 del registro di funzioni speciali della CPU, cioé l'IE (Interrupt Eneable), e dallo stato dei bit che determinano se il BASIC 52 o l' utente sta controllando questi interrupts. Il Real Time Clock che é stato attivato dall' istruzione CLOCK1 non é stato influenzato da CLEARI. Questa istruzione può essere usata per disabilitare selettivamente gli interrupts durante le sezioni specifiche del programma BASIC dell' utente. Le istruzioni ONTIME e/o ONEX1 devono essere eseguite ancora prima che gli interrupts siano attivati. Quando l' istruzione CLEARI é listata, appare come CLEAR I. NON ALLARMARSI, si può continuare a lavorare.

CLEARS

Istruzione: **CLEARS** (clear stacks)

Modo: Comando interno o esterno al programma

Tipo: Controllo

Descrizione:

Resetta tutti gli STACKS del BASIC 52. Gli STACK di CONTROLLO e di ARGOMENTO sono resettati per il loro valore di inizializzazione, rispettivamente 254 (0FEH) e 510 (1FEH). Lo STACK INTERNO (lo STACK POINTER della CPU, Registro Funzione Speciale-SP) é stato caricato con il valore che é nella locazione 62 (3EH) della RAM INTERNA. Questa istruzione può essere usata per “depurare” lo stack nel caso ci sia un errore nella subroutine. In più, questa istruzione può essere usata per fornire un’ uscita “speciale” dai cicli FOR-NEXT, DO-WHILE, o DO-UNTIL.
Quando l' istruzione CLEARS é listata, appare come CLEAR S. NON ALLARMARSI, si può continuare a lavorare.

Esempio:

```
>10 PRINT “TEST DI ADDIZIONE; 10 SECONDI A DISPOSIZIONE”
>20 FOR I=1 TO 10
>30 N=INT(RND*10)
>40 PRINT “QUANTO FA”,N,”+”,I”?”
>50 CLOCK1: TIME=0
>60 ONTIME 10, 130
>70 INPUT RIS
>80 IF RIS<>N+1 THEN 90 ELSE 100
>90 PRINT “SBAGLIATO, RIPROVA”: GOTO 50
>100 PRINT “ESATTO”: TIME 0
>110 NEXT I
>120 PRINT “CI SEI RIUSCITO, COMPLIMENTI”: END
>130 REM CONTROLLO DELLO STACK SPRECATO: CLEARS
>140 PRINT “TEMPO SCADUTO”: GOTO 10
```

CLOCK0

Istruzione: **CLOCK0**

Modo: Comando interno o esterno al programma.

Tipo: Controllo

Descrizione:

Disabilita la caratteristica del Real Time Clock. Questa istruzione cancella il bit 2 del registro di funzione speciale della CPU, IE. Dopo che il CLOCK0 é stato eseguito, l' operatore di funzione speciale TIME non incrementerà ulteriormente. L' istruzione CLOCK0 inoltre riporta all' utente il controllo degli interrupts associati con TIMER/COUNTER 0, così questo interrupt puo' essere maneggiato a livello di linguaggio assembly. Il CLOCK0 é la sola istruzione del BASIC 52 che può disabilitare il Real Time Clock. CLEAR e CLEAR1 non disabilitano il Real Time Clock.

Esempio:

```
>10 CLOCK0
>20 TIME=0
>30 CLOCK1
>40 DO
>50 TEMPO=INT(TIME)
>60 UNTIL TEMPO>=30
>70 PRINT"SONO TRASCORSI 30 sec DAL LANCIO DEL PROGRAMMA"
>80 END
```

CLOCK1

Istruzione: **CLOCK1**

Modo: Comando interno o esterno al programma.

Tipo: Controllo

Descrizione:

Attiva il Real Time Clock presente nel dispositivo BASIC 52. Lo speciale operatore di Time é incrementato una volta ogni 5 millisecondi dopo che é stata eseguita l' istruzione CLOCK1. Tale istruzione utilizza il TIMER/COUNTER 0 nel modo a 13 bit per generare un' interrupt una volta ogni 5 millisecondi. Questo accade perché l' operatore speciale di TIME ha una risoluzione di 5 millisecondi. Il BASIC 52 calcola automaticamente il proprio valore ricaricato per il TIMER/ COUNTER 0, dopo che il valore del quarzo é stato assegnato (XTAL= valore). Se il valore del quarzo non é stato fissato, il BASIC 52 assume un valore di 11.0592 MHz. L' operatore di funzione speciale TIME conta da 0 a 65535.995 secondi. Dopo avere raggiunto un conteggio di 65535.995 secondi, il TIMER riprende a contare da 0. Perché l' istruzione CLOCK1 usi gli interrupts associati con TIMER/COUNTER 0 (l' istruzione CLOCK1 colloca i bit 7 e 2 nel registro di funzioni speciali della CPU, IE), l' utente non può usare questo interrupt in un programma in linguaggio assembly, se l' istruzione CLOCK1 é stata eseguita in BASIC. Gli interrupts associati con l' istruzione CLOCK1 fanno sì che i programmi BASIC 52 girino a circa il 99.6% della velocita' normale. Questo significa che l' interrupt manuale, per la caratteristica del Real Time Clock, consuma soltanto lo 0.4% circa del tempo totale della CPU. Questo brevissimo tempo di interrupt globale é da attribuire alla veloce ed efficace gestione degli interrupt da parte della CPU.

Esempio:

```
>10 CLOCK0
>20 TIME=0
>30 CLOCK1
>40 DO
>50 TEMPO=INT(TIME)
>60 UNTIL TEMPO>=30
>70 PRINT"SONO TRASCORSI 30 sec DAL LANCIO DEL PROGRAMMA"
>80 END
```

CONT

Comando: **CONT(cr)**

Descrizione:

Se un programma é fermo dopo che é stato digitato un control-C sulla tastiera oppure se é stata eseguita un' istruzione come STOP, si può riprendere l' esecuzione del programma premendo CONT(cr). Nel periodo che intercorre tra il momento in cui il programma é stato fermato ed il momento in cui riparte l' esecuzione, possono essere visualizzati ed anche modificati i valori delle variabili. Comunque non si può continuare l' esecuzione nel caso sia stato modificato il programma durante lo STOP o dopo che si é incorsi in un errore.

Esempio:

```
>100 A=0
>200 DO
>300 PRINT A
>400 A=A+1
>500 UNTIL 250
>RUN

0
1
2
3
4      - (é stato premuto control-C sulla tastiera)

STOP - IN LINE 300

READY
>PRINT A
5
>A=35

>CONT

35
36
37
38
```

DATA

Istruzione: **DATA**

Modo: Comando interno al programma

Tipo: Trasferimento

Descrizione:

Specifica le espressioni che possono essere recuperate dall' istruzione READ. Se per ogni linea sono usate espressioni multiple, esse devono essere separate da una virgola.

Esempio:

Vedi istruzione RESTORE.

DIM

Istruzione: **DIM**

Modo: Comando interno o esterno al programma

Tipo: Trasferimento

Descrizione:

Il DIM riserva memoria per le matrici. L' area di memorizzazione é prima fissata a zero. Le matrici del BASIC 52 possono avere solo una dimensione e la misura del vettore dimensionato non puo' superare i 254 elementi. Ogni volta che una variabile é stata dimensionata in un programma, essa non può più essere ridimensionata. Un tentativo di ridimensionare il vettore causerà un ARRAY SIZE ERROR (Dimensione del Vettore Errata). Se viene usata una variabile che non é stata dimensionata dall' istruzione DIM, il BASIC assegnerà al vettore un valore di default di 10. Tutti i vettori sono settati al valore zero, quando il comando RUN, il comando NEW o l' istruzione CLEAR sono state eseguite. Il numero di bytes allocati per un vettore e' 6 volte il vettore dimensionato piu' 1. Così il vettore A(100) richiederà 606 bytes di memoria. La dimensione della memoria limita l' estensione del vettore da dimensionare.
Più di una variabile può essere dimensionata da una singola istruzione DIM per esempio:
DIM A(10), B(15), A1(20)

Esempio:

ERRORE DI DEFAULT NEL TENTATIVO DI RIDIMENSIONARE IL VETTORE

```
>10 V(5)=10          -Il BASIC assegna un default di  10 elementi   al vettore V.
>20 DIM V(5)         - Il vettore non può essere ridimensionato
>RUN

ERROR ARRAY SIZE - IN LINE 20

20     DIM V(5)
-------------X
```

DO-UNTIL

Istruzioni: **DO-UNTIL [espr. relazionata]**

Modo: Comando interno al programma

Tipo: Controllo

Descrizione:

L' istruzione DO-UNTIL [espr. rel.] fornisce una struttura di controllo del loop interno al programma BASIC 52. Tutte le istruzioni comprese tra DO-UNTIL [espr. rel.] saranno eseguite fino a che l' espressione relazionale seguente l' istruzione UNTIL é vera. Il ciclo DO-UNTIL può essere anche nidificato.

Esempi:

DO-UNTIL SEMPLICE

```
>10 PRINT"INSERIRENUMERODI FINECONTEGGIO"
>20 INPUT FINE
>30 I=0
>40 DO
>50    PRINT I
>60    I=I+1
>70 UNTIL I=FINE+1
>80 PRINT"FINE CONTEGGIO"
>RUN

INSERIRE  NUMERO DI FINE CONTEGGIO
? 5

0
1
2
3
4
5

FINE CONTEGGIO

READY
>
```

DO-UNTIL NIDIFICATO

```
>10PRINT”SOMMADELLE POSSIBILI COMBINAZIONI DI 2 NUMERI TRA 0 E 5"
>20 A=0: B=0
>30 DO
>40    DO
>50       PRINT A,”+”, B,”=”, A+B
>60        B=B+1
>70     UNTIL B=6
>80     B=0
>90     A=A+1
>100 UNTIL A=6
>110 END
>RUN

SOMMA DELLE POSSIBILI COMBINAZIONI DI 2 NUMERI TRA 0 E 5

0+0=0
0+1=1
0+2=2
.
.
.
.
5+4=9
5+5=10

READY
>
```

DO-WHILE

Istruzioni: **DO-WHILE [espr. relazionata]**

Modo: Comando interno al programma

Tipo: Controllo

Descrizione:

L' istruzione DO-WHILE [espr. rel.] fornisce una struttura di controllo per loop interni al programma. L' operazione di questa istruzione é similare a quella di DO-UNTIL [espr. rel.], eccetto che tutte le istruzioni comprese tra DO e WHILE [espr. rel.] saranno eseguite finché l'espressione relazionale, che segue l' istruzione WHILE, é verificata. Le istruzioni DO-WHILE e DO-UNTIL possono essere nidificate.

Esempi:

DO-WHILE SEMPLICE

```
>10 PRINT"PRIME 10 POTENZE DI 2"
>20 A=1
>30 PRINT A
>40 DO
>50    A=A*2
>60 WHILE A<1024
>70 END
>RUN

PRIME 10 POTENZE DI 2
1
2
4
8
16
32
64
128
256
512
1024

READY
>
```

DO-WHILE - DO-UNTIL NIDIFICATE

```
>10 PRINT”PRODOTTO DELLE POSSIBILI COMBINAZIONI DI 2 NUMERI TRA 0 E 3"
>20 A=0: B=0
>30    DO
>40       DO
>50          PRINT A,”*”, B,”=”, A*B
>60          B=B+1
>70       WHILE B<4
>80       B=0
>90       A=A+1
>100   UNTIL A=B
>110   END
>RUN

PRODOTTO DELLE POSSIBILI COMBINAZIONI DI 2 NUMERI TRA 0 E 3

0*0=0
0*1=0
0*2=0
.
.
3*2=6
3*3=9

READY
>
```

END

Istruzione: **END**

Modo: Comando interno al programma

Tipo: Controllo

Descrizione:

L' istruzione END pone fine all' esecuzione del programma. Il comando per continuare, CONT, non sarà operante se l' istruzione END é stata usata per terminare l' esecuzione e, in questo caso, apparirà sullo schermo la dicitura CAN'T CONTINUE ERROR (Errore di Non Continuità). L' ultima istruzione in un programma BASIC 52 farà terminare, automaticamente, il programma in esecuzione se non é stata usata un' istruzione END.

Esempi:

CONCLUSIONE CON L'ULTIMA ISTRUZIONE

```
>10 I=1
>20 DO
>30 PRINT I
>40 I=I+1
>50 UNTIL I=6
>RUN

1
2
3
4
5

READY
>
```

CONCLUSIONE CON L'ISTRUZIONE END

```
>10 I=1
>20 DO
>30 GOSUB 100
>40 UNTIL I=6
>50 END
>100 PRINT I
>110 I=I+1
>120 RETURN
>RUN

1
2
3
4
5

READY
>
```

FOR-NEXT

Istruzioni: **FOR-TO-{STEP}-NEXT**

Modo: Comandi interni o esterni al programma

Tipo: Controllo

Descrizione:

Le istruzioni FOR-TO-{STEP}-NEXT sono usate per fissare e controllare i cicli.

Esempi:

```
10 FOR A=B TO C STEP D
20 PRINT A
30 NEXT A
```

Se B=0, C=10, e D=2, l' istruzione PRINT che compare alla riga 20 sarà eseguita 6 volte. I valori di"A" che verranno stampati sono 0, 2, 4, 6, 8, 10. "A" rappresenta il nome dell' indice o del contatore del ciclo. Il valore di "B" é il valore di partenza dell' indice, il valore di "C" é il valore limite dell' indice, ed il valore di "D" é l' incremento dell' indice. Se l' istruzione STEP ed il valore "D" sono tralasciati, l' incremento del valore di default é pari ad 1, quindi l' istruzione STEP é facoltativa. L' istruzione NEXT fa sì che il valore di "D" venga sommato all' indice. L' indice é allora comparato al valore limite di "D". Se l' indice é minore o uguale al limite, il controllo é ritrasferito all' istruzione sucessiva a quella di FOR, altrimenti é trasferito all' istruzione successiva a quella di NEXT. É anche possibile tornare indietro, come si può notare nel seguente comando: FOR I=100 TO 1 STEP-1. A differenza di altri BASICS, l' indice non puo' essere omesso dall' istruzione NEXT del BASIC 52 (il NEXT deve sempre essere seguito dalla variabile approppiata).
L' istruzione FOR-TO-{STEP}-NEXT può essere anche nidificata.

```
>10 FOR I=0 TO 6 STEP 2
>20 FOR J=I TO 6
>30 PRINT I,J
>40 NEXT J
>50 NEXT I
>60 END
>RUN

0 0
0 1
0 2
0 3
0 4
0 5
```

```
0 6
2 2
2 3
2 4
2 5
2 6
4 4
4 5
4 6
6 6

READY
>
```

Nel BASIC 52 é possibile eseguire l' istruzione FOR-TO-{STEP}-NEXT come comando esterno al programma. Per l' utente diventa possibile fare cose come la visualizzazione di regioni di memoria direttamente dalla scrittura di un breve programma quale:
FOR I=512 TO 560: PHO. XBY(I),: NEXT I.
Tale istruzione può avere anche altri usi, ma per il momento non verranno trattati.

```
10 FOR J=1 TO 1000: REM RITARDO
20 PRINT J
30 NEXT
```

Tale scrittura é permessa nel BASIC 52. La variabile associata all'istruzione NEXT é sempre assunta per essere la variabile usata nell' ultima istruzione FOR.

GOSUB

Istruzioni: **GOSUB [numero intero]**

Modo: Comando interno al programma

Tipo: Controllo

Descrizione:

L'istruzione GOSUB [num. int.] farà sì che il BASIC 52 trasferisca il controllo del programma direttamente al numero della linea ([num. int.]) che segue l' istruzione GOSUB. In più, l' istruzione GOSUB salva sul Control Stack l' indirizzo della locazione che memorizza l' istruzione seguente il GOSUB stesso, così che un' istruzione RETURN puo' essere compiuta per tornare al controllo del programma.

Esempio:

```
SUBROUTINE SEMPLICE

>10 FOR I=1 TO 10
>20 GOSUB 100
>30 NEXT I
>40 END
>100 PRINT I
>110 RETURN
>RUN

1
2
3
4
5
6
7
8
9
10

READY
>
```

Vedi anche l' istruzione RETURN.

GOTO

Istruzione: **GOTO [numero intero]**

Modo: Comando interno o esterno al programma

Tipo: Controllo

Descrizione:

L' istruzione GOTO [num. int.] farà sì che il BASIC trasferisca direttamente il controllo al numero della linea ([num. int.]) seguente l' istruzione GOTO.

Esempio:

```
10 GOTO 1000
```

Tale istruzione, se esiste la linea 1000, farà effettuare un' esecuzione del programma a partire dalla linea 1000. Nel caso che la linea 1000 non dovesse esistere, sullo schermo comparirà il seguente messaggio: ERROR: INVALID LINE NUMBER
Diversamente dal comando RUN l' istruzione GOTO, se eseguita all' esterno del programma non cancellerà lo spazio di immagazzinamento delle variabili e degli interrupts.
Comunque, se l' istruzione GOTO é stata eseguita come comando esterno dopo avere editato una linea , il BASIC 52 cancellerà lo spazio di memorizzazione delle variabili e degli interrupts. Ciò é necessario perché lo spazio di immagazzinamento delle variabili ed il programma BASIC risiedono nella stessa zona di memoria. Così editando un programma si possono cancellare le variabili stesse.

IDLE

Istruzione: **IDLE**

Modo: Comando interno al programma

Tipo: Controllo

Descrizione:

L' istruzione IDLE forza il dispositivo in un modo interrupt di tipo "wait until". L' esecuzione delle istruzioni e` fermata fino a che ciascuno degli interrupts ONTIME [espres.], [num. int.] o ONEX1 [num. int.] si e` verificata. L' utente deve fare si` che uno o entrambi gli interrupts siano stati abilitati, eseguendo prima un' istruzione IDLE, altrimenti il dispositivo BASIC 52 entrera` in un modo "wait forever" ed in pratica il sistema sara` bloccato.
Quando un ONTIME [espres.], [num. int.] o un ONEX1 [num. int.] é ricevuto fino al IDLE, il dispositivo BASIC 52 eseguirà una routine di interrupt, poi esegue l' istruzione che segue il comando IDLE. Da questo momento, l' esecuzione dell' istruzione IDLE termina, perché é già stata ricevuto un interrupt.
Quando si entra nell' IDLE mode, il dispositivo BASIC 52 utilizza il pin /DMA ACKNOWLEDGE (PORT 1, BIT 6=0) per indicare che l' istruzione IDLE é attiva e non occorrerà attivare il bus esterno. Questo pin é fisicamente il pin 7 sul dispositivo BASIC 52. Quando il dispositivo BASIC 52 esce dall' IDLE mode, tale pin é posto allo stato logico 1 (Non Attivo).
L' utente può inoltre uscire dall' IDLE mode con una routine di interrupt in linguaggio assembly. Questa é realizzabile settando il bit 33 (21H) (che é il bit indirizzabile dalla locazione 36.1 RAM) quando il controllo ritorna dalla routine di interrupt in linguaggio assembly. Se questo bit non é settato dall' utente, il dispositivo BASIC 52 rimmarrà nel modo IDLE fino a quando la routine in linguaggio assembly dell' utente ritorna al BASIC.
Un tentativo di eseguire l' istruzione IDLE in modo diretto produrrà un BAD SYNTAX ERROR.

IF-THEN-ELSE

Istruzioni: **IF-THEN-ELSE**

Modo: Comando interno al programma

Tipo: Controllo

Descrizione:

L' istruzione IF fissa una prova condizionale. La forma generalizzata dell' istruzione IF-THEN-ELSE é la seguente:

[num.int.] IF [espres.rel.] THEN comando valido ELSE com. valido

Esempi:

```
>10 IF B=200 THEN A=0 ELSE A=A+1
```

Riguardo l' esecuzione della linea 10, se B é uguale a 200, allora ad A é assegnato un valore pari a 0. Se B non é uguale a 200, ad A é assegnato un valore pari ad A+1. Se si desidera trasferire il controllo ad un' altra linea, usando l' istruzione IF, si può tralasciare l' istruzione GOTO. Dal seguente esempio si vede come:

```
>20 IF INT(A)<10 THEN GOTO 100 ELSE GOTO 200    Equivale a:

>20 IF INT(A)<10 THEN 100 ELSE 200
```

In più, l' istruzione THEN può essere sostituita da una qualsiasi istruzione del BASIC 52, come é mostrato più sotto:

```
>30 IF A<>10 THEN PRINT A ELSE 10

>30 IF A<>10 PRINT A ELSE 10
```

L' istruzione ELSE può essere tralasciata. Questo fa sì che il controllo passi all' istruzione successiva.

Nel seguente esempio, se A é uguale a 10, verrà eseguita l' istruzione PRINT A, se A non é uguale a 10, verrà eseguita la linea numero 30.

```
>20 IF A=10 THEN PRINT A
>30 PRINT A/2
```

INPUT

Istruzione: **INPUT**

Modo: Comando interno al programma

Tipo: INPUT/OUTPUT

Descrizione:

L' istruzione INPUT permette agli utenti di inserire dei dati dalla tastiera durante l' esecuzione del programma. Una o più variabili possono essere assegnate come data con un' istruzione singola di INPUT, in questo caso le variabili devono essere separate da una virgola.

Esempi:

```
INPUT X1,X2
```

Sul video appare un punto interrogativo(?), che suggerisce all' operatore di introdurre due numeri separati da una virgola. Se l' operatore non introduce abbastanza dati, il BASIC 52 risponde editanto sul video il seguente messaggio: TRY AGAIN.

```
>10 INPUT X1,X2
>20 PRINT X1,X2
>RUN

?10

TRY AGAIN

?10,20

10  20

READY
>
```

L' istruzione INPUT può essere scritta in modo che, un suggerimento descrittivo sia stampato per dare all' utente indicazioni su quale dati inserire. Il messaggio, per essere stampato, deve essere posto tra virgolette dopo l' istruzione INPUT. Se una virgola appare prima della lista di variabili di ingresso, allora il punto interrogativo non viene visualizzato.

```
>10 INPUT"INSERIRE IL NUMERO DI
        CUI FARE IL QUADRATO" X
>20 PRINT"QUADRATO=", X*X
>RUN
INSERIREIL NUMERO DI CUI FARE
IL QUADRATO

?10
QUADRATO=100

READY
>
```

```
>10 PRINT"INSERIRE IL NUMERO
        DI CUI FARE IL QUADRATO", X
>20 PRINT"QUADRATO=", X*X
>RUN
INSERIRE IL NUMERO DI CUI FARE
IL QUADRATO 10
QUADRATO=100

READY
>
```

Anche le stringhe possono essere assegnate con un' istruzione INPUT. Le stringhe sono sempre ultimate con un carriagereturn (CR). Così, se più di una stringa di input é richiesta con unica istruzione di INPUT, il BASIC 52 lo segnalerà all' utente con un punto interrogativo.

```
>10 STRING 110,10
>20 INPUT "NOME: ",$(1)
>30 PRINT "CIAO",$(1)
>RUN
NOME: GIOVANNI
CIAO GIOVANNI

READY
>
```

```
>10 STRING 110,10
>20 INPUT "NOMI:", $(1), $(2)
>30 PRINT "CIAO", $(1), "E", $(2)
>RUN
NOMI: GIOVANNI
?ROBERTA
CIAO GIOVANNI E ROBERTA

READY
>
```

Inoltre, le stringhe e le variabili possono essere assegnate con un' unica istruzione INPUT

```
>10 STRING 100, 10
>20 INPUT "NOME(CR),ETA`,PESO,ALTEZZA:",$(1),ETA`,PESO,ALTEZZA
>30 PRINT "CIAO",$(1),"TU HAI", ETA`,"ANNI; PESI",PESO,"KG E
        MISURI",ALTEZZA,"cm."
>RUN
NOME(CR), ETÀ, PESO, ALTEZZA: GIOVANNI
?18
?70
?170
CIAO GIOVANNI TU HAI 18 ANNI; PESI 70 KG E MISURI 170 cm.

READY
>
```

LD@

Istruzioni: **LD@ [espressione]**

Modo: Comando interno o esterno al programma

Tipo: Input/Output

Descrizione:

L' istruzione LD@ [espres.] lascia che l' utente riprenda i numeri a virgola mobile che erano stati salvati con l' istruzione ST@ [espres.]. L' espressione [espres.] che segue l' istruzione LD@ specifica l' indirizzo della locazione dove il numero é immagazzinato e, dopo l' esecuzione dell' istruzione LD@ [espres.],il numero é prelevato dallo Stack. Da ricordare che ogni numero a virgola mobile richiede 6 bytes di immagazzinamento. Inoltre é da notare che l' espressione delle istruzioni ST@ [espres.] e LD@ [espres.] punta il byte più significativo del numero immagazzinato. Quindi per esempio, ST@ (00E05H) salva il numero nelle locazioni 00E05H, 00E04H, 00E03H, 00E02H, 00E01H e 00E00H.

Esempio:

Si salva e si riprende in memoria un vettore di 50 elementi a partire dalla locazione 00E00H.

```
>10 REM SALVA IL VETTORE IN MEMORIA
>20 FOR I=0 TO 50
>30 REM NUMERO DELLO STACK
>40 PUSH X(I)
>50 REM MEMORIZZAZIONE DEL NUMERO NEI 6 BYTE NECESSARI
>60 ST@ 00E00+6*I
>70 NEXT I
>80 REM RIPRENDE IL VETTORE IN MEMORIA
>90 FOR I=0 TO 50
>100 REM PRELEVA IL NUMERO NEI 6 BYTE NECESSARI
>110 LD@ 00E00+6*I
>120 REM PRELEVA NUMERO DALLO STACK
>130 POP X(I)
>140 NEXT I
>150 END
```

LET

Istruzione: **LET**

Modo: Comando interno o esterno al programma

Tipo: Trasferimento

Descrizione:

L' istruzione LET é usata per assegnare ad una variabile il valore di un' espressione. La forma generalizzata di LET é la seguente:

LET [variabile]= [espressione]

Esempi:

LET X = 100+SQR(A) oppure

LET X = X+1

Da notare che il segno = usato nell' istruzione LET non é l' operatore di uguaglianza, ma piuttosto un operatore di sostituzione, per cui l' istruzione sarebbe letta :
X é sostituita da X più 1. La parola LET é facoltativa.

LET X = 2 é lo stesso di X = 2

Quando l'istruzione LET é trascurata, viene chiamato un LET Implicito. Questo manuale usa l' istruzione LET in forma implicita.
L' istruzione LET é anche usata per assegnare delle variabili di stringa:

LET $(1)= "QUESTA E` UNA SEQUENZA DI CARATTERI" oppure

LET $(2)=$(1)

Prima che le stringhe possano essere assegnate l' istruzione [espres.] deve essere eseguita, altrimenti si occorrerà in un MEMORY ALLOCATION ERROR.
Anche i valori di funzioni speciali possono essere assegnati dall' istruzione LET:

LET IE = 82H oppure

LET XBY(2000H)=00H oppure

LET DBY(25)=XBY(100H)

LIST

Comando: **LIST(cr)**

Descrizione:

Il comando LIST(cr) visualizza il programma sul video. Da notare che il comando LIST "formatta" il programma in modo che possa essere letto facilmente. Gli spazi sono inseriti dopo il numero di linea, prima e dopo ogni istruzione. Questa caratteristica e` presente per aiutare il debugging dei programmi BASIC 52. Il listato di un programma puo` terminare ogni volta che viene premuto un control-C sulla tastiera.
In BASIC 52 sono possibili due variazioni del comando LIST:
1) LIST [numero intero] (cr)
2) LIST [numero intero]-[numero intero] (cr)
La prima variazione consente al programma di essere visualizzato dal numero di linea designato (numero intero) fino alla fine del programma stesso. La seconda variazione al comando LIST, permette al programma di poter essere visualizzato sul video dal numero di linea indicato con il numero intero contenuto nella prima parentesi, al numero di linea indicato con il numero intero contenuto nella seconda parentesi. É importante notare che i due numeri di linea devono essere separati da un trattino (-).

Esempio:

```
>LIST
100 PRINT"CICLO DEL PROGRAMMA"
200 A=0
300 DO
400 PRINT A
500 A=A+3
600 WHILE A<100
READY

>LIST 400
400 PRINT A
500 A=A+3
600 WHILE A<100
READY

>LIST 200-500
200 A=0
300 DO
400 PRINT A
500 A=A+3
READY
```

LIST#

Comando: **LIST#(cr)**

Descrizione:

Il comando LIST#(cr) permette la stampa del listato su di una stampante. Il baud rate di tale dispositivo deve essere inizializzato da un' istruzione BAUD[espress.]. Tutte le considerazioni fatte per il comando LIST, valgono anche per il comando LIST#. Il comando LIST#(cr) consente all' utente di avere una "hard copy" del programma. L' output per la stampante e` posta sul pin P1.7 del dispositivo BASIC 52.

LIST@

Comando: **LIST@(cr)**

Descrizione:

Il comando LIST@ fa la stessa cosa del comando LIST, eccetto che l' output é diretta al driver definito dall' utente. Questo comando fa sì che l' utente riponga una routine di output in linguaggio assembly alla locazione di memoria esterna 403CH. Per abilitare la routine del driver @, l' utente deve settare il Bit 27H (39D) nella memoria interna del dispositivo BASIC 52. Il Bit 27H (39D) é il Bit 7 della locazione di memoria interna 24H (36D). Questo Bit può essere settato da un' istruzione BASIC come DBY(24H)=DBY(24H).OR.80H oppure utilizzando una routine in linguaggio assembly fornita dall' utente. Se l' utente richiama la routine del driver @ e questo bit non é settato, l' output sarà diretta lo stesso al driver. L' unica ragione per la quale questo pin deve essere settato per abilitare il driver @ é che si aggiunge un certo grado di protezione dalla digitatura accidentale di un LIST@, nel caso non esista una routine in linguaggio assembly. La filosofia qui presente é che se l' utente setta tale bit, é in grado di utilizzare il driver o altro.
Quando il BASIC 52 chiama la routine dell' utente per il drive di uscita posta alla locazione 403CH, il byte di uscita é nell' accumulatore ed in R5 del banco dei registri 0 (RB0).
L' utente può modificare l' accumulatore (A) ed il data pointer (DPTR) con una routine di output in linguaggio assembly, ma non può modificare alcun registro di RB0. Questo é stato progettato per facilitare l' utente nell' operazione di implementing relativa al port di uscita seriale o parallelo del driver senza dover utilizzare le istruzioni POP e PUSH.

NEW

Comando: **NEW(cr)**

Descrizione:

Quando un comando NEW(cr) entra, il BASIC 52 cancella il programma contenuto in quel momento nella memoria RAM. Inoltre, tutte le variabili sono poste a zero, tutte le stringhe e tutti gli interrupts richiamati dal BASIC sono cancellati. Il Real Time Clock, l' allocazione per stringhe, ed il valore dello stack pointer interno (locazione 3EH) non hanno più effetto. In generale, il comando NEW(cr) é semplicemente utilizzato per cancellare un programma e tutte le sue variabili.

NULL

Comando: **NULL [intero](cr)**

Descrizione:

Il comando NULL[intero](cr) determina quanti caratteri nulli (00H) il BASIC 52 farà uscire dopo un carriage return. Il comando NULL era molto importante qualche tempo fa, quando le stampanti erano puramente meccaniche e più comunemente utilizzate rispetto ai dispositivi di I/O. Le più moderne stampanti contengono molti tipi di buffer RAM che eliminano virtualmente la necessità di far uscire caratteri nulli dopo un carriage return.

Nota:

Il conteggio di quanti NULL utilizzati dal BASIC 52 viene effettuato nella RAM interna alla locazione 21 (15H). Il valore di NULL può essere modificato dinamicamente in un programma, usando un' istruzione DBY(21)=[espress.].
L' [espress.] può avere un qualsiasi valore compreso tra 0 e 255 (0FFH) compresi.

ONERR

Istruzione: **ONERR[numero intero]**

Modo: Comando interno al programma

Tipo: Controllo

Descrizione:

L' istruzione ONERR[num. int.] lascia al programmatore la gestione di errori aritmetici, che potrebbero capitare durante l' esecuzione del programma. Solo gli errori aritmetici di Overflow e di Underflow, di divisione per Zero e di Bad Argument possono essere "accalappiati" dall' istruzione ONERR[num. int.]. Se un errore aritmetico si verifica dopo che l' istruzione ONERR e` stata eseguita, l' interprete del BASIC 52 passa il controllo alla linea che segue l' istruzione ONERR [numero int.].
Il programmatore può manipolare le condizioni di errore nella maniera più idonea alla particolare applicazione. Tipicamente, l' istruzione ONERR[num. int.] potrebbe essere vista come una facile strada per manipolare gli errori che occorrono quando l' utente fornisce dei dati inappropiati per un' istruzione INPUT.
Con l' istruzione ONERR[num. int.], il programmatore ha l' opzione di determinare in che tipo di errore incorre. Questo é possibile esaminando la locazione di memoria esterna 257 (101H), dopo che la condizione di errore é stata esaminata.
I codici di errore sono i seguenti:

ERROR CODE = 10 - DIVISIONE PER ZERO

ERROR CODE = 20 - ERRORE ARITMETICO DI OVERFLOW

ERROR CODE = 30 - ERRORE ARITMETICO DI UNDERFLOW

ERROR CODE = 40 - ARGOMENTO ERRATO

Questa locazione può essere esaminata usando l' istruzione XBY(257).

ONEX1

Istruzione: **ONEX1 [numero intero]**

Modo: Comando interno al programma

Tipo: Controllo

Descrizione:

L' istruzione ONEX1[num. int.] lascia all' utente la possibilità di manipolare gli interrupts presenti sul pin INT1 della CPU con un programma BASIC. Il numero della linea seguente l' istruzione ONEX1 informa l' interprete del BASIC 52 a che linea passare il controllo quando si incorre in un interrupts. Fondamentalmente, l' istruzione ONEX1 "forza" un GOSUB al numero di linea indicato dal numero intero che segue l' istruzione ONEX1, quando il pin INT1 della CPU é posto basso. Il programmatore deve eseguire un' istruzione RETI per uscire dalla routine dell' interrupt ONEX1. Se questo non é fatto tutti i futuri interrupts sul pin INT1 non saranno acquisiti e quindi ignorati fino a che un RET1 viene eseguito.
L' istruzione ONEX1 setta i bits 7 e 3 del registro dell' interrupt enable IE della CPU. Prima che un interrupt possa essere eseguito, l' interprete del BASIC 52 deve completare l' esecuzione dell' istruzione che é attualmente in corso. Questo perché gli interrupts latenti possono variare da qualche microsecondo a 10 millisecondi. L' interrupt ONTIME[espres.], [num. int.] ha priorità sull' interrupt ONEX1. Così, l' interrupt ONTIME può interrompere la routine dell' interrupt ONEX1.

Esempio:

```
>  .
>100 ONEX1 1000
>  .
>990 END
>1000 PRINT "SI E` VERIFICATA UNA RICHIESTA DI INTERRUPT DALL' ESTERNO"
>1010 RETI
```

ON - GOSUB

Istruzione: **ON [espres.] GOSUB [num. int.0], [num. int.1],...[num. int.n]**

Modo: Comando interno al programma

Tipo: Controllo

Descrizione:

Il valore dell' espressione seguente l' istruzione ON é il numero della linea del listato alla quale sarà trasferito il controllo.

Esempio:

Se [espres.] é uguale a 0, il controllo viene trasferito alla linea [num. int.0].
Se [espres.] é uguale a 1, il controllo viene trasferito alla linea [num. int.1].
e così via.
Se [espres.] é minore di 0 verrà dato un BAD ARGUMENT ERROR.
Se [espres.] é maggiore del numero della linea del listato seguente l' istruzione GOSUB, verrà dato un BAD SYNTAX ERROR. Tutte le osservazioni fatte a proposito di GOSUB valgono anche per l' istruzione ON. L' istruzione ON fornisce delle opzioni "conditional branching" interne alla struttura di un programma BASIC 52

```
>10 PRINT"SELEZIONARE MODO DI FUNZIONAMENTO (1-4)"
>20 INPUT SCELTA
>30 ON SCELTA -1 GOSUB 100, 200, 300, 400
>40 END
>100 <GESTIONE MODO 1>
> .
>190 RETURN
>200 <GESTIONE MODO 2>
> .
>290 RETURN
>300 <GESTIONE MODO 3>
> .
>390 RETURN
>400 <GESTIONE MODO 4>
> .
>410 RETURN
```

ON - GOTO

Istruzione: **ON [espres.] GOTO [num. int.0], [num. int.1],...[num. int.n]**

Modo: Comando interno al programma

Tipo: Controllo

Descrizione:

Il valore dell' espressione seguente l' istruzione ON é il numero della linea del listato alla quale sarà trasferito il controllo.

Esempio:

Se [espres.] é uguale a 0, il controllo viene trasferito alla linea [num. int.0].
Se [espres.] é uguale a 1, il controllo viene trasferito alla linea [num. int.1].
e così via.
Se [espres.] é minore di 0 verrà dato un BAD ARGUMENT ERROR.
Se [espres.] é maggiore del numero della linea del listato seguente l' istruzione GOTO, verrà dato un BAD SYNTAX ERROR. Tutte le osservazioni fatte a proposito di GOTO valgono anche per l' istruzione ON. L' istruzione ON fornisce delle opzioni "conditional branching" interne alla struttura di un programma BASIC 52

```
>10 PRINT"SELEZIONARE MODO DI FUNZIONAMENTO (1-4)"
>20 INPUT SCELTA
>30 ON SCELTA -1 GOTO 100, 200, 300, 400
>40 END
>100 <GESTIONE MODO 1>
> .
>190 GOTO 40
>200 <GESTIONE MODO 2>
> .
>290 GOTO 40
>300 <GESTIONE MODO 3>
> .
>390 GOTO 40
>400 <GESTIONE MODO 4>
> .
>410 GOTO 40
```

ONTIME

Istruzione: **ONTIME [espressione], [numero intero]**

Modo: Comando interno al programma

Tipo: Controllo

Descrizione:

Siccome il BASIC 52 esegue una linea in un intervallo di tempo di alcuni millisecondi ed i timer/counters sulla CPU operano in un intervallo di tempo di microsecondi, c'é una incompabilità tra i timer/counters della CPU e il BASIC 52. Per aiutare a risolvere questa situazione é stata ideata l' istruzione ONTIME [espres.], [num. int.]. Questo interrupt ONTIME é generato tutte le volte che, l' Operatore Speciale di Funzione TIME é uguale o maggiore dell' espressione seguente l' istruzione ONTIME. Attualmente, solo la porzione intera del Tempo é comparata alla porzione intera dell' espressione. Gli interrupts forzano un GOSUB al numero della linea ([num. int.]) che segue l' espressione ([espres.]) nell' istruzione ONTIME. A partire dall' istruzione ONTIME, che utilizza l' Operatore di Funzione Speciale, Time, l' istruzione CLOCK1 deve essere eseguita in ordine per far funzionare l' ONTIME stesso. Se il CLOCK1 non é eseguito, l' Operatore Speciale di Funzione, Time, non incrementerà e l' istruzione ONTIME non entrerà mai in funzione. Siccome l' istruzione ONTIME genera un' interrupt quando il Time é maggiore o uguale all' espressione seguente l' istruzione ONTIME, gli interrupts periodici possono essere generati eseguendo l' istruzione ONTIME anche nella routine di interrupt.

Esempi:

```
>10 PRINT"INCREMENTO PERIODICO DI VARIABILE CON PERIODO DI 5 SEC"
>20 I=0: TIME=0: CLOCK1: ONTIME 5,100
>30 END
>100 PRINT I, "TEMPO=", TIME, "secondi"
>110 I=I+1
>120 ONTIME TIME+5,100: RETI
>RUN
0  TEMPO= 5.045 secondi
1  TEMPO= 10.045 secondi
2  TEMPO= 15.045 secondi
.
READY
>
```

Ci si può meravigliare perché il TIME, che era stato stampato, era 45 millisecondi più grande del tempo che l' interrupt aveva supposto per essere generata. Questo perché il terminale usato nell' esempio girava a 4800 BAUD e utilizzava circa 45 millisecondi per stampare il messaggio TEMPO= " ".
Se il programmatore non vuole questo ritardo, una variabile può essere assegnata dall' Operatore

di Funzione Speciale, Time, all' inizio della routine di interrupt.

```
>10 PRINT"INCREMENTO PERIODICO DI VARIABILE CON PERIODO DI 5 SEC"
>20 I=0: TIME=0: CLOCK1: ONTIME 5,100
>30 END
>100 X=TIME
>110 PRINT I, "TEMPO=", X, "secondi"
>120 I=I+1
>130 ONTIME TIME+5,100: RETI
>RUN

0  TEMPO= 5 secondi
1  TEMPO= 10 secondi
2  TEMPO= 15 secondi
.  .

READY
>
```

Come per l' istruzione ONEX1, anche per uscire dalla routine dell' istruzione ONTIME deve essere editata l' istruzione RETI. Se si omette tale operazione, tutti i futuri interrupts verranno tralasciati.
L' interrupt ONTIME ha priorità sull' interrupt ONEX1. Questo significa che gli interrupt ONTIME possono interrompere la routine dell' interrupt ONEX1. Questa priorità é stata stabilita perché, le funzioni relative il tempo in applicazioni di controllo sono viste come routine critiche.
Se l' utente non vuole che la routine ONEX1 sia interrotta da un' interrupt ONTIME, si può eseguire, all' inizio di una routine ONEX1, un' istruzione CLOCK0 o CLEARI. Gli interrupt possono essere riabilitati prima della fine di una routine ONEX1. Un' interrupt ONEX1 non può interrompere una routine ONTIME.
L' istruzione ONTIME del BASIC 52 é l' unica comune a molti BASICs. Questa potente istruzione elimina la necessità, per l' utente, di testare periodicamente il valore dell' operatore Time dal principio alla fine del programma BASIC.

PGM

Istruzione: **PGM**

Modo: Comando interno o esterno al programma

Tipo: Input/Output

Descrizione:

L' istruzione PGM dà all' utente la possibilità di programmare una EPROM o EEPROM mentre sta eseguendo un programma BASIC.
L' ISTRUZIONE PGM RICHIEDE CHE L' UTENTE FISSI LE LOCAZIONI DI MEMORIA INTERNA 18H (24D), 19H (25D), 1AH (26D), 1BH (27D), 1EH (30D) e 1GH (31D). Da notare che queste locazioni di memoria interna sono normalmente riservate all' utente!!
L' utente deve inizializzare queste locazioni di memoria interna nel seguente modo:

Esempi:

LOCAZIONI	CONTENUTI
1BH: 19H (27D: 25D)	L' INDIRIZZO DELL' INFORMAZIONE DI SORGENTE CHE STA PER ESSERE PROGRAMMATA ALL' INTERNO DELL' EPROM, LA LOCAZIONE 19H É IL BYTE BASSO E LA LOCAZIONE 1BH É IL BYTE ALTO
1AH: 18H (26D: 24D)	L' INDIRIZZO - 1 DELLA/E LOCAZIONE/I DELL' EPROM CHE DEVONO ESSERE PROGRAMMATE, LA LOCAZIONE 18H É IL BYTE BASSO E LA LOCAZIONE 1AH É IL BYTE ALTO.
1FH: 1EH (31D: 30D)	IL NUMERO DI BYTE CHE L' UTENTE VUOLE PROGRAMMARE. LA LOCAZIONE 1EH É IL BYTE BASSO E LA LOCAZIONE 1FH É IL BYTE ALTO

L' utente deve anche inizializzare l' ampiezza desiderata per l' impulso di programmazione della EPROM e immagazzinare tale valore nelle locazioni di memoria 40H (64D) (byte alto) e 41H (65D) (byte basso). Il ricaricamento per un impulso di programmazione della EPROM pari a 70 millisecondi é calcolato come segue:

```
10 REM R=VALORE DI CARICAMENTO, A=AMPIEZZA IN SECONDI (70 ms)
20 A= .07
30 R=65536-A*XTAL/12
40 DBY(40H)=R/256: REM SCRITTURA BYTE ALTO
50 DBY(41H)=R.AND.0FFH: REM SCRITTURA BYTE BASSO
```

L' utente deve inoltre settare o cancellare il bit 38.3 (26.3H) per selezionare l' algoritmo di programmazione della EPROM intelligente. Il bit é settato per selezionare la programmazione

intelligente e cancellato per selezionare il normale algoritmo di 70 ms.
Persettare il bit, si esegue un'istruzione: DBY(38)=DBY(38).OR.8H
Per cancellareilbitsiesegueun' istruzione: DBY(38)=DBY(38).AND.0F7H
Quando é eseguita come comando interno al programma, l' istruzione PGM non fa visualizzare un errore, nel caso che si incorra in uno sbaglio di programmazione. Il controllo del programma ritornerà poi all' utente proprio come se l' EPROM fosse programmata correttamente. L' utente deve poi esaminare le locazioni 1EH e 1FH. Se i contenuti di entrambe le locazioni 1EH e 1FH sono uguali a zero, l' EPROM sarà programmata correttamente. Se però i contenuti non sono uguali a zero, si é incorsi in un ERRORE durante il processo di programmazione. L' utente può allora esaminare le locazioni 1AH:18H per determinare in quale locazione dell' EPROM si é verificato l' errore di programmazione. Il seguente programma é un esempio di programmatore universale di EPROM/EEPROM per il BASIC 52, può programmare un blocco di RAM all' interno di una EPROM o EEPROM che é indirizzata a 8000H.

```
10 PRINT "PROGRAMMATORE UNIVERSALE DI PROM": PRINT "TIPO DI PROM"
20 PRINT : PRINT "1=EEPROM": PRINT "2=EPROM INTELLIGENTE"
30 PRINT "3=NORMALE (50 MS) EPROM":PRINT:INPUT"TIPO(1,2,3)-",T
40 ON (T-1) GOSUB 340, 350, 360
50 REM questa fissa l' intelligent programming se occorre
60 IF W=.001 THEN DBY(26)=DBY(26).OR.8 ELSE DBY(26)=DBY(26).AND.0F7H
70 REM calcolare ampiezza impulso e salvare
80 PUSH (65536-(W*XTAL/12)): GOSUB 380
90 POP G1: DBY(40H)=G1: POP G1: DBY(41H)=G1: PRINT
100 INPUT "INDIRIZZO INIZIALE DEI DATI",S:IF S<512.OR.S>0FFFFH THEN 100
110 PRINT : INPUT "INDIRIZZO FINALE DEI DATI", E
120 IF E<S.OR.E>0FFFFH THEN 110
130 PRINT:INPUT"INDIRIZZO DELLLA PROM",P: IF P<8000H.OR.P>0FFFFH THEN 100
140 REM calcolare il numero di bytes del programma
150 PUSH (E-S)+1: GOSUB 380: POP G1: DBY(31)=G1: POP G1: DBY(30)=G1
160 REM fissare indirizzo EPROM
170 PUSH (P-1): GOSUB 380: POP G1: DBY(26)=G1: POP G1: DBY(24)=G1
180 REM fissare indirizzo di sorgente
190 PUSH S: GOSUB 380: POP G1: DBY(27)=G1: POP G1: DBY(25)=G1
200 PRINT:PRINT"PREMERE UN CR PER INIZIARE LA PROGRAMMAZIONE"
210 REM aspettare per un 'cr' poi programmare l' EPROM
220 X=GET: IF X<0DH THEN 220
230 REM programmare l' EPROM
240 PGM
250 REM vedere se ci sono errori
260 IF (DBY(30).OR.DBY(31))=0 THEN PRINT"PROGRAMMAZIONE COMPLETA": END
270 PRINT: PRINT "ERRORE": PRINT
280 REM queste routine calcolano l' indirizzo della sorgente
290 REM la locazione EPROM che é stata sbagliata
300 S1=DBY(25)+256*DBY(27): S1=S1-1: DBY(24)+256*DBY(26)
310 PHO."IL VALORE", XBY(D1),: PH1. "LETTO ALLA LOCAZIONE", S1: PRINT
320 PHO. "SULLA EPROM SI LEGGE", XBY(D1),: PH1. "ALLA LOCAZIONE", D1: END
330 REM queste subroutines fissano l' ampiezza dell' impulso
340 W=.0005: RETURN
350 W=.001: RETURN
360 W=.05: RETURN
370 REM Routine che prende un dato a 16 bit dallo Stack e fornisce i 2 byte H e L
380 POP G1: PUSH (G1.AND.0FFH): PUSH (INT(G1/256)): RETURN
```

PH0. PH1.

Istruzioni: **PH0.**, **PH1.**, **PH0.#**, **PH1.#**

Modo: Comandi interni o esterni al programma

Tipo: Input/Output

Descrizione:

Le istruzioni PH0. e PH1. svolgono le stesse funzioni dell' istruzione PRINT, eccetto che i valori sono stampati all' esterno in un formato esadecimale. L' istruzione PH0. sopprime i 2 zeri incontrati, se il numero stampato é minore di 255 (0FFH). L' istruzione PH1. stampa sempre, in uscita, 4 cifre esadecimali. Il carattere "H" é sempre stampato dopo i numeri, quando PH0 o PH1 sono usati per direzionare un' uscita. I valori stampati sono sempre approssimati agli interi. Se il numero stampato non é interno al range dell' intero valido (cioé compreso tra 0 e 65535 (0FFFFH)), il BASIC 52 avrà un normale default del modo di stampa. Se questo non accade, "H" sarà stampato dopo il valore. A partire dagli interi, che possono entrare ciascuno in forma decimale o esadecimale, le istruzioni PRINT, PH0., e PH1. possono essere usate per compiere delle conversioni da decimale ad esadecimale e da esadecimale a decimale. Tutti i commenti fatti per l' istruzione PRINT, si possono trasferire anche alle istruzioni PH0. e PH1. .
PH0.# e PH1.# svolgono le stesse funzioni di PH0. e PH1., eccetto che per queste due istruzioni le uscite sono dirette dal dispositivo di list (Stampante).

Esempi:

```
>PH0. 4*4
0FH

>PH1. 4*4
000FH

>PRINT 50H
80

>PH0. 200
C8H

>PH0. 500
1F4H

>PH1. 500
01F4H

>P. 1F4H
500

>RAGGIO=5
>PHO. PI*RAGGIO*RAGGIO
4EH
```

POP

Istruzione: **POP [variabile]**

Modo: Comando interno o esterno al programma

Tipo: Trasferimento

Descrizione:

La parte superiore dello Stack é assegnata alla variabile seguente l' istruzione POP e lo Stack perde tale parte (incremento di 6 dello Stack Pointer). I valori possono essere posti sullo stack da ciascuna istruzione PUSH o da una CALLS in linguaggio assembly.
Più di una variabile può essere fatta uscire dallo Stack con un' unica istruzione POP. Le variabili sono semplicemente seguite da una virgola (POP [var], [var],...[var]).
Se un' istruzione POP é stata eseguita ed un numero non é contenuto sullo Stack, si incorrerà in un ERROR A-STACK.
Le istruzioni PUSH e POP sono uniche nel BASIC 52. Queste potenti istruzioni possono essere usate per trattare i problemi delle variabili Globali, spesso usate in programmi BASIC. Questo problema si presenta perché, nel BASIC, il programma principale e tutte le subroutines usate dal programma principale operano sugli stessi nomi di variabile (VARIABILE GLOBALE). Non é sempre conveniente usare le stesse variabili in una subroutine e nel programma principale, in quanto si possono vedere programmi che riassegnano un numero di variabili prima che sia eseguita un' istruzione GOSUB. Se l' utente destina molti nomi di variabili proprio per le subroutine ed introduce delle variabili sullo stack, come si può vedere nell' esempio precedente, si eviteranno tutti i problemi inerenti l' uso di variabili Globali nel BASIC 52.

Esempi:

Vedere gli esempi dell' istruzione PUSH

PRINT

Istruzione: **PRINT** o **P.** o **?**

Modo: Comando interno o esterno al programma

Tipo: Input/Output

Descrizione:

L' istruzione PRINT ordina al BASIC 52 di mandare le uscite sullo schermo. I valori di espressioni, stringhe, valori alfabetici, variabili o test di stringa possono essere visualizzati. Alcuni valori possono essere uniti nel listato print ma devono essere separati tra loro con una virgola. Se il listato é terminato con una virgola, il carriage return/line feed verrà soppresso. P. é una forma abbreviata di PRINT. Nella versione 1.1 "?" é una forma abbreviata di PRINT.

Esempi:

```
>PRINT 10*10, SQR(100)        >PRINT "BASIC"        >PRINT 150, 1E4, 1E-4

100  10                       BASIC                      150  10000  0.0001
```

I valori sono stampati uno di seguito all' altro con due spazi di intervallo. Un' istruzione PRINT priva di argomenti causa una sequenza di carrige return/ line feed che vengono mandati sullo schermo.

Funzioni Speciali dell' Istruzione PRINT.

- TAB

La funzione **TAB([espres.])** é usata nell' istruzione PRINT per fare sì che i dati vengano stampati nell' esatta locazione sull' apparato di uscita. TAB([espres.]) dice al BASIC 52 da quale posizione inizia a stampare il valore sucessivo del listato print. Se la testa di stampa o il cursore é sopra o più spostato dalla specifica posizione del TAB, il BASIC 52 ignorerà tale funzione.

Esempio:

```
>PRINT TAB(10), "1", TAB(10), "231", TAB (10), "3"

          1          231          3

<10 spazi>  <10 spazi>      <10 spazi>
```

- SPC

La funzione **SPC([espres.])** é usata nell' istruzione PRINT per fare sì che il BASIC 52 metta in uscita un numero di spazi pari all' argomento dell' SPC.

Esempio:

```
>PRINT X, SPC(10),Y
```

Può essere usato per porre 10 spazi addizionali tra X e Y oltre i due che normalmente devono essere stampati.

- CR

La funzione **CR** é interessante ed unica per il BASIC 52. Quando CR é usata in un' istruzione PRINT, viene forzato un carriage return, ma non un line feed. Questo può essere usato per creare una linea su di un dispositivo CRT che é ripetutamente aggiornato.

Esempio:

```
>10 FOR I=1 TO 1000
>20 PRINT I, CR,
>30 NEXT I
```

Farà sì che l' uscita rimanga solo su una linea. Nessun line feed é mandato sulla console video.

- USING (caratteri speciali)

La funzione **USING** é usata per chiedere al BASIC 52 quale codice usare per visualizzare i valori che sono stampati. Il BASIC 52 fornisce il formato desiderato dopo che é stata eseguita l'istruzione USING. Così, tutte le uscite seguenti un' istruzione USING saranno del formato richiamato dall' ultima istruzione USING eseguita. L' istruzione USING non serve che sia eseguita all' interno di un' istruzione PRINT, a meno che il programmatore voglia cambiare il formato. U. é una forma abbreviata di USING.
Le opzioni per USING sono le seguenti:

USING (Fx)
Questa opzione forza il BASIC 52 a porre in uscita tutti i numeri usando il formato a virgola mobile. Il valore di x determina quante cifre significative saranno stampate. Se x é uguale a 0 il BASIC 52 non porrà in uscita nessun zero incontrato, così il numero di cifre varierà a seconda del numero. Il BASIC 52 porrà, sempre, in uscita le 3 cifre meno
significative anche se x é 1 o 2. Il massimo valore di x é 8.
Esempio:

```
>10 PRINT USING(F3), 1, 1/2, 1/3
>20 PRINT USING(F4), 1, 1/2, 1/3
>30 PRINT USING(F6), 1, 1/2, 1/3
>40 FOR I=100 TO 400 STEP 100
```

```
>50 PRINT I
>60 NEXT I
>RUN
1. 00 E 0  5. 00 E -1  3.33 E -1
1. 000 E 0  5. 000 E -1  3.333 E -1
1. 00000 E 0  5. 00000 E -1  3.33333 E -1
1. 00000 E+2
2. 00000 E+2
3. 00000 E+2
4. 00000 E+2
READY
>
```

USING (#.#)

Questa opzione forza il BASIC 52 a porre in uscita tutti i numeri usando un formato intero e/o frazionale. Il numero "#" che precede il punto decimale rappresenta il numero intero delle cifre significative, che saranno stampati nella frazione. Il punto decimale puo` essere tralasciato, in questo caso verra` stampato solo l' intero. USING puo` essere abbreviato con U. .
USING (###.###), USING (######) e USING (######.##) sono tutte valide per il BASIC 52. Il numero massimo di caratteri "#" e` 8. Se il BASIC 52 non puo` porre in uscita il valore nel formato desiderato (di solito perche` il valore e` troppo grande), sullo schermo compare un punto interrogativo (?), dopo di che il BASIC stesso porra` in uscita il numero nel FREE FORMAT (Formato Libero) descritto sotto.
Esempio:

```
>10    FOR I=50 TO 2000 STEP 250
>20    IF I>1000THENPRINT USING(##.##),I ELSE
       PRINT USING (###.##),I
>30    NEXT I
>RUN
  50.00
  300.00
  550.00
  800.00
  ?1050
   1300
   1550
   1800
READY
>
```

I formati USING(Fx) e USING(#.#) allineano sempre i punti decimali quando si stampa un numero. Con questa caratteristica vengono visualizzate le colonne dei numeri per facilitarne la lettura.

USING (0)

Questo argomento lascia determinare al BASIC 52 il formato da usare. Il metodo é semplice, se il numero é compreso tra + o - 99999999 e + o - 1,il BASIC visualizzerà gli interie le frazioni. Se é fuori da questo range, il BASIC userà il formato USING(F0). Mano a mano che si incontrano degli zeri, questi vengono sempre soppressi. Dopo il reset, il BASIC 52 é posto in un formato USING(0).

PRINT#

Istruzione: **PRINT#** o **P.#** o **?#**

Modo: Comandi interni o esterni al programma

Tipo: Input/Output

Descrizione:

L' istruzione PRINT#, P.# e ?# (solo nella versione 1.1) fa la stessa cosa dell' istruzione PRINT, P e ? (solo nella versione 1.1) eccetto che l' uscita é mandata alla stampante.
Il Baud rate per tale dispositivo deve essere inizializzata dall' istruzione BAUD[espres.] prima che sia usata l' istruzione PRINT#, P.# o ?#. Tutte le considerazioni fatte per l' istruzione PRINT, P. o ? valgono anche per l' istruzione PRINT#, P.# o ?#. P.# e ?# (solo per la versione 1.1) sono forme abbreviate di PRINT#.

Esempio:

```
>10 BAUD 2400
>20 PRINT#"STAMPA DEI PRIMI 100 NUMERI INTERI"
>30 FOR I=1 TO 100
>40 PRINT# I
>60 END
>RUN
```

Se opportunamente collegata e settata a 2400 BAUD, la stampante visualizzerà la stringa:
STAMPA DEI PRIMI 100 NUMERI INTERI
seguita dall' elenco dei numeri interi compresi tra 1 e 100.

PRINT@

Istruzioni: **PRINT@**, **PH0.@**, **PH1.@**

Modo: Comandi interni o esterni al programma

Tipo: Input/Output

Descrizione:

Le istruzioni PRINT@ (P.@ o ?@), PH0.@, e PH1.@ svolgono rispettivamente le stesse funzioni di PRINT (P. o ?), PH0., e PH1, eccetto che l' uscita é diretta dal driver d' uscita definito dall' utente. Queste istruzioni fanno sì che l' utente abbia posto una routine di uscita in linguaggio assembly in una locazione di memoria esterna (403CH). Per abilitare la routine di driver @, l' utente deve settare il BIT 27H (39D) nella memoria interna del dispositivo BASIC 52 che é il bit 7 della locazione di memoria interna 27H (39D). Questo bit può essere settato dall' istruzione BASIC DBY(27H)=DBY(27H).OR.80H o dall' utente mediante una routine in linguaggio assembly.
Se l' utente richiama la routine del driver @ e il bit non é settato , l' uscita sarà diretta al driver della console. L' unica ragione per cui il bit deve essere settato per abilitare il driver @ é che egli aggiunge un certo grado di protezione dalla digitatura accidentale di LIST@, quando non esiste la routine in linguaggio assembly. La filosofia qui presente é che se l' utente setta il bit, andrà a utilizzare il driver o altro.
Quando il BASIC 52 chiama la routine del driver di uscita definito dall' utente e posta alla locazione 403CH, il byte di uscita é nell' accumulatore e nel registro 5 (R5) del banco di registro 0 (RB0). L' utente può modificare l' accumulatore (A) e il data pointer (DPTR) della routine di uscita in linguaggio assembly, ma non può modificare niente del registro in RB0. In questo modo risulta semplice per l' utente completare un driver con uscita seriale o parallela senza dover fare un PUSH o un POP.

PROG

Comandi: **PROG(cr) e FPROG(cr)**

Descrizione:

Il comando PROG programma la EPROM presente con il programma selezionato. Tale programma può risiedere in altra RAM o EPROM. Dopo che é stato premuto il comando PROG(cr), viene visualizzato sul video il numero che il file occupa all' interno della EPROM.
FPROG(cr) fa esattamente le stesse cose di PROG(cr) eccetto che per effettuare l' operazione di programmazione impiega un algoritmo programmabile veloce "INTELLIGENT" della INTEL.

Esempio:

```
>LIST
10 A=1
20 A=A*3
30 PRINT A
40 UNTIL A=30

READY
>PROG
  8

READY
>ROM 8

READY
>LIST
10 A=1
20 A=A*3
30 PRINT A
40 UNTIL A=30

READY
>
```

In questo esempio, il programma posto in EPROM é l' ottavo programma immagazzinato.

PROG1

Comandi: PROG1(cr) e FPROG1(cr)

Descrizione:

Normalmente, dopo che il dispositivo BASIC 52 é stato alimentato, l' utente deve digitare uno spazio per inizializzare il port seriale della CPU. Il BASIC 52 contiene un comando di PROG1, la cui funzione é di programmare la EPROM con le relative informazioni riguardanti il Baud Rate. In questo modo tutte le volte che é presente un segnale di power-up sul dispositivo BASIC 52, come ad esempio il Reset, il chip legge queste informazioni e va ad inizializzare il port seriale con un determinato valore di Baud Rate. Il messaggio sarà spedito alla console immediatamente dopo che il dispositivo BASIC 52 ha completato la sequenza di Reset. Il carattere “spazio” non occorre che venga premuto. Sicuramente, se il Baud Rate della console é variato, deve essere programmata una nuova EPROM per poter rendere compatibile il BASIC 52 con la nuova console.
FPROG1(cr) fa esattamente le stesse cose di PROG1(cr) eccetto che per effettuare l' operazione di programmazione impiega un algoritmo programmabile veloce "INTELLIGENT" della INTEL.

PROG2

Comandi: **PROG2(cr) e FPROG2(cr)**

Descrizione:

Il comando PROG2 effettua le stesse operazioni del comando PROG1, ma invece di assumere e di entrare in modo comando, il dispositivo BASIC 52 inizia ad eseguire immediatamente il primo programma immagazzinato nella EPROM.
FPROG2(cr) fa esattamente le stesse cose di PROG2(cr) eccetto che per effettuare l' operazione di programmazione impiega un algoritmo programmabile veloce "INTELLIGENT" della INTEL.

Nota:

Usando il comando PROG2 é possibile eseguire un programma partendo da una condizione di Reset e senza mai connettere il chip BASIC 52 alla console. Concludendo, se si salva l' informazione PROG2 é l' equivalente a digitare un ROM1, cioé un comando di esecuzione. Questo é l' ideale per l' applicazione in controlli, dove non é sempre possibile avere un terminale a portata di mano. In più, questa caratteristica permette all' utente di scrivere delle sequenze di inizializzazioni speciali in BASIC oppure anche in linguaggio Assembly e di generare un messaggio utente specifico al tipo di applicazione.

PROG3

Comandi: PROG3(cr) e FPROG3(cr)

Descrizione:

Il comando PROG3 funziona allo stesso modo del comando PROG1, eccetto che PROG3 salva anche il valore di controllo del sistema MTOP, ogni volta che viene richiamato. Durante un Reset o un power-up, il BASIC 52 cancella solo la memoria dei dati esterna che ha valore superiore a quello indicato da MTOP, il quale é stato salvato dal comando PROG3. Questo permette all' utente di proteggere regioni di memoria durante una cancellazione che avviene con un Reset o con un condizione di power-up. Nell' uso tipico, il comando PROG3 permette all' utente di salvare molte informazioni particolari se qualche tipo di batteria di back-up o di memoria non volatile non sono in grado di assicurarne il mantenimento durante un Reset o un power-up.
FPROG3(cr) fa esattamente le stesse cose di PROG3(cr) eccetto che per effettuare l' operazione di programmazione impiega un algoritmo programmabile veloce "INTELLIGENT" della INTEL.

PROG4

Comandi: **PROG4(cr) e FPROG4(cr)**

Descrizione:

Il comando PROG4 é una combinazione dei comandi PROG2 e PROG3. PROG4 salva le stesse informazioni di PROG3, ma ha la caratteristica di eseguire il primo programma contenuto nella EPROM dopo un Reset o una condizione di power-up.
FPROG4(cr) fa esattamente le stesse cose di PROG4(cr) eccetto che per effettuare l' operazione di programmazione impiega un algoritmo programmabile veloce "INTELLIGENT" della INTEL.

PROG5

Comandi: PROG5(cr) e FPROG5(cr)

Descrizione:

Il comando PROG5 salva sia le informazioni riguardanti il baud rate che quelle riguardanti MTOP, proprio come il comando PROG3. Comunque, durante un Reset o un condizione di power-up, il dispositivo BASIC 52 esamina la locazione della memoria dei dati esterna 5FH (95 decimale). Se l' utente ha posto in questa locazione il valore 0A5H (165 decimale), il dispositivo BASIC 52 **non cancellerà la memoria esterna** durante un Reset o una condizione di power-up. Questo permette all' utente di salvare programmi nella memoria esterna, provvedendo ad impiegare alcuni tipi di batterie di back-up. Normalmente, quando si usa il comando PROG5 per effettuare il Reset od una condizione di power-up, il dispositivo BASIC 52 entrerà in modo comando dopo il Reset o il power-up. Comunque, se l' utente vuole eseguire il programma contenuto nella memoria esterna, occorre che il carattere 34H (52 decimale) sia posto nella locazione di memoria esterna 5EH (94 decimale). Ponendo 34H alla locazione 5EH, il BASIC 52 entrerà nel "RUN TRAP MODE".
FPROG5(cr) fa esattamente le stesse cose di PROG5(cr) eccetto che per effettuare l' operazione di programmazione impiega un algoritmo programmabile veloce "INTELLIGENT" della INTEL.

PROG6

Comandi: **PROG6(cr) e FPROG6(cr)**

Descrizione:

Effettua la stessa operazione di PROG5, ed in più chiama la locazione di memoria esterna 4039H, nel caso vi sia un Reset o un power-up. L' utente deve riporre una routine di inizializzazione in linguaggio assembly alla locazione di memoria dei dati 4039H, altrimenti viene cancellato il tutto non appena si presenta un Reset. Quando l' utente richiama la routine di Reset in linguaggio assembly, egli può utilizzare le seguenti 3 opzioni:

- Opzione 1 per PROG6
Se il bit di CARRY é resettato (CARRY=0), non appena si ritorna dalla routine di Reset dell' utente, il BASIC 52 entrerà in una routine che determina automaticamente il baud rate. L' utente a questo punto deve premere il carattere "spazio" (20H) sul terminale per completare la routine di Reset e produrre un messaggio di Reset sul terminale.

- Opzione 2 per PROG6
Se il bit di CARRY é settato (CARRY=1) ed il Bit 0 dell' Accumulatore é resettato (ACC.0=0), il BASIC 52 produrrà un messaggio standard non appena ritorna dalla routine di Reset fornita dall' utente. Il baud rate sarà quello che é stato salvato con l' utilizzo dell' opzione PROG6.

- Opzione 3 per PROG6
Se il bit di Carry é settato (CARRY=1) ed il bit 0 dell' Accumulatore é settato (ACC.0=1), il BASIC 52 eseguirà il primo programma immagazzinato dall' utente in EPROM (l' indirizzo di partenza del programma é 8010H) non appena si ritorna dalla routine di Reset fornita dall' utente.

FPROG6(cr) fa esattamente le stesse cose di PROG6(cr) eccetto che per effettuare l' operazione di programmazione impiega un algoritmo programmabile veloce "INTELLIGENT" della INTEL.

PUSH

Istruzione: **PUSH [espressione]**

Modo: Comando interno o esterno al programma

Tipo: Trasferimento

Descrizione:

L' espressione aritmetica o le espressioni, seguenti l' istruzione PUSH, sono valutate e sucessivamente poste sequenzialmentenello Stack del BASIC 52. L' istruzione PUSH, insieme all' istruzione POP, provvede a semplificare il trasferimento di parametri a routine in linguaggio assembly. In più, le istruzioni PUSH e POP possono essere usate per trasferire dei parametri a subroutine BASIC o a variabili "SWAP". L' ultimo valore introdotto sullo Stack sarà il primo valore messo fuori dallo Stack stesso (Lo Stack é organizzato con la tecnica LIFO)
Più di una espressione può essere introdotta nello Stack con un' unica istruzione PUSH. Le espressioni sono seguite semplicementedaunavirgola:
PUSH[espres.], [espres.],...[espres.]
L' ultimo valore introdotto sullo Stack sarà la prima espressione [espres.] incontrata nell' istruzione PUSH.

Esempi:

VARIABILI DI SWAPPING

```
>10 X1=100
>20 X2=200
>30 PRINT X1, X2
>40 PUSH X1, X2
>50 POP X1, X2
>60 PRINT X1, X2
>RUN

100  200
200  100

READY
>
```

ROUTINE DI PASSING

```
>10    PRINT"INSERIRE IL VALORE DEL RAGGIO DELLA CIRCONFERENZA"
>20    INPUT RAGGIO
>30    PUSH RAGGIO
```

```
>40    GOSUB 100
>50    POP AREA, CIRC
>60    PRINT"AREA=", AREA, "CIRCONFERENZA=", CIRC
>70    END
>100   POP RAGGIO
>110   PUSH RAGGIO*RAGGIO*PI
>120   PUSH 2*RAGGIO*PI
>130   RETURN
>RUN

INSERIRE IL VALORE DEL RAGGIO DELLA CIRCONFERENZA
?5
AREA= 78.539816  CIRCONFERENZA= 31.415927

READY
>
```

PWM

Istruzione: **PWM [espres.], [espres.], [espres.]**

Modo: Comando interno o esterno al programma

Tipo: Input/Output

Descrizione:

PWM sta per Pulse Width Modulation (modulazione ampiezza impulso). Per generare, un' utente definisce una sequenza di impulsi sul P1.2 (bit 2 del PORT 1 I/O) del dispositivo BASIC 52. La prima espressione che segue l' istruzione PWM é il numero di cicli di clock per i quali l' impulso rimarrà alto. Un ciclo di clock equivale a 12/XTAL, che sono 1.085 microsecondi a 11.0592 MHz. La seconda espressione é il numero di cicli di clock per i quali l' impulso rimarrà basso, e la terza espressione indica il numero totale di cicli che l' utente vuole mettere in uscita. Tutte le espressioni seguenti l' istruzione PWM devono essere degli interi validi (cioé compresi tra 0 e 65535 (0FFFFH) inclusi). In più, il valore minimo delle prime due espressioni dell' istruzione PWM é di 25.
L' istruzione PWM può essere utilizzata per creare risposte udibili in un sistema. In più, per divertimento, il programmatore può suonare della musica, usando appunto l' istruzione PWM.

Esempi:

```
>PWM 100, 100, 1000
```

A 11.0592 MHz si possono generare 1000 cicli di un' onda quadra che ha un periodo di 217 microsecondi (4608 Hz) sul P1.2.

Determinare il massimo ed il minimo di 5 numeri

```
>10 PRINT"INSERIRE 5 NUMERI"
>20 FOR I=1 TO 5
>30 INPUT A(I)
>40 PWM 100, 100, 500
>50 NEXT I
>60 MAX= A(1): MIN= A(1)
>70 FOR I=2 TO 5
>80 IF A(I)>MAX THEN MAX= A(I)
>90 IF A(I)<MIN THEN MIN= A(I)
>100 NEXT I
>110 PWM 100, 100, 500
>120 PRINT" MASSIMO=", MAX, "MINIMO=", MIN
>130 END
>RUN
```

```
INSERIRE 5 NUMERI
?1  <BEEP>
?8  <BEEP>
?3  <BEEP>
?7  <BEEP>
?20 <BEEP>

  <BEEP>

MASSIMO= 20  MINIMO= 1

READY
>
```

In questo esempio l' istruzione PWM viene utilizzata per creare un feed-back sonoro per le operazioni di ingresso-uscita su console.

RAM

Comando: **RAM(cr)**

Descrizione:

Questo comando indica all' interprete del BASIC 52 che il programma attuale é prelevato da RAM (il programma corrente é quello visualizzato con il comando LIST e può essere eseguito quando si digita RUN). L' indirizzo di inizio del programma in RAM é 512 (200H).
Quando si inserisce il comando RAM(cr), il BASIC 52 seleziona il programma corrente dalla memoria RAM. Questo é considerato il normale modo di operazione.

READ

Istruzione: **READ**

Modo: Comando interno al programma

Tipo: Trasferimento

Descrizione:

Rintraccia le espressioni che sono state specificate nell' istruzione DATA ed assegna il valore dell' espressione alla variabile dell' istruzione READ. L' istruzione READ deve essere seguita sempre da una o più variabili. Se più di una variabile segue l' istruzione READ , esse devono essere separate da una virgola.

Esempio:

Vedi istruzione RESTORE.

REM

Istruzione: **REM**

Modo: Comando interno o esterno al programma

Tipo: Commento - Non compie operazioni

Descrizione:

REM é un' abbreviazione di REMark (appunto). Non svolge alcuna operazione, ma permette all' utente di aggiungere commenti al programma. I commenti normalmente servono a rendere un programma più comprensibile. Se un REM appare su di una linea, l' intera linea é considerata come un commento, così un' istruzione REM non può mai essere seguita da altre istruzioni distinte dai due punti (:). É comunque possibile che tale istruzione venga posta dopo i due punti (:), in modo da permettere al programmatore di porre un commento ad ogni linea.

Esempi:

```
>10 REM LETTURA DEI COEFFICIENTI DI UNA EQUAZIONE DI II GRADO CON
>15 REM DELTA>0
>20 INPUT A, B, C
>30 REM CALCOLO DELLE RADICI
>40 X1=+(-B+SQR(B*B-4*A*C))/(2*A)
>50 X2=-(-B-SQR(B*B-4*A*C))/(2*A)
>60 REM STAMPA DELLE RADICI
>70 PRINT X1, X2
```

```
>10 INPUT A, B, C: REM   LETTURA DEI COEFFICIENTI DI UNA EQUAZIONE
                          DI II GRADO CON DELTA>0
>20 X1=(-B-SQR(B*B-4*A*C))/(2*A): REM CALCOLO PRIMA RADICE
>30 X2=(-B+SQR(B*B-4*A*C))/(2*A): REM CALCOLO SECONDA RADICE
>40 PRINT X1, X2: REM STAMPA DELLE RADICI
```

Il seguente esempio non può funzionare perché l' intera linea viene interpretata come un REMark, così l' istruzione PRINT non é eseguita:

```
>10 REM STAMPA IL NUMERO: PRINT X
```

Nota: La ragione per cui l' istruzione REM é stata resa eseguibile come comando interno o esterno al programma, nella versione 1.1 del BASIC 52, é che se l' utente sta sviluppando qualche tipo di programma UPLOAD/DOWNLOAD con un computer, questi lascia inserito le istruzioni REM, senza i numeri di linea nel testo e senza il loro caricamento nel dispositivo del BASIC 52. Questo aiuta a risparmiare memoria.

RESTORE

Istruzione: **RESTORE**

Modo: Comando interno al programma

Tipo: Trasferimento

Descrizione:

Resetta il puntatore interno di lettura, il quale torna all' inizio del DATA così che puo' essere letto ancora.

Esempio:

```
>10 FOR I=1 TO 10
>20 READ DAT
>30 PRINT "DATO N.ro",I,"=",DAT
>40 IF I=5 THEN RESTORE
>50 NEXT I
>60 DATA 10, 20, 15*2, 80/2, 5*5*2
>70 END
>RUN
DATO N.ro 1 = 10
DATO N.ro 2 = 20
DATO N.ro 3 = 30
DATO N.ro 4 = 40
DATO N.ro 5 = 50
DATO N.ro 6 = 10
DATO N.ro 7 = 20
DATO N.ro 8 = 30
DATO N.ro 9 = 40
DATO N.ro 10 = 50
READY
>
```

Tutte le volte che l' istruzione READ incontra l' espressione successiva all' istruzione DATA, tale espressione viene valutata e assegnata alla variabile che segue l' istruzione READ. Le istruzioni DATA possono essere poste ovunque all' interno di un programma, esse non saranno eseguibili e neanche causeranno errore. Le istruzioni DATA sono considerate per essere legate insieme ed apparire come un' unica grande istruzione di DATA. Ogni volta che tutti i DATA sono stati letti ed un' altra istruzione READ viene eseguita, il programma termina e compare sullo schermo il seguente messaggio:
ERROR: NO DATA - IN LINE XX

RETI

Istruzione: **RETI**

Modo: Comando interno al programma

Tipo: Controllo

Descrizione:

L' istruzione RETI é utilizzata per uscire dagli interrupts che sono manipolati da un programma BASIC 52. Più specificatamente dagli interrupts ONTIME e ONEX1. L' istruzione RETI fa la stessa cosa dell' istruzione RETURN, eccetto che il RETI cancella anche un software interrupts flags, così gli interrupts successivi possono ancora essere riconosciuti.
Se l' utente non riesce ad eseguire l' istruzione RETI con la dovuta procedura degli interrupts, tutte i futuri interrupts saranno ignorati.

Esempio:

Vedi istruzioni ONTIME e ONEX1.

RETURN

Istruzioni: **RETURN**

Modo: Comando interno al programma

Tipo: Controllo

Descrizione:

Questa istruzione é usata per ritornare, con il controllo, all' istruzione successiva alla più recente istruzione di GOSUB. La sequenza GOSUB-RETURN può essere "nidificata", questo significa che una subroutine chiamata dall' istruzione GOSUB può chiamare un' altra subroutine con un' altra istruzione GOSUB.

Esempi:

SUBROUTINE NIDIFICATA

```
>10 PRINT"QUADRATI DEI PRIMI 10 NUMERI INTERI"
>20 FOR I=1 TO 10
>30 GOSUB 100
>40 NEXT I
>50 END
>100 GOSUB 150
>110 PRINT I,I2
>120 RETURN
>150 I2=I*I
>160 RETURN
>RUN

QUADRATI DEI PRIMI 10 NUMERI INTERI
1       1
2       4
3       9
4       16
5       25
6       36
7       49
8       64
9       81
10      100

READY
>
```

Il Control Stack permette un' uscita dal ciclo di controllo incompleto. Vediamo come:

```
.
.
50  GOSUB 1000
.
.
1000 FOR I=1 TO 500
1010 IF X=I THEN RETURN
1020 PRINT I*X
1030 NEXT I
```

La versione 1.1 può permettere al programmatore di uscire dalla subroutine anche se il ciclo di FOR-NEXT potrebbe non essere completo, cioé se, come nell' esempio, X non era uguale ad I. Nella versione 1.0 del BASIC 52 si potrebbe produrre un errore di Control Stack, se il ciclo FOR-NEXT non era stato completato prima che l' istruzione RETURN fosse eseguibile.

ROM

Comando: **ROM [interi](cr)**

Descrizione:

Questo comando indica all' interprete del BASIC 52 che il programma attuale é prelevato da EEPROM (il programma corrente é quello visualizzato con il comando LIST e può essere eseguito quando si digita RUN). L' indirizzo di inizio del programma in EEPROM é 32784 (8010H).
Quando entra un comando di ROM [intero](cr), il BASIC 52 seleziona il programma da prelevare dalla EPROM. Se dopo il comando ROM non viene inserito alcun numero intero (ad esempio ROM(cr)), il BASIC 52 utilizza un valore di default pari ad 1, quindi si avrà un comando ROM 1.
Dal momento che i programmi sono immagazzinati sequenzialmente nella EPROM, il numero intero che segue il comando ROM seleziona quale programma l' utente vuole listare o far eseguire. Se si prova a selezionare un programma che non esiste (ad esempio se si digita ROM 5, quando sono soltanto 4 i file contenuti nella EPROM), compare il seguente messaggio errore: ERROR: PROM MODE.
Il BASIC 52 non trasferisce il programma dalla EPROM alla RAM, quando viene selezionato il modo ROM. In questo modo, non si può editare un programma in modo ROM, se ci si prova verrà generato un ERROR: PROM MODE. Il comando che permette di trasferire un programma da EPROM a RAM per poter essere editato é XFER.
Non potendo trasferire, con il comando ROM, un programma alla RAM, é possibile avere simultaneamente differenti programmi sia in ROM che in RAM. L' utente può quindi rientrare ed uscire dai due modi tutte le volte che lo desidera. Un altro vantaggio di non poter trasferire un programma alla RAM, é che tutta la memoria RAM può essere utilizzata per immagazzinare variabili, se il programma é contenuto in EPROM. I valori di controllo del sistema MTOP e FREE sono sempre riferiti alla RAM e non alla EPROM.

RROM

Istruzione: **RROM [intero]**

Modo: Comando interno o esterno al programma

Tipo: Controllo

Descrizione:

RROM significa RUN ROM. Questo fa selezionare un programma nel file EPROM per poi eseguirlo. L' intero successivo all' istruzione RROM seleziona quale programma nel file EPROM sta per essere eseguito. Nel modo di comando RROM 2 é equivalente a ROM 2 e poi RUN. C' é però da dire che RROM [int.] é un' istruzione. Questo significa che se si sta già eseguendo un programma, può allora forzare l' esecuzione di un programma completamente differente, che é situato nel file EPROM. Questo dà all' utente la capacità di "cambiare programmi" velocemente.
Se l' utente esegue un' istruzione RROM [int.] ed inserisce un intero sbagliato (ad esempio se nel file EPROM sono contenuti 6 programmi e l' utente inserisce RROM 8, oppure se non c' é EPROM nel sistema), non sarà generato un errore e il BASIC 52 eseguirà l' istruzione successiva all' istruzione RROM [int.].
Ogni volta che viene eseguita un' istruzione RROM [int.], tutte le variabili e le stringhe sono settate al valore zero, così le variabili e le stringhe non possono passare da un programma ad un altro per mezzo dell' istruzione RROM [int.].In più,tutti gli interrupts richiamati dal BASIC 52 sono cancellati.

Esempio:

```
>10 FOR I=1 TO 6
>20 A(I)=I
>30 NEXT I
>40 RROM 1
>50 FOR I=1 TO 6
>60 PRINT A(I)
>70 NEXT I
>80 END
>RUN
SALUTI A TUTTI QUANTI
0
0
0
0
0
0
READY
>
```

```
>10 REM SALUTI : REM In EPROM.
>20 PRINT"SALUTI A TUTTI QUANTI"
>30 END
```

RUN

Comando: **RUN(cr)**

Descrizione:

Dopo che é stato digitato RUN(cr), tutte le variabili sono settate a zero, tutte gli interrupts richiamati dal BASIC sono cancellati ed inizia l' esecuzione del programma a partire dal primo numero di linea del programma selezionato. Il comando RUN e l' istruzione GOTO sono gli unici modi con i quali l' utente è in grado di porre l' interprete del BASIC 52 all' interno del modo di esecuzione o del modo di comando. L' esecuzione del programma può essere fermata ogni volta che viene premuto sulla tastiera un control-C.
A differenza di molti interpreti BASIC che permettono di eseguire un solo numero di linea (ad esempio RUN 300), il BASIC 52 non permette tale variazione del comando RUN. L' esecuzione inizia sempre con il primo numero di linea. Per ottenere la stessa funzionalità del comando RUN [numero intero], é opportuno usare l' istruzione GOTO [numero intero] in modo diretto. A questo proposito fare riferimento al capitolo relativo l' istruzione GOTO.

Esempio:

```
>100 X=0
>200 DO
>300 PRINT I
>400 X=X+1
>500 WHILE X<6
>RUN

1
2
3
4
5
```

ST@

Istruzioni: **ST@ [espressione]**

Modo: Comando interno o esterno al programma

Tipo: Input/Output

Descrizione:

L' istruzione ST@ [espres.] lascia che l' utente specifichi dove devono essere immagazzinati i numeri a virgola mobile del BASIC 52. L' espressione [espres.] seguente l' istruzione ST@ specifica l' indirizzo al quale il numero deve essere immagazzinato, quando il numero é memorizzato sullo Stack. L' istruzione ST@ [espres.] é designata per essere usata con l' istruzione LD@ [espres.]. Lo scopo principale di queste due istruzioni é di permettere all' utente di salvare, dovunque nella memoria, i numeri a virgola mobile a patto che l' utente impieghi qualche tipo di batteria back-up o schema non-volatile con questa memoria.

Esempio:

Vedi istruzione LD@.

STOP

Istruzione: **STOP**

Modo: Comando interno al programma

Tipo: Controllo

Descrizione:

L' istruzione STOP permette al programmatore di interrompere il programma in un suo punto specifico. Dopo che un programma é stato fermato, le variabili possono essere visualizzate e/o modificate. L' esecuzione del programma può riprendere con il comando CONT (continua). Lo scopo dell' istruzione STOP é di permettere il "debugging" per programmi facili.

Esempio:

```
>10 A=0
>20 DO
>30 A=A+1
>40 B=A*5
>50 PRINT A,B
>60 STOP
>70 UNTIL A=10
>RUN

1  5
STOP - IN LINE 70

READY
>CONT

2  10
3  15
ecc.
```

Da notare che il numero della linea stampato sullo schermo, dopo che l' istruzione STOP é stata eseguita, é il numero della linea che segue l' istruzione di STOP. NON é il numero della linea che contiene l' istruzione STOP.

STRING

Istruzione: **STRING [espres.], [espres.]**

Modo: Comando interno o esterno al programma

Tipo: Controllo

Descrizione:

L' istruzione STRING [espres.], [espres.] riserva memoria per le stringhe. Inizialmente, la memoria non é allocata per stringhe. Se l' utente prova a definire una stringa con un' istruzione, come LET $(1)="SALVE", prima che una parte di memoria sia stata allocata per stringhe, sarà generato un MEMORY ALLOCATION ERROR (errore di allocazione di memoria). La prima espressione nell' istruzione STRING [espres.], [espres.] corrisponde al numero totale di bytes che l' utente vuole riservare per immagazzinare le stringhe. La seconda espressione indica il numero massimo di bytes che sono in ogni stringa. Questi due numeri determinano il numero totale di variabili stringa definibili. Si può pensare che il numero totale di stringhe definite deve essere uguale alla prima espressione che segue l' istruzione STRING [espres.], [espres.] diviso la seconda espressione. Il BASIC 52 richiede un byte addizionale per ogni stringa, più un byte addizionale globale. Questo significa che l' istruzione STRING 100,10 riserva abbastanza memoria per 9 variabili di tipo stringa, disposte da $(0) a $(8) e tutti i 100 bytes allocati sono utilizzati. Da notare che $(0) é una stringa valida nel BASIC 52. Dopo che la memoria é stata allocata per la memorizzazione di stringhe, nessun comando, come NEW, nessuna istruzione, come CLEAR, potranno deallocare questa memoria. Il solo modo per poterlo fare é di eseguire un' istruzione STRING 0,0. L' istruzione STRING 0,0 non allocherà memoria per variabili di tipo stringa.
Ogni volta che l' istruzione STRING [espres.], [espres.] viene eseguita, il BASIC 52 esegue l' equivalente ad un' istruzione CLEAR. Questo é necessario perché le variabili di stringa e le variabili numeriche occupano lo stesso spazio di memoria esterna. Così, dopo che é stata eseguita l' istruzione STRING, tutte le variabili sono cancellate. Per questo motivo l' istruzione STRING [espres.],[espres.] deve comparire nella prima linea che contiene istruzioni per il programma, in modo da evitare dannose ridefinizioni di tutte le variabili dello stesso programma.

UI0

Istruzioni: **UI0** (User Input)

Modo: Comando interno o esterno al programma

Tipo: Controllo

Descrizione:

Assegna la console per tale input tornando al software dei drivers residenti sul dispositivo BASIC 52. UI1 e UI0 possono essere posti ovunque internamente ad un programma. Questo permette al programma BASIC di ricevere gli inputs da diversi dispositivi e con differenti tempi. La funzione di UI0 é controllata dal bit 30 (1EH) della memoria interna del BASIC 52. Il bit 30 é nella locazione di memoria interna 35.6 (23.6H) ovverosia é il sesto bit all' interno della locazione di memoria 35 (23H). Quando il bit 30 é settato (BIT 30=1), sarà chiamata la routine dell' utente. Quando il bit 30 é resettato (BIT 30=0), sarà usata la routine di driver dell' input del BASIC 52. Il programmatore può usare questa informazione per cambiare la selezione del dispositivo di ingresso tramite il linguaggio assembly.

UI1

Istruzioni: **UI1** (User Input)

Modo: Comando interno o esterno al programma

Tipo: Controllo

Descrizione:

Permette all' utente di scrivere sul tipo di console di ingresso che si addice maggiormante alle richieste del momento. Dopo che UI1 é stato eseguito e quando viene richiesto un input da console, il BASIC chiamerà un programma posto nella locazione esterna di memoria (4033H). L' utente deve provvedere a un' istruzione JUMP per una routine di input in linguaggio assembly posta a quella locazione (4033H). L' input ASCII, destinato ad un certo uso da questa routine, é posto nell' accumulatore della CPU e la routine di input dell' utente ritorna al BASIC dopo l' esecuzione di un' istruzione RET in linguaggio assembly.
L' utente non deve modificare alcun registro della CPU in un programma in linguaggio assembly con l' eccezione della memoria e del banco di registro allocati dall' utente.
In più per fornire la routine al driver di input per un' istruzione UI1, l' utente deve anche provvedere ad una routine di Console Status Check (Controllo dello Stato della Console). Questa routine effettua un controllo per vedere se il dispositivo di console ha un carattere pronto per essere letto dal BASIC 52. Il BASIC chiama una locazione di memoria esterna (4036H) per controllare lo stato della console. La routine di stato della console setta il bit di Carry a 1 (C=1), se un carattere é pronto per essere letto dal BASIC e resetta il bit di Carry (C=0) se il carattere non é pronto. Ancora, il contenuto dei registri non deve essere cambiato. Il BASIC 52 usa la routine di Controllo dello Stato della Console per esaminare la tastiera con un carattere di control-C durante esecuzione del programma e durante un programma Listing. Questa routine é ancora usata per compiere l' operazione GET.
La funzione di UI1 é controllata dal bit 30 (1EH) della memoria interna del BASIC 52. Il bit 30 é nella locazione di memoria interna 35.6 (23.6H) ovverosia é il sesto bit all' interno della locazione di memoria 35 (23H). Quando il bit 30 é settato (BIT 30=1), sarà chiamata la routine dell' utente. Quando il bit 30 é resettato (BIT 30=0), sarà usata la routine di driver dell' input del BASIC 52. Il programmatore può usare questa informazione per cambiare la selezione del dispositivo di ingresso tramite il linguaggio assembly.

UO0

Istruzioni: **UO0** (User Output)

Modo: Comando interno o esterno al programma

Tipo: Controllo

Descrizione:

Riassegna la routine di console di uscita al software dei drivers residenti nel dispositivo BASIC 52. UO0 e UO1 possono essere posti ovunque internamente ad un programma. Questo permette al programma BASIC di fare uscire all' esterno caratteri per diversi dispositivi con tempi differenti.
La funzione di UO0 é controllata dal bit 28 (1CH) della memoria interna della CPU. Il bit 28 é nella locazione di memoria interna 35.4 (23.4H), ed é quindi il quarto bit all' interno della locazione di memoria 35 (28H). Quando il bit 28 é settato (BIT 28=1), la routine dell' utente sarà chiamata. Quando il bit 28 é resettato (BIT 28=0), saranno utilizzate i drivers di uscita. Il programmatore può usare questa informazione per cambiare la selezione delle uscite del dispositivo tramite il linguaggio assembly.

UO1

Istruzioni: **UO1** (User Output)

Modo: Comando interno o esterno al programma

Tipo: Controllo

Descrizione:

Permette all' utente di scrivere sulla console di uscita che si addice maggiormente all' uso del momento. Dopo che UO1 é stata eseguita il BASIC chiamerà un programma posto nella locazione di memoria esterna 4030H, quando viene richiesta una console di uscita. L' utente deve provvedere ad una istruzione di JUMP per una routine di output in linguaggio assembly posta a quella locazione. Il BASIC pone i caratteri uscita nel Registro 5 (R5) del Registro di Banco 0 (RB0). L' utente ritorna al BASIC eseguendo un' istruzione RET in linguaggio assembly. L' utente non può modificare alcun registro della CPU, incluso l' accumulatore, durante la procedura di uscita voluta dall' utente stesso, fatta eccezione per la memoria e per il banco di registro allocato dall' utente. UO1 dà la libertà all' utente di scrivere routine di uscita per il BASIC 52. La funzione di UO1 é controllata dal bit 28 (1CH) della memoria interna della CPU. Il bit 28 é nella locazione di memoria interna 35.4 (23.4H), ed é quindi il quarto bit all' interno della locazione di memoria 35 (28H). Quando il bit 28 é settato (BIT 28=1), la routine dell' utente sarà chiamata. Quando il bit 28 é resettato (BIT 28=0), saranno utilizzate i drivers di uscita. Il programmatore può usare questa informazione per cambiare la selezione delle uscite del dispositivo tramite il linguaggio assembly.

XFER

Comando: **XFER(cr)**

Descrizione:

Il comando XFER (transfer) trasferisce il programma selezionato nella EPROM alla RAM, selezionando il modo RAM. Se XFER viene digitato quando il BASIC 52 si trova nel modo RAM, il programma immagazzinato nella RAM viene ritrasferito nella RAM stessa ed il modo di selezione rimane lo stesso. Con questa operazione non accade niente, eccetto che la CPU impiega un tempo di pochi millisecondi per fare uno spostamento inutile. Dopo che é stato eseguito il comando XFER, l' utente può editare il programma nello stesso modo di ogni altro programma RAM.

OPERATORI ARITMETICI E LOGICI

Il BASIC 52 contiene una serie di operatori aritmetici e logici. Tali operatori sono divisi in 2 gruppi, doppi operatori od operatori bivalenti ed operatori singoli o monovalenti. La forma generalizzata per tutti gli operatori doppi é la seguente:

[espress.] OP [espress.], dove **OP** é uno dei seguenti operatori:

Operatore di addizione: +

Esempio:

```
PRINT 9+2
11
```

Operatore di divisione: /

Esempio:

```
PRINT 450/3
150
```

Operatore esponenziale: **

Tale operatore eleva la prima espressione alla potenza indicata dalla seconda espressione. La potenza a cui ogni numero può essere elevato é al massimo 255. Il simbolo **** é stato scelto invece del simbolo della “freccia in alto” perché quest ultimo può apparire con significati diversi a seconda del terminale usato.

Esempio:

```
PRINT 4**2
16
```

Operatore di moltiplicazione: *

Esempio:

```
PRINT 5*6
30
```

Operatore di sottrazione: -

Esempio:

```
PRINT 20-13
7
```

Operatore Logico AND: .AND.

Esempio:

```
PRINT 6.AND.3
2
```

Operatore Logico OR: .OR.

Esempio:

```
PRINT 7.OR.8
15
```

Operatore Logico OR esclusivo: .XOR.

Esempio:

```
PRINT 4.XOR.5
2
```

Note sugli Operatori Logici .AND., .OR. e .XOR.

Questi operatori compiono una funzione logica BIT-WISE sugli interi validi. Una caratteristica importante é che gli argomenti per questi operatori devono essere compresi tra 0 e 65535 (0FFFFH) compresi. Se non lo sono, il BASIC 52 genera un BAD ARGUMENT ERROR. Tutti i non interi non vengono approssimati, ma bensì troncati. La ragione per la quale é stata adottata la scrittura .OP. per indicare le funzioni logiche, é che il BASIC 52 elimina tutti gli spazi quando opera in una linea utente ed inserisce spazi prima e dopo l' istruzione quando é listato un programma utente. Il BASIC 52 non inserisce spazi prima e dopo gli operatori. Se l' utente digita in una linea: 10 A = 10 * 10, essa viene listata come 10 A=10*10, infatti tutti gli spazi inseriti dall' utente vengono eliminati. La scrittura .OP. é stata scelta per gli operatori logici, perché nel caso l' utente digiti:
10 B= A AND B, tale scrittura può essere listata come 10 B=AANDB
creando confusione, così i punti indicano la presenza di una istruzione logica e, prendendo l' esempio precedente, la linea verrebbe listatanel seguente modo: 10 B=A.AND.B

FUNZIONI MATEMATICHE

ABS([espress.])

Ritorna al valore assoluto dell' espressione:
Esempi:

```
PRINT ABS(12)          PRINT ABS(-12)
12                     12
```

NOT([espress.])

Effettua il complemento a uno dei 16 bit dell' espressione.
L' espressione deve essere un intero valido compreso tra 0 e 65535 (0FFFFH) inclusi. In caso contrario verrà troncata e non approssimata.
Esempi:

```
PRINT NOT(65535)       PRINT NOT(237)
0                      65298
```

INT([espress.])

Preleva la porzione intera dell' espressione.
Esempi:

```
PRINT INT(5.4)         PRINT(321.490)
5                      321
```

SGN([espress.])

Assegna il valore +1 se l' argomento é maggiore di zero, zero se l' argomento é zero, e -1 nel caso l' argomento sia minore di zero.
Esempi:

```
PRINT SGN(37)    PRINT SGN(0)    PRINT SGN(-21)
1                0               -1
```

SQR([espress.])

Calcola la radice quadra dell' argomento. L' argomento non deve mai essere minore di zero. Il risultato sarà compreso di un +/- un valore di 5 sul digit meno significativo.
Esempi:

```
PRINT SQR(4)     PRINT SQR(321)     PRINT SQR(10000)
2                17.916473          100
```

RND

Assegna un numero a caso compreso tra 0 ed 1 inclusi.
L' operatore RND utilizza un codice binario a 16 bit e genera 65536 numeri a caso prima di ripetere la sequenza. I numeri generati sono compresi tra 0/65535 e 65535/65535 inclusi. A differenza di altri BASICs, l' operatore RND del BASIC 52 non richiede un argomento o altro. Infatti, se viene posto un argomento dopo l' operatore RND, si genera un BAD SYNTAX ERROR.
Esempi:

```
PRINT RND
  .6230869
```

PI

PI non é un vero e proprio operatore, ma contiene costanti. Nel BASIC 52, PI contiene 3.1415926. I matematici affermano che in PI attualmente é racchiuso 3.141529653, così per approssimazione il numero contenuto in PI potrebbe essere 3.1415297. La ragione per la quale il BASIC 52 usa un 6 invece di un 7 per l' ultima cifra,é che si può incorrere in errori negli operatori SIN, COS e TAN nel caso venga utilizzato il numero 7 invece del 6.
Questo é il motivo per il quale il numero PI/2 risulta essere necessario per calcoli di tale genere ed é quindi preferibile, per la precisione, avere l' equazione PI/2+PI/2=PI. Questo non può essere fatto se l' ultima cifra di PI é dispari, così che l' ultima cifra viene approssimata a 6 invece di 7, in modo che i calcoli risultino più accurati.

FUNZIONI LOGARITMICHE

LOG([espress.])

Calcola il logaritmo naturale dell' argomento. L' argomento deve essere maggiore di 0. Il risultato del calcolo é visualizzato con 7 cifre significative.
Esempi:

```
PRINT LOG(23)
3.135494
```

```
PRINT LOG(EXP(5))
5
```

EXP([espress.])

La funzione effettua l' elevamento del numero "e" (2.7182818) per la potenza indicata dall' argomento.
Esempi:

```
PRINT EXP(2)
7.3890559
```

```
PRINT EXP(LOG(7))
6.9999994
```

FUNZIONI TRIGONOMETRICHE

SIN([espress.])
Calcola il Seno dell' argomento. L' argomento é espresso in radianti. I risultati sono visualizzati con 7 cifre significative. Inoltre l' argomento deve essere compreso tra +/- 200000.
Esempi:

```
PRINT SIN(PI/15)
.2079116

PRINT SIN(-1)
-.841471
```

COS([espress.])
Questo operatore calcola il Coseno dell' argomento. L' argomento é espresso in radianti e deve essere compreso tra +/- 200000. Il risultato viene espresso con 7 cifre significative.
Esempi:

```
PRINT COS(PI/15)
.9781477

PRINT COS(-1)
.5403023
```

TAN([espress.])
Effettua il calcolo della Tangente dell' argomento. Quest ultimo viene espresso in radianti e deve essere compreso tra +/- 200000.
Esempi:

```
PRINT TAN(2*PI)
0

PRINT TAN(1)
1.5574078
```

ATN([espress.])
Calcola l' Arcotangente dell' argomento in radianti. Il risultato viene espresso con 7 cifre significative. Il calcolo di ATN viene effettuato se l' argomento é compreso tra -PI/2 (3.1415926/2) e PI/2.
Esempi:

```
PRINT ATN(PI/2)
1.0038848

PRINT ATN(0)
0
```

Note sulle funzioni trigonometriche

Gli operatori SIN, COS e TAN usano una serie di Taylor per calcolare la funzione. Tali operatori come prima operazione riducono l' argomento ad un valore compreso tra 0 e PI/2. Questa riduzione viene realizzata dalla seguente equazione:
Argomento Ridotto = (arg. utente/PI - INT(arg. utente/PI)) * PI
L' Argomento Ridotto, come si può notare dalla precedente equazione, sarà compreso tra 0 e PI. Inoltre é anche utilizzato per controllare se é maggiore di PI/2. Se lo é, viene sottratto a PI per ottenere il valore finale. Sebbene questo metodo di riduzione dell' angolo fornisca un semplice ed economico mezzo per generare un argomento appropriato per la serie di Taylor, esiste un problema di precisione associato all' utilizzo di questa tecnica. Il problema di precisione si verifica quando l' argomento introdotto dall' utente é grande (maggiore di 1000). Questo é il motivo per il quale le cifre significative, nella porzione decimale (frazione) dell' Argomento Ridotto, vengono perdute nell' espressione (arg. utente/PI - INT(arg. utente/PI)). La regola generale é quella di cercare di prendere l' argomento per le funzioni trigonometriche il più piccolo possibile.

FUNZIONI SU STRINGHE

ASC()

L' operatore ASC() permette di conoscere il valore intero del carattere ASCII contenuto tra parentesi. Esempi:

```
PRINT (C)
67
```

67 é la rappresentazione decimale del carattere ASCII "C". Inoltre uno specifico carattere contenuto all' interno di una stringa ASCII può essere convertito con l' operatore ASC().

```
>100 STRING 110,30
>200 $(1)="GIANNI HA 17 ANNI"
>300 PRINT $(1)
>400 PRINT ASC($(1),6)
>RUN
GIANNI HA 17 ANNI
  73
```

Quando l' operatore ASC() é usato nel modo mostrato sopra, $([espress.]) indica che attualmente vi é un accesso nella stringa e che l' espressione dopo la virgola individua un determinato carattere nella stringa stessa. Nell' esempio precedente, il sesto carattere ASCII della stringa era individuato nella lettera "I", la cui rappresentazione in carattere decimale é 73.

```
>100 $(1)="PAROLA"
>200 FOR I=1 TO 6
>300 PRINT ASC($(1),I),
>400 NEXT X
>RUN
80  65  82  79  76  65
```

I caratteri stampati nell' esempio precedente sono i valori decimali che rappresentano i caratteri ASCII P, A, R, O, L ed A. L' operatore ASC() può inoltre essere usato per cambiare singoli caratteri in una stringa ben definita.

```
>100 $(1)="VIAGGIATORE"
>200 PRINT $(1)
>300 ASC($(1),3)=73
>400 PRINT $(1)
>500 ASC($(1),7)=ASC($(1),11)
>600 PRINT $(1)
>RUN
VIAGGIATORE
VIIGGIATORE
VIIGGIETORE
```

In generale, l' operatore ASC() permette al programmatore di manipolare singoli caratteri in una stringa. Vediamo ora un programma che determina l' uguaglianza tra due stringhe.

```
>100 REM  LA PAROLA DA INDOVINARE É CHIAVISTELLO
>200 $(1)="CHIAVISTELLO"
>300 INPUT"INSERISCI LA PAROLA - ", $(2)
>400 FOR X=1 TO 12
>500 IF ASC($(1),X)=ASC($(2),X) THEN 600 ELSE 900
>600 NEXT X
>700 PRINT"COMPLIMENTI HAI INDOVINATO LA PAROLA"
>800 END
>900 PRINT"MI DISPIACE HAI SBAGLIATO, RIPROVA"
>1000 GOTO 200
RUN
INSERISCI LA PAROLA - TRENO
MI DISPIACE HAI SBAGLIATO, RIPROVA
INSERISCI LA PAROLA - CHIAVISTELLO
COMPLIMENTI HAI INDOVINATO LA PAROLA
```

CHR()

L' operatore CHR() é il contrario dell' operatore ASC(). Egli converte un' espressione numerica in un carattere ASCII.
Esempi:

```
PRINT CHR(67)
C
```

Come per ASC(), anche per l' operatore CHR() si può scegliere un carattere specifico in una stringa ASCII.

```
>100 $(1)="BARRA DI FERRO"
>200 FOR D=1 TO 14
>300 PRINT CHR($(1),D),
>400 NEXT D
>500 PRINT
>600 FOR D=14 TO 1 STEP -1
>700 PRINT CHR($(1),D),
>800 NEXT D
>RUN
BARRA DI FERRO
ORREF ID ARRAB
```

Nell' esempio sopra riportato, l' espressione contenuta all' interno della parentesi che segue l' operatore CHR ha lo stesso significato dell' espressione dell' operatore ASC().
A differenza dell' operatore ASC(), all' operatore CHR() non può essere assegnato alcun valore. Un' istruzione come CHR($(1),1)=H, é invalidata e genera un BAD SYNTAX ERROR. Quindi si può concludere che l' operatore ASC() é usato per modificare un valore in una stringa, mentre l' operatore CHR() può solo essere utilizzato all' interno di un' istruzione di PRINT.

FUNZIONI SPECIALI

Gli operatori speciali di funzione sono chiamati in questo modo perché consentono di manipolare direttamente le linee di I/O hardware e gli indirizzi di memoria della CPU.
Tutti gli operatori speciali di funzione, ad eccezione di CBY([espress.]) e GET, possono essere collocati su ambedue i lati del simbolo di uguale (=) in una istruzione LET.

Esempi:

C = DBY(225) e DBY(225) = C

Entrambi gli esempi appena visti sono istruzioni valide in BASIC 52. Passiamo ora alla descrizione di tutti gli operatori speciali di funzione del BASIC 52.

CBY([espress.])

L' operatore CBY([espress.]) é usato per recuperare i dati del programma o lo spazio di indirizzamento per i codici di memoria della CPU. A partire da quando il codice di memoria non viene più ad essere scritto sulla CPU, l' operatore CBY([espress.]) non può più assegnare alcun valore, ma può solamente essere letto.
Esempio:
C=CBY(5450)
Farà si che il valore collocato nello spazio di memoria di programma con indirizzo 5450, venga assegnato alla variabile C.
L' argomento per il quale é valido l' operatore CBY([espress.]) deve essere un intero valido compreso tra 0 e 65535 (0FFFFH), in caso contrario si incorre in un BAD ARGUMENT ERROR.

DBY([espress.])

L' operatore DBY([espress.]) é utilizzato per recuperare o assegnare un valore alla memoria interna dei dati della CPU. Sia il valore che l' argomento nell' operatore DBY devono essere compresi tra 0 e 255 inclusi. Questo perché ci sono solo 256 locazioni di memoria interna nella CPU ed un byte può rappresentare soltanto un numero compreso tra 0 e 255 inclusi.
Esempi:
C=DBY(D) e DBY(145) = CBY(5450)
Nel primo esempio si assegna alla variabile C il valore che é interno alla locazione di memoria D. Come é già stato detto D deve essere compreso tra 0 e 255. Nel secondo esempio viene caricata la locazione 145 della memoria interna con il medesimo contenuto che é nella locazione 5450 della memoria di programma.

XBY([espress.])

L' operatore XBY([espress.]) é usato per recuperare o assegnare un valore alla memoria dei dati esterna della CPU.
L' argomento contenuto tra la parentesi che segue XBY deve essere un intero valido compreso tra 0 e 65535 (0FFFFH), mentre il valore assegnato all' operatore XBY([espress.]) deve essere

compreso tra 0 e 255. In caso contrario si incorre in un BAD ARGUMENT ERROR.
Esempi:
XBY(3200H)=DBY(235) e C=XBY(0FAC2H)
Nel primo esempio si carica la locazione esterna di memoria 3200 con lo stesso contenuto della locazione di memoria interna 235.
Nel secondo esempio si pone la variabile C uguale al contenuto della locazione di memoria esterna 0FAC2H.

GET

L' operatore GET risulta essere utilizzato in modo di esecuzione. In modo comando risulterà sempre a zero. Il GET permette la lettura della tastiera e riceve tutti gli impulsi dalla tastiera stessa. Se un carattere é premuto sulla tastiera, il valore del carattere stesso sarà assegnato al GET. Dopo tale operazione, il GET é letto nel programma e gli si assegnerà il valore zero fino a che un altro carattere non viene spedito dalla tastiera.
Nel seguente esempio sarà stampata la rappresentazione decimale di ogni carattere digitato.
Esempio:

```
>100 X=GET
>200 IF X=0 THEN 100
>300 PRINT X
>400 GOTO 100
>RUN
78   (digitato il tasto "N")
73   (digitato il tasto "I")
52   (digitato il tasto "2")
32   (digitato il tasto "SPAZIO")
```

La ragione per la quale l' operatore GET può essere letto solo una volta prima che gli sia assegnato il valore di zero, é che questa operazione garantisce che il primo carattere entrato sia sempre letto, indipendentemente dalla posizione dell' operatore GET all' interno del programma.

IE

L' operatore IE é utilizzato per recuperare o assegnare un valore relativo al registro di funzioni speciali IE della CPU. Dal momento in cui il registro IE della CPU é un byte register (8 bit), il valore assegnato a IE deve essere compreso tra 0 e 255. Tale registro contiene anche un bit inutilizzato, BIT IE.6. Fino a che tale bit rimane indefinito, potrà essere letto a caso a 1 o a 0, così l' utente ha la possibilità di mascherare questo bit quando viene letto il registro IE. Questa operazione di mascheramento può essere fatta con una istruzione come C=IE.AND.0BFH. Le uniche istruzioni in BASIC 52 che possono essere scritte nel registro IE sono: CLOCK0, CLOCK1, ONEX1, CLEAR e CLEARI.
Esempi:
IE=32H e C=IE.AND.0BFH

IP

L' operatore IP é usato per recuperare o assegnare un valore al registro di funzioni speciali IP della CPU. Dal momento in cui il registro IP é un byte register, il valore ad esso assegnato dovrà essere compreso tra 0 e 255. In tale registro due bit sono inutilizzati BIT IP.6 ed IP.7. Fino a che questi due bit non sono definiti, possono essere letti a caso a 1 o a 0, così che l' utente può creare una maschera quando viene letto il registro IP. Tale mascheramento può essere effettuato con un' istruzione come C=IP.AND.3FH. Il BASIC 52 non scrive nel registro IP durante l' inizializzazione, così da permettere all' utente di stabilire qualsiasi priorità di interrupts richiesti in una applicazione specifica.

Esempi:

IP= 45 e C=IP.AND.3FH

PORT1

L' operatore PORT1 é utilizzato allo scopo di prelevare o assegnare un valore al port di I/O P1 della CPU. Il valore che può assumere il port P1 é variabile da 0 a 255 inclusi, in quanto sulla CPU é un byte register. Alcuni bit presenti su P1 hanno funzioni predefinite, infatti se l' utente non effettua nessuna operazione hardware riguardo a queste funzioni predefinite, l' istruzione PORT1 può essere usata in applicazioni nella maniera appropriata.

PCON

L' operatore PCON é utilizzato allo scopo di prelevare o assegnare un valore al registro PCON della CPU. Nella CPU, viene usato solo il bit più significativo del registro PCON, mentre tutti gli altri bit sono indefiniti. Settando questo bit raddoppierà il baud rate se il TIMER/COUNTER 1 é usato come generatore di baud rate per il port seriale. PCON é un byte register.

RCAP2

L' operatore RCAP2 assegna e/o preleva un valore ai registri di funzioni speciali RCAP2H ed RCAP2L della CPU. Questo operatore tratta RCAP2H ed RCAP2L in coppia come se fossero un unico registro a 16 bit. RCAP2H é il byte alto, mentre RCAP2L é il byte basso. Questi due registri sono dei registri reload/capture per il TIMER2. L' utente deve prestare molta attenzione quando scrive nel registro RCAP2, perché tale registro controlla il baud rate del port seriale del dispositivo BASIC 52. La seguente equazione consente di determinare con quale baud rate deve operare il BASIC 52:

BAUD = XTAL/(32*(65536-RACP2))

T2CON

Questo operatore speciale di funzione consente di prelevare e/o di far assumere un determinato valore al registro speciale di funzione T2CON. Il T2CON é un registro ad 8 bit che controlla il modo in cui deve operare il TIMER2 e che determina quale timer (TIMER1 oppure TIMER2) é usato come generatore di baud rate della CPU. Il BASIC 52 inizializza T2CON con il valore 52 (34H) che non viene mai modificato. Se lo si modifica a casaccio, cioé senza sapere la procedura da seguire, il port seriale della CPU si può danneggiare. Bisogna prestare molta attenzione a come viene utilizzato questo registro.

TCON

Tale operatore consente di prelevare e/o far assumere un determinato valore al registro di funzioni speciali TCON della CPU. TCON é un registro ad 8 bit il quale permette di abilitare o di disabilitare TIMER0 e TIMER1, più tutti gli interrupts che sono associati a questi due timer. Inoltre, TCON determina se i pin degli interruputs esterni sulla CPU stanno operando in un level sensitive oppure in modo edge-trigger.
Il BASIC 52 inizializza il registro TCON con il valore 244 (0F4H) ed evita che tale valore venga modificato. Il valore 244 (0F4H) pone sia il TIMER0 che il TIMER1 in run mode (abilitato). Se l' utente disabilita l' operazione del TIMER0, resettando il bit 4 del registro TCON, il Real Time Clock non sarà attivato. Se l' utente disabilita l' operazione del TIMER1, questo avviene resettando il bit 6 del registro TCON, non saranno attivate ne le routine di programmazione delle EPROM ne il software del port seriale, e nemmeno l' istruzione PWM. Anche per modificare questo registro bisogna prestare molta attenzione.

TMOD

L' operatore speciale TMOD permette di prelevare e/o di far assumere un valore fissato al registro di funzioni speciali TMOD della CPU. TMOD é un byte register che controlla il modo di operazione sia del TIMER0 che del TIMER1. Questo registro viene inizializzato dal BASIC 52 con il valore 16 (10H). Con questo valore si pone il TIMER0, che é un contatore a modulo 13, in modo 0 ed il TIMER1, che é un contatore a modulo 16, in modo 1. Il BASIC 52 fa sì che i modi di questi due timers/counters non siano mai modificati. Se l' utente effettua una modifica sul modo di TIMER0, il Real Time Clock non opererà correttamente. Se l' utente modifica il modo di TIMER1, non funzioneranno più correttamente, la programmazione di EPROM, il software del port seriale e l' istruzione PWM. Nel caso l' utente non desideri utilizzare queste caratteristiche disponibili in BASIC 52, egli può disporre di un altro timer/counter funzionante nel modo che si addice maggiormente alla applicazione specifica.

TIMER0

L' operatore TIMER0 viene utilizzato allo scopo di assegnare o prelevare un valore dai registri di funzioni speciali TH0 e TL0 della CPU. Questo operatore tratta i due byte registers TH0 e TL0 in coppia come se fossero un unico registro a 16 bit. TH0 é il byte alto, mentre TL0 é il byte basso. Il BASIC 52 usa TH0 e TL0 per implementare la funzione REAL TIME CLOCK. Se l' utente non implementa tale funzione nel programma BASIC (per esempio non utilizza l' istruzione CLOCK1), TH0 e TL0 potranno essere utilizzati per applicazioni particolari.

TIMER1

Questo operatore é utilizzato per assegnare oppure per prelevare un determinato valore dai registri di funzioni speciali TH1 e TL1 della CPU. TH1 e TL1 sono due registri ad 8 bit, che però l' operatore considera in coppia, come un registro a 16 bit. TH1 é il byte alto, mentre TL1 é il byte basso. Il BASIC 52 utilizza i due registri allo scopo di implementare la sincronizzazione del software del port seriale, le caratteristiche di programmazione delle EPROM e l' istruzione PWM. Nel caso l' utente non usi alcuna delle precedenti caratteristiche, TH1 e TL1 possono essere utilizzati in modi diversi, soprattutto per adattarsi alle richieste dell' utente.

TIMER2

L' operatore TIMER2 consente di prelevare o di assegnare un certo valore ai registri di funzioni speciali TH2 e TL2 della CPU. Anche questi due registri sono ad 8 bit, ma l' operatore li vede accoppiati in un unico registro a 16 bit. TH2 é il byte alto e TL2 é il byte basso. TH2 e TL2 vengono utilizzati dal BASIC 52 per generare il baud rate per il port seriale. Se l' operatore TIMER2 non viene utilizzato allo scopo di generare il baud rate del port seriale, TH2 e TL2 possono essere usati per esigenze tipiche dell' utente.

TIME

L' operatore TIME permette di prelevare e/o di far assumere un certo valore al REAL TIME CLOCK presente nel BASIC 52. Subito dopo il reset, il TIME é uguale a 0. L' istruzione CLOCK1 abilita il REAL TIME CLOCK. Quando il REAL TIME CLOCK é già stato abilitato, l' operatore speciale di funzione TIME incrementerà una volta ogni 5 millisecondi. L' operatore TIME utilizza il TIMER0 e gli interrupts ad esso associati. L' unità dell' operatore TIME sono i secondi ed un appropriato valore di XTAL deve essere assegnato per assicurare che l' operatore TIME sia corretto.
Quando a TIME viene assegnato un valore con una istruzione LET (ad esempio TIME=40), verrà modificata solo una porzione intera di TIME.
Esempio:

```
>CLOCK1          (abilitazione del Real Time Clock)
>CLOCK0          (disabilitazione del Real Time Clock)
>PRINT TIME      (visualizzazione del TIME)
5.425
>TIME=0          (il TIME é fissato uguale a 0)
>PRINT TIME      (visualizzazione del TIME)
.425             (é variata solo la porzione intera)
```

La frazione della porzione intera di TIME può essere modificata manipolando i contenuti della locazione di memoria interna 71 (47H). Questa operazione viene realizzata da una istruzione come DBY(71). É importante notare che ogni conteggio nella locazione di memoria interna 71 (47H) rappresenta 5 millisecondi di TIME. Vediamo ancora l' esempio:

```
>DBY(71)=0       (frazione di TIME uguale a 0)
>PRINT TIME
0
>DBY(71)=5       (frazione di TIME uguale a 5, cioé 25 ms)
>PRINT TIME
2.5 E-2
```

L' unica ragione per la quale l' intero della porzione di TIME é cambiato quando gli é stato assegnato un valore, é che esso permette all' utente di generare intervalli di tempo precisi. Per esempio vogliamo creare un clock di 12 ore. In 12 ore ci sono 43200 secondi, così verrà utilizzata l' istruzione ONTIME 43200,[numero intero]. A questo punto quando l' interrupt TIME incorre nell' esecuzione dell' istruzione TIME=0, ma il contatore dei millisecondi non riassegna alcun valore così se si verifica l' interrupt latente eccede di 5 millisecondi, il clock manterrà ancora la sua precisione.

XTAL

L' operatore XTAL indica al BASIC 52 a che frequenza é operante il sistema. L' operatore XTAL é utilizzato dal BASIC 52 per calcolare il valore ricaricato del REAL TIME CLOCK, la temporizzazione per la programmazione delle PROM e la generazione del baud rate da parte del software per il port seriale. Il valore di XTAL viene espresso in Hz. Così:
XTAL=9000000 : REM Setta il valore di XTAL a 9 MHz.

ESEMPI GENERALI SULLE FUNZIONI SPECIALI

Usando gli operatori logici disponibili in BASIC 52, é possibile scrivere o leggere i byte dei registri speciali di funzione che il BASIC 52 considera come una coppia di registri:

Esempio 1:
SCRITTURA DEL BYTE ALTO
>TIMER0 = (TIMER0 .AND. 00FFH) + INT(256*(BYTE DELL' UTENTE))

Esempio 2:
SCRITTURA DEL BYTE BASSO
>TIMER0 = (TIMER0 .AND. 0FF00H) + (BYTE DELL' UTENTE)

Esempio 3:
LETTURA DEL BYTE ALTO
>PH0. INT(TIMER0/256)

Esempio 4:
LETTURA DEL BYTE BASSO
>PH0. TIMER.AND.0FFH

Il TIMER1 può funzionare come generatore di baud rate per il BASIC 52. Per consentire a TIMER1 di agire come generatore di baud rate, devono essere eseguite le seguenti istruzioni:

>TMOD = 32 (TIMER1 in modo di autoricaricamento)

>TIMER1 = 256*(256-(65536-RCAP2)/12) (Caricamento del TIMER1)

>T2C0N = 0 (TIMER1 utilizzato come generatore di baud rate)

La sequenza di istruzioni appena riportata può essere eseguita tanto in modo diretto quanto come parte di programma. Quando TIMER1 é utilizzato come generatore di baud rate, TIMER2 può essere destinato ad altri utilizzi. I comandi e le istruzioni come PROG, FPROG, LIST#, PRINT# e PWM non possono essere utilizzati quando TIMER1 é usato come generatore di baud rate per il dispositivo BASIC 52. I cristalli possono non essere in grado di permettere l' uso di TIMER1 come generatore di baud rate, specialmente per baud rates elevati (superiori a 2400).

VALORI PER IL CONTROLLO DEL SISTEMA

I valori di controllo del sistema determinano o rivelano come é stata allocata la memoria dal BASIC 52.

MTOP

Dopo il reset, Il BASIC 52 misura la memoria esterna ed assegna l' ultimo indirizzo di memoria valido al valore di controllo del sistema MTOP. Il BASIC 52 non usa altra memoria RAM esterna oltre il valore assegnato ad MTOP. Se l' utente vuole allocare una parte di memoria esterna per una routine in linguaggio assembly, può fare uso dell' istruzione LET (ad esempio MTOP = Indirizzo dell' Utente). Infine se l' utente assegna ad MTOP un valore che é più grande dell' ultimo indirizzo di memoria valido, si incorrerà in un MEMORY ALLOCATION ERROR.

Esempi:

```
>PRINT MTOP
8191

>MTOP=3500
>PRINT MTOP
3500
```

LEN

Il valore per il controllo del sistema LEN indica all' utente quanti bytes di memoria occupa il programma attualmente selezionato. Ovviamente LEN non può assegnare alcun valore, ma può soltanto essere letto. Un programma nullo riporterà il valore di LEN a 1. 1 rappresenta la fine dell' ultimo carattere di un programma.

FREE

Il valore per il controllo del sistema FREE indica all' utente quanti bytes di memoria RAM ha ancora a disposizione. Quando il programma selezionato si trova nella memoria RAM, sarà valida la seguente relazione:

FREE = MTOP - LEN - 511

Nota: A differenza di altri BASICs, il BASIC 52 non richiede alcun argomento per i valori di controllo del sistema.

MESSAGGI ERRORE

Il BASIC 52 ha un processore di errore relativamente sofisticato. Quando il BASIC é in modo di esecuzione, la forma generalizzata di un messaggio errore é la seguente:

ERROR: AAA - IN LINE BBB

BBB ISTRUZIONE BASIC
-------------------------X

Dove AAA é il tipo di errore e BBB é il numero di linea in cui é contenuto l' errore. Vediamo ora un esempio specifico:

ERROR: BAD SYNTAX - IN LINE 200

200 PRINT 240+23*5.76*
----------------------------------X

La X indica approssimativamente la posizione dell' errore nella linea. La posizione della X può essere di uno o due caratteri o espressioni spostata rispetto all' errore, in quanto dipende dal tipo di errore che é stato commesso e dalla sua posizione nel programma. Se si incorre in un errore in modo comando viene visualizzato solo il tipo di errore commesso e non il numero di linea. Questo avviene perché in modo comando non ci sono i numeri di linea. I tipi i Errore sono i seguenti:

BAD SYNTAX

Un Bad Syntax Error significa che é entrato un comando, o un' istruzione oppure un operatore che risulta essere invalido per il BASIC 52, e che quindi non é in grado di riconoscere. L' utente può quindi verificare ed essere sicuro che ogni cosa sia stata inserita correttamente. Nella versione 1.1 del BASIC 52 un Bad Syntax Error si genera anche se il programmatore prova ad usare una parola chiave riservata come parte di una variabile.

BAD ARGUMENT

Nel caso l' argomento di un operatore non sia compreso all' interno del limite dell' operatore, si genera un Bad Argument Error. Per esempio, DBY(280) genera un Bad Argument Error perché l' argomento dell' operatore DBY deve essere compreso tra 0 e 255 inclusi. In modo analogo si genera un Bad Argument Error nel caso di XBY(0FAC4H)=-2, perché il valore assunto dall' operatore XBY é compreso tra 0 e 255.

ARITH. UNDERFLOW

Se il risultato di una operazione aritmetica eccede oltre il limite minimo di un numero a virgola mobile del BASIC 52, si genera un Arith. Underflow Error. Il numero a virgola mobile più piccolo in BASIC 52 é +/- 1E-127. Per esempio 1E-70/1E+70 provoca un Arith. Underflow Error.

ARITH. OVERFLOW

Un Arith. Overflow Error si genera se il risultato di una operazione aritmetica eccede il limite superiore di un numero a virgola mobile del BASIC 52. Il numero a virgola mobile più grande in BASIC 52 é +/-.99999999E+127. Per esempio, con il numero 1E+90*1E+90 si genera un Arith. Overflow Error.

DIVIDE BY ZERO

Una divisione di un qualsiasi numero per ZERO, provoca un DIVIDE BY ZERO ERROR. Per esempio 75/0, ecc.

NO DATA

Se é stata eseguita un' istruzione READ e non esiste un' istruzione DATA o se sono già stati letti tutti i DATA e non é stata eseguita un' istruzione RESTORE, comparirà sul video il seguente messaggio errore:
ERROR: NO DATA - IN LINE AAA

CAN' T CONTINUE

Un programma in esecuzione può essere fermato premendo un control-C sulla tastiera oppure eseguendo un' istruzione di STOP. Normalmente il programma in esecuzione può essere recuperato eseguendo un comando di CONT. Comunque, se l' utente edita il programma dopo averne fermato l' esecuzione e dopo che é stato inserito il comando CONT, si verifica un CAN' T CONTINUE ERROR. Per arrestare l' esecuzione di un programma, deve essere digitato un control-C durante l' esecuzione del programma, oppure STOP deve essere eseguita un' istruzione STOP prima dell' attivazione del comando CONT.

PROGRAMMING

Se si incorre in un errore mentre il dispositivo BASIC 52 sta programmando una EPROM, si verifica un PROGRAMMING ERROR. Un errore incontrato durante la programmazione distrugge la struttura dell' EPROM file, così l' utente non può salvare altri programmi su quella specifica EPROM una volta che si incorre in un PROGRAMMING ERROR.

A-STACK

Un errore di A-STACK (Argument Stack) é presente quando l' argomento dello stack pointer é forzato ad uscire dai limiti. Questo può accadere se l' utente eccede nel riempire l' argument stack introducendo troppe espressioni sullo stack stesso, oppure nel tentativo di far uscire dallo stack, con l' istruzione POP, dei dati non presenti.

C-STACK

Un errore di C-STACK (Control Stack) é provocato se lo stack pointer é forzato ad uscire dai propri limiti. 158 bytes della memoria esterna sono allocati per il control stack, 17 bytes sono richiesti per il ciclo FOR-NEXT, mentre 3 bytes vengono richiesti per DO-UNTIL, DO-WHILE e GOSUB.Questo significa che 9 inserimenti di cicli FOR-NEXT sono il massimo che il BASIC 52 può manipolare, poiché 9 volte 17 bytes sono 153 bytes. Se l' utente cerca di usare più spazio del control stack di quello disponibile in BASIC 52, verrà generato un C-STACK ERROR. Inoltre si incorre in un C-STACK ERROR nel caso in cui un RETURN viene eseguito prima di un GOSUB, un WHILE o un UNTIL prima di un DO, oppure un NEXT prima di un FOR.

I-STACK

Un errore I-STACK (Internal Stack) é provocato quando il BASIC 52 non ha abbastanza spazio di stack per valutare l' espressione. Normalmente non si incorre in un errore di I-STACK, a meno che non sia stata allocata memoria insufficiente per lo stack pointer della CPU.

ARRAY SIZE

Se un vettore é dimensionato da un' istruzione DIM e si cerca di accedere ad una variabile che é all' esterno dei limiti di dimensionamento, si incorre in un errore di ARRAY SIZE.

Esempio:

```
>DIM A(5)
>PRINT A(7)

ERROR: ARRAY SIZE
READY
```

MEMORY ALLOCATION

Un errore di MEMORY ALLOCATION viene generato quando l' utente cerca di accedere a stringhe che sono esterne al limite definito. Inoltre, se al valore di controllo del sistema MTOP é assegnato un valore che non é contenuto in alcuna memoria RAM, si incorre ancora in un errore MEMORY ALLOCATION.

APPENDICE A: RIFERIMENTI RAPIDI

COMANDI:

COMANDI	FUNZIONE	ESEMPI
RUN	Esegue un programma.	RUN
CONT	Prosegue l' esecuzione dopo uno stop o un control-c.	CONT
LIST	Visualizza sul video il listato del programma.	LIST LIST 200-600
LIST#	Stampa il listato del programma sulla stampante.	LIST# LIST# 100
LIST@	Pone il listato del programma sul driver utente.	LIST@ LIST@ 100
NEW	Cancella il programma immagazzinato in RAM.	NEW
NULL	Fissa i caratteri nulli dopo ogni "carriage return/line feed.	NULL
RAM	Richiama il modo RAM, il programma si trova in RAM.	RAM
ROM	Richiama il modo ROM, il programma si trova in memoria non volatile.	ROM ROM 5
XFER	Trasferisce un programma da memoria non volatile a RAM.	XFER
PROG FPROG	Salva il programma attuale in memoria non volatile.	PROG
PROG1 FPROG1	Salva in memoria non volatile le informazioni riguardanti il baud-rate.	PROG1
PROG2 FPROG2	Salva in memoria non volatile le informazioni riguardanti il baud-rate e l' auto-run.	PROG2
PROG3 FPROG3	Salva in memoria non volatile le informazioni riguardanti il baud-rate e il MTOP.	PROG3
PROG4 FPROG4	Salva in memoria non volatile le informazioni riguardanti il baud-rate, il MTOP e l' auto-run.	PROG4
PROG5 FPROG5	Come PROG4, eccetto che la RAM non si cancella con un reset o power-up, se la locazione esterna 05EH contiene il valore 0A5H.	PROG5
PROG6 FPROG6	Come PROG5, eccetto che dopo un reset o un power-on viene eseguito il codice presente a partire dalla locazione 04039H.	PROG6

ISTRUZIONI:

ISTRUZIONI	FUNZIONE	ESEMPI
BAUD	Setta il baud-rate per il port della stampante.	BAUD 9600
CALL	Chiama un programma in linguaggio assembly.	CALL 8000H
CLEAR	Cancella variabili, interrupts e stringhe.	CLEAR
CLEARI	Cancella gli STACKS.	CLEARI
CLEARS	Cancella gli interrupts.	CLEARS
CLOCK1	Abilita il REAL TIME CLOCK.	CLOCK1
CLOCK0	Disabilita il REAL TIME CLOCK.	CLOCK0
DATA	Indica i dati che devono essere letti dall' istruzione READ.	DATA 50
READ	Legge i dati dell' istruzione DATA.	READ I
RESTORE	Resetta il puntatore di lettura (READ).	RESTORE
DIM	Alloca la memoria per variabili vettoriali.	DIM A(10)
DO	Istruzione iniziale per i cicli con WHILE o UNTIL.	DO
UNTIL	Verifica la condizione del ciclo DO (si esegue il corpo del ciclo se la condizione non é verificata).	UNTIL A=50
WHILE	Verifica la condizione del ciclo DO (si esegue il corpo del ciclo se la condizione é verificata).	WHILE A=20
END	Ferma l' esecuzione del programma.	END
FOR-TO-{STEP}	Istruzioni iniziali per l' esecuzione del ciclo FOR-NEXT.	FOR X=0 TO 20
NEXT	Verifica le condizioni del ciclo FOR-NEXT.	NEXT X
GOSUB	Esegue una subroutine.	GOSUB 3500
RETURN	Ritorna dalla subroutine.	RETURN

ISTRUZIONI:

ISTRUZIONI	FUNZIONE	ESEMPI
GOTO	L' esecuzione salta al numero di linea specificato.	GOTO 1000
ON GOTO	GOTO condizionale.	ON X GOTO 70,
ON GOSUB	GOSUB condizionale.	ON X GOSUB 5,
IF-THEN-{ELSE}	Se la condizione é verificata si eseguono le istruzioni dopo THEN, altrimenti quelle dopo l' ELSE.	IF X<Y THEN 5
INPUT	INPUT di una stringa o di una variabile.	INPUT X
LET	Assegna un valore a una stringa o a una variabile.	LET X=34
ONERR	Salta al numero di linea specificato, in caso di errori aritmetici.	ONERR 300
ONTIME	Salta al numero di linea specificato, quando TIME é uguale o maggiore all' argomento di ONTIME.	ONTIME 5,200
ONEX1	Si genera un GOSUB al numero di linea specificato, quando il pin INT1 è basso.	ONEX1 200
PRINT	Stampa variabili, stringhe ecc. (P. o ? sono forme contratte di PRINT).	PRINT X
PRINT#	Stampa sulla stampante.	PRINT# X
PH0.	Stampa valori esadecimali eliminando gli zeri.	PH0. X
PH1.	Stampa valori esadecimali lasciando gli zeri.	PH1. X
PH0.#	PH0. sulla stampante.	PH0.# X
PH1.#	PH1. sulla stampante.	PH1.# X
PRINT@	Stampa sul driver definito dall' utente.	PRINT@ 5*5
PH0.@	PH0. sul driver definito dall' utente.	PH0.@ X
PH1.@	PH1. sul driver definito dall' utente.	PH1.@ X
PGM	Programma nella memoria non volatile.	PGM

ISTRUZIONI:

ISTRUZIONI	FUNZIONE	ESEMPI
PUSH	Salva dei valori nell' argument stack.	PUSH 50,X
POP	Salva dei valori dall' argument stack.	POP X,Y,Z
PWM	Effettua un PWM.	PWM 10,10,500
REM	Permette di aggiungere commenti alle linee del programma.	REM Stampa
RETI	Ritorna dall' interrupt.	RETI
STOP	Blocca l' esecuzione del programma.	STOP
STRING	Alloca memoria per le stringhe.	STRING 110,10
UI1	Richiama una routine di INPUT definita dall' utente.	UI1
UI0	Richiama la routine di INPUT di default.	UI0
UO1	Richiama una routine di OUTPUT definita dall' utente.	UO1
UO0	Richiama la routine di OUTPUT di default.	UO0
ST@	Immagazzina dei numeri in virgola mobile nella parte superiore dello stack a partire dalla locazione specificata.	ST@ 2000H
LD@	Estrae dei numeri in virgola mobile dalla parte superiore dello stack a partire dalla locazione specificata.	LD@ X
IDLE	Si attende che intervenga un interrupt.	IDLE
RROM	Esegue un programma in memoria non volatile.	RROM 5

OPERATORI A DUE OPERANDI:

OPERATORI	FUNZIONE	ESEMPI
+	Addizione.	3+2
/	Divisione.	50/25
**	Esponenziale.	3**3
*	Moltiplicazione.	10*10
-	Sottrazione.	14-5
AND	AND logico.	6.AND.3
OR	OR logico.	7.OR.8
XOR	XOR logico.	4.XOR.5

OPERATORI AD UN OPERANDO:

OPERATORI	FUNZIONE	ESEMPI
ABS()	Calcolo del valore assoluto.	ABS(-5)
NOT()	Calcolo del complemento ad uno.	NOT(32)
INT()	Calcola l' intero.	INT(27.9)
SGN()	Determina il segno.	SGN(9)
SQR()	Calcola la radice quadrata.	SQR(25)
RND	Genera un numero a caso.	RND
LOG()	Calcola il logaritmo naturale	LOG(100)
EXP()	Calcolo di "e" (2.7182818) elevato alla X.	EXP(5)
SIN()	Calcolo del seno.	SIN((3/2)*PI)
COS()	Calcolo del coseno.	COS(PI)
TAN()	Calcolo della tangente.	TAN(1)
ATN()	Calcolo dell' arcotangente.	ATN(.707)

OPERATORI SPECIALI DI FUNZIONE:

OPERATORI	FUNZIONE	ESEMPI
CBY()	Legge l' area codice.	P. CBY(5450)
DBY()	Gestisce la ram interna.	DBY(145)=X
XBY()	Gestisce la ram esterna.	P. XBY(6000)
GET	Legge dal dispositivo di input selezionato.	P. GET
IE	Gestisce il registro interno della CPU "IE".	IE=032H
IP	Gestisce il registro interno della CPU "IP".	IP=45
PORT1	Gestisce il registro interno della CPU "PORT1".	PORT1=0D3H
PCON	Gestisce il registro interno della CPU "PCON".	PCON=080H
RCAP2	Gestisce i registri interni della CPU "RCAP2H e RCAP2L".	RCAP2=145
T2CON	Gestisce il registro interno della CPU "T2CON".	P. T2CON
TCON	Gestisce il registro interno della CPU "TCON".	TCON=028H
TMOD	Gestisce il registro interno della CPU "TMOD".	P. TMOD
TIME	Gestisce il REAL TIME CLOCK.	P. TIME
TIMER0	Gestisce i registri interni della CPU "TH0 e TL0".	P. TIMER0
TIMER1	Gestisce i registri interni della CPU "TH1 e TL1".	TIMER1=0
TIMER2	Gestisce i registri interni della CPU "TH2 e TL2".	P. TIMER2

OPERATORI DI STRINGHE:

OPERATORI	FUNZIONE	ESEMPI
ASC()	Calcola il valore decimale del codice ASCII contenuto tra parentesi.	P. ASC(C)
CHR()	Calcola il codice ASCII del numero decimale contenuto tra parentesi.	P. CHR(67)

OPERATORI PER IL CONTROLLO DEL SISTEMA:

OPERATORI	FUNZIONE	ESEMPI
MTOP	Legge o assegna la quantità di RAM esterna che il BASIC-52 può utilizzare.	MTOP=3500 P. MTOP
LEN	Indica quanti bytes di memoria sono occupati dal programma attualmente selezionato.	P. LEN
FREE	Indica quanti bytes di memoria sono ancora liberi.	P. FREE

COSTANTI IMMAGAZZINATE:

COSTANTE	VALORE	ESEMPI
PI	3.1415926	P. PI

COMANDI	ISTRUZIONI	OPERATORI
RUN	BAUD	ADDIZIONE (+)
CONT	CALL	DIVISIONE (/)
LIST	CLEAR	ESPONENZIALE (**)
LIST#	CLEAR(S&I)	MOLTIPLICAZIONE (*)
LIST@	CLOCK(1&0)	SOTTRAZIONE (-)
NEW	DATA	AND LOGICO (.AND.)
NULL	READ	OR LOGICO (.OR.)
RAM	RESTORE	X-OR LOGICO (.XOR.)
ROM	DIM	NOT LOGICO (NOT())
XFER	DO-WHILE	ABS()
PROG	DO-UNTIL	INT()
PROG1	END	SGN()
PROG2	FOR-TO-STEP	SQR()
PROG3	NEXT	RND()
PROG4	GOSUB	LOG()
PROG5	RETURN	EXP()
PROG6	GOTO	SIN()
FPROG	ON-GOTO	COS()
FPROG1	ON-GOSUB	TAN()
FPROG2	IF-THEN-ELSE	ATN()
FPROG3	INPUT	=,>,>=,<,<=,<>
FPROG4	LET	ASC()
FPROG5	ONERR	CHR()
FPROG6	ONEX1	CBY()
	ONTIME	DBY()
	PRINT	XBY()
	PRINT#	GET
	PRINT@	IE
	PH0.	IP
	PH0.#	PORT1
	PH0.@	PCON
	PH1.	RCAP2
	PH1.#	T2CON
	PH1.@	TCON
	PGM	TMOD
	PUSH	TIME
	POP	TIMER0
	PWM	TIMER1
	REM	TIMER2
	RETI	XTAL
	STOP	MTOP
	STRING	LEN
	UI(1&0)	FREE
	UO(1&0)	PI
	LD@	
	ST@	
	IDLE	
	RROM	

APPENDICE B: INDICE ANALITICO